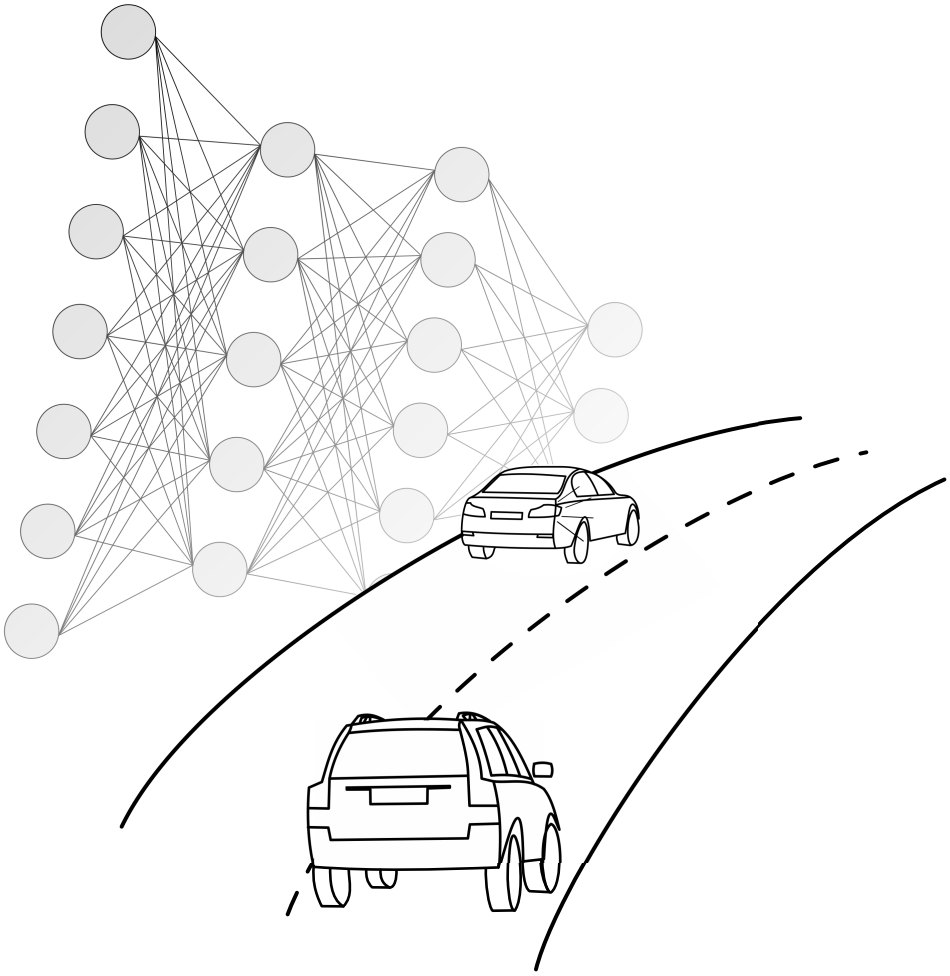
CHALMERS
UNIVERSITY OF TECHNOLOGY



Predicting Cut-Ins in Traffic Using a Neural Network

Master's thesis in Systems, Control and Mechatronics

Tonja Heinemann

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2017

Master's Thesis 2017

Predicting Cut-Ins in Traffic Using a Neural Network

Tonja Heinemann

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Göteborg, Sweden 2017

Predicting Cut-Ins in Traffic Using a Neural Network
Tonja Heinemann

Supervisor: Tommy Tram, Zenuity
Examiner: Jonas Sjöberg, Professor, Department of Electrical Engineering, Chalmers University of Technology

Master's Thesis 2017
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 31 772 1000

Cover: Structure of a Fully-Connected Multilayer Perceptron Behind a Cut-In Scenario on a Road.

Typeset in LaTeX
Göteborg, Sweden 2017

Acknowledgements

This thesis would not have been possible without the help of several key people. I appreciate their kind support and want to show my gratitude to all of them in this place.

First and foremost, I want to thank my supervisor Tommy Tram at Zenuity and supervisor and examiner Jonas Sjöberg at Chalmers for their support and guidance. They helped me to gain insight in both the research and usage of neural network algorithms in practice as well as in the professional world in general.

I also want to thank Mats Lestander and Mohammad Ali for their support and guidance. The filescan script from Cristina Westermark, along with her instructions how to use it brougt this thesis clearly forward, I am really grateful for it.

The work has been carried out at Volvo Cars Corporation in close collaboration with Zenuity. All sampled data used in the experiments in this work has been collected and provideded by Volvo Cars Corporation, and so have many important resources and tools. Thus, I also want to show my acknowledgements for Volvo Car Corporation and all the people there and at Zenuity supporting me, that have not been mentioned so far.

Tonja Heinemann

Predicting Cut-Ins in Traffic Using a Neural Network
Tonja Heinemann
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This work deals with the task of detecting a cut-in performed by an adjacent vehicle in front of the ego vehicle. The perception of the traffic scenario is based on fused sensor data from radar and camera available to a car, without car-to-car-communication. An algorithm to perform the traffic scenario detection, based on a neural network is developed. It is shown that a Multilayer Perceptron is able to predict a cut-in, when trained using supervised learning with backpropagation. The prediction of cut-ins can be used in an autonomous car as a basis for decisions regarding its driving behaviour and eventual precautionary safety measures. It can additionally be of use in a development situation to search the previously recorded and stored sensor data for cut-in scenarios. Training data for the neural network is taken from recorded sensor data of driving situations on public roads. The composition of a neural network training dataset from the recorded data is also handled in this thesis. For training, the true and false examples are identified in the recorded data and combined in the right ratio. The true examples are related to a cut-in, the false ones aren't. From both classes, a similar number of examples needs to be taken. Concerning the features given to the algorithm as input, both a dataset containing a high number of features and one with selective features are taken into consideration. In this work, it is shown that cut-ins can be detected 2.5 seconds in advance with an accuracy of 96% on training data. These results suggest that the prediction of cut-in scenarios is possible with the data available in this thesis and a neural network algorithm. Capabilities of the detection and possible improvements to the algorithm are discussed. Directions to prepare appropriate training data are given.

Keywords: Machine Learning, Neural Networks, Autonomous Drive, Cut-in Detection

Contents

List of Figures

List of Tables

Acronyms

NN Neural Network

RNN Recurrent Neural Network

FFNN Feed-Forward Neural Network

MLP Multilayer Perceptron

SVM Support Vector Machines

VCC Volvo Car Corporation

ACC Adaptive Cruise Control

Definitions

This itemisation specifies the terms and concepts used in this thesis.

Adaptive Cruise Control Adaptive Cruise Control enables a vehicle to keep the same speed and at the same time keep a safe distance from the vehicle in front of it.

Cut-In A cut-in happens if a vehicle on an adjacent lane changes lanes onto the ego vehicle's lane and is then positioned in front of the ego vehicle.

Dataset A set of samples, that is used to train a Neural Network.

Ego Vehicle From this vehicle's point of view, the sensors perceive the world. The cut-in is predicted from these sensor's data.

Feature In the scope of this work, a feature (in the data) refers to one specific physical or logical value assigned to a defined entity (e.g. the speed of the ego vehicle).

Feed-Forward Neural Network In a Feed-Forward Neural Network, the signal flow of an input pattern is always directed from the input to the output, never reversed. The Term Feed-Forward Neural Network is used as an equivalent to the term Multilayer Perceptron throughout this thesis.

Ground Truth In this work a reliable indication if a cut-in has happened or not.

Hidden Neurons/Units Units that process information within the neural network. Hidden neurons are neither input nor output units.

Hyperparameters Variable values of a network training algorithm, e.g. step-size, learning rate, steepness of the activation function, number of neurons in the hidden layers

Input Neurons/Units Input Neurons are the interface of the Neural Network (NN) for the input values.

Log, Record Captured sensor data, stored in a data structure containing multiple sensor values at multiple continuous time steps.

Multilayer Perceptron See Feed-Forward Neural Network.

Neural Network In this work, Neural Networks are used as input-output mapping function approximators. They are designed to be able to learn the function they should represent from training data.

Neuron In this work, the term neuron refers to the mathematical unit used in Neural Network rather than to the biological nerve cell.

Output Neurons/Units Output Neurons get the inputs to their calculations from other neurons, and calculate the networks output from them.

Recurrent Neural Network A Recurrent Neural Network's network graph contains cycles. This means the network saves some neurons' states and inputs them back into the network again.

Sample A set of values taken from different features at the same time step.

Self-driving/Autonomous Car/Vehicle A self-driving or autonomous vehicle participates in traffic without a driver's input. It perceives its surroundings via sensors. For this thesis, it is important that the autonomous car's sensor data includes positions of other vehicles on the road and the geometry of the road itself.

Target Value The desired output of the Neural Network, assigned per sample.

Target Vehicle In the context of this thesis: the car (likely to be soon) performing a cut-in.

Training Iteration One weight update performed by the Neural Network learning algorithm. In this thesis: presentation of the whole batch of training samples to the Neural Network followed by calculating the average squared error and updating the weights once.

1 Introduction to Cut-In Prediction

This section intends to explain the topic of this work and its context to self-driving cars. It shows how an algorithm to detect cut-in scenarios is beneficial. Possible sustainability effects are reflected briefly.

1.1 Benefits of Cut-In Prediction with a Neural Network

A vehicle in traffic always has to adapt to is surroundings. One very important action for an autonomous car is to follow the car in front of itself [1, 2, 3].
In order to use any kind of car-following model in traffic, the leading car needs to be identified. This is especially important if a leading car appears or vanishes. If it appears, the ego vehicle's velocity and distance needs to be adjusted to it; if it vanishes, the velocity can no longer depend on the leading vehicle. An autonomous car needs to be able to handle these situations, thus to predict them. Detecting another driver's intent to cut in in front of the ego vehicle as early as possible is beneficial to an autonomous car, as it gains the ability to react early on to the event.
An algorithm that can detect cut-ins happening on the road is available at Volvo Car Corporation (VCC). It is capable of detecting a cut-in less than half a second before the respective vehicle passes the lane marker. This algorithm is explained in greater detail in Section 2.1. Another algorithm, that predicts if a car on an adjacent lane is likely to cut in, is a valuable addition to those existing functions. With such an algorithm on hand, a self-driving car can get a broader context of its surroundings and thus a better base for its driving behaviour and decisions.
In addition to the use in a car, such an algorithm might be used in development to scan the recordings of sensor data for corresponding scenarios, so to perform post-processing. State-of-the-art algorithms to scan those recordings are scripts based on logical relations of parameters within the records. Their biggest drawback is that they need to be rewritten if for example the parameters in the records change. Writing and testing such a script takes time in every iteration, additionally they bear a certain complexity that needs to be overcome when working with these scanning algorithms. A NN avoids the inconvenience of rewriting such an algorithm as it can be re-trained with new parameters. The NN algorithms used in this work were able to detect cut-in scenarios in recorded time-series data.

1.2 Definition of a Cut-In Scenario

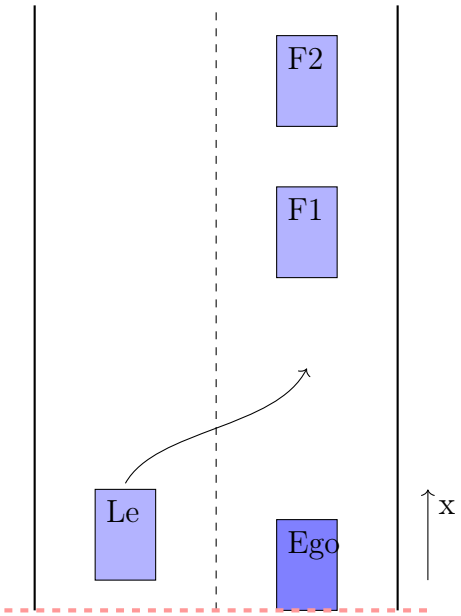


Figure 1: Cut-In Scenario

Figure 1 shows a bird's eye view of a cut-in situation as it is addressed in this work. The boxes represent vehicles heading in x-direction. In the scenario pictured, the ego vehicle follows F1, the first car in front of it and F2, the car in front of F1. The vehicle to the left of the ego vehicle, marked Le, wants to perform a lane change and position itself in front of the ego vehicle. This action is referred to as a cut-in throughout this work. The moment the Le car's centre passes the lane marker to the left of the ego vehicle is taken as the instance of the event happening.
Even though not shown in the figure, a similar manoeuvre performed from the lane to the right of the ego vehicle is also dealt with in this thesis. Throughout the thesis, there is no differentiation if a cut-in happens from the right or the left.

1.3 Approach to Predict Cut-Ins

In this thesis, a Neural Network (NN) is trained so it can predict cut-in scenarios. The algorithmic concept for the NN is a Feed-Forward Neural Network (FFNN) trained with the backpropagation algorithm. The background to the Multilayer Perceptron (MLP) can be found in Section 3.3, the mathematical background to the backpropagation algorithm is covered in Section 3.4. This NN classifies its input data in two groups: the 'positive' group referring to a cut-in and the 'negative' group, referring to no cut-in. Each

data point given to the NN consists of values gathered at a single timestep, referred to as a sample.
In order to predict a cut-in, the network is expected to produce a positive output during a given timespan before a cut-in until the cut-in event and a negative output for all samples not in this timespan. This is explained further in Section 4.1.
In order to choose the input features to the NN, two approaches are made in this thesis. The first one uses many input features that might be valuable to the classification task. In this thesis, this approach to input features could not generate satisfying results. The second way to assign features in this thesis is to select some features that are clearly correlating with the cut-in scenario. Using this approach, the cut-in could be predicted in advance. Section 4 goes into detail about the expected target of the NN and the selection of the features.
The training dataset for the NN is generated from recorded data. Section 5.1 focuses on how this dataset generation is implemented in this thesis. The training dataset contains both positive and negative samples in similar orders of magnitude. So it contains approximately equally many datapoints where an upcoming cut-in should be detected as datapoints that are not related to a cut-in. Two algorithms to locate the positive scenarios in the logged data are considered. The first one is based on the velocities and positions of the vehicles in the respective time instance, it's result is part of the logged data. The second algorithm scans the recorded data for vehicles, whose position changes from an adjacent lane to the leading vehicle position. This second algorithm is able to make use of the knowledge of the whole scenario, while the first one can not know the outcome of the scenario it detects. Those two algorithms are focused on in Section 2. In this thesis, valuable results could be generated using the second algorithm. A description of the tests, including setup and results, can be found in Section 6.

1.4 Influence of Cut-In Prediction in Vehicles on Society and Environment

This thesis is related to both self-driving cars and artificial intelligence, which are topics widely discussed in public today. One essential part of this discussion are variants of the trolley problem - making an acceptable decision in situations where humans can't find the best moral option with certainty (e.g. each possible option will result in deaths) [4]. The topic covered by this master thesis is considered a mechanism, that can provide a basis for such decisions, if not help to avoid critical situations. To ensure an unbiased

decision, the algorithm presented in this work should be reliable and work in the desired way.
Another important aspect is the transparency of the algorithm, the possibility to understand its operation [5]. This is the basis for both development and prohibition of erroneous behaviour. In case of the MLP used in this work, the number of free parameters, that are adapted during training, obstructs the calculation conducted. On the other hand, this algorithm performs a classification with two possible outcomes and a certain precision. Such a classification seldom reaches a perfect accuracy. Decisive functions that build on it need to take the boundaries of the classification's accuracy into account, regardless of the underlying algorithm.
These disconcerting considerations are opposed by the opportunities of self-driving cars. An algorithmic analysis of the traffic situation, e.g. a cut-in, allows for an optimization of fuel consumption. This, in return can result in reduced cost for mobility and fewer exhaust fumes (or other side effects of energy generation).
Self-driving vehicles can also enable those people to use a car, that are currently not able to drive themselves (e.g. physically impaired people, people on a certain medication or people to drunk or tired to drive). Such a car, and a cut-in detection in it, can also enhance overall safety on the road; it is e.g. not affected by tiredness, drugs or emotions as human drivers are. In comparison to the cut-in detection currently in use, a sooner detection can add comfort to the passengers.
As every piece of software, the presented algorithm will neither be perfect from the beginning nor the best option in every possible option of surroundings. The conditions it operates in, as well as the software it is part of, are subject to changes in ongoing development and an agile driving culture on the world's roads. For this reason, updates and changes to the software need to be included in the development process and guaranteed over the lifetime of any product.

1.5 Contributions

In this thesis, a non-time-dependent MLP is successfully trained to predict cut-in scenarios. The training data is taken from recorded real-world data; car-to-car communication is not considered. This thesis contributes to the prediction by introducing a set of input features that allows for achieving the task.
In this work, it was possible to push the detection boundary up to 2.5 seconds before the cut-in event.

1.6 Outline

In Section 2, the two existing algorithms that are used to locate cut-ins in the recording data are described. A closer look to the necessary variables describing the road geometry is taken.
The theory for MLP and their training is discussed in Section 3.
Section 4 discusses the target vector chosen in this thesis, the selection of the MLP as a network structure and the features chosen to input to the networks in the tests.
The method to construct the training dataset is outlined in Section 5, along with the specific parametrization of the NNs in the tests and the general test setup.
Section 6 gives the prerequisites, results and analyses to tests carried out with different approaches to compose the dataset and assign the target vector.
In Section 8, final conclusions are drawn. Reference points for future work are given.

2 Prerequisites to Cut-In Prediction

This section presents previous works using NN to interpret traffic scenarios. It also presents existing algorithms that have been used in this thesis to prepare the dataset for network training. At last, it describes a characteristic of the measured values that needs to be taken into account for their interpretation.

2.1 Algorithm to Detect Cut-Ins during Driving

When driving, there is an algorithm looking for cut-ins running in the car. Its output is stored in the log.
The calculation is based on the time, that a surrounding vehicle would take to cross the lane marker. This time is evaluated based on the velocity and the angle the respective vehicle is heading. If the expected time to cross the lane marker gets very small, the cut-in marker is set for the respective car. This is usually the case right before the vehicle actually crosses the lane marker, so the prediction happens less than a second in advance.
Instead of indicating just one moment of a cut-in, this value is often assigned in multiple consecutive samples.
On the downside, this detection relies on a recognisable lateral speed of the vehicle in question, which is not the case in all cut-in scenarios. So this method of detection leaves out some cases. Another drawback of this detection are false positives generated when the ego car performs a lane change.
The NN algorithm presented in this thesis allows to base the detection of a cut-in on more factors than the lateral speed of the target vehicle. The prediction of the cut-in can happen sooner with the NN. To eliminate the false positives due to ego car lane changes, the ego car's position is taken into account.

2.2 Algorithm to Scan the Logs for Cut-Ins

For postprocessing the recorded data, there exists a script to scan the logs for cut-ins at VCC. The vehicle in front of the ego car and the vehicles to the right and left can be identified in each sample. With this property a cut-in can be detected, if a vehicle is next to the ego car in one sample and in front of it in the following sample.
In order to be recognised as a cut-in by the VCC script, the following conditions have to be complied with:

- A vehicle to the left or right becomes the leading vehicle from one sample to the next
- The ego car does not perform a lane change during a specified time before and after the cut-in
- Tracking of the target vehicle is stable for a specified time before the cut-in
- Tracking of the leading car is stable for a specified time after the cut-in
- There are at least two lanes (to prevent lane merges to be classified as cut-ins)

Due to its rather restrictive policies concerning stability of the tracing of the vehicles, this algorithm is unable to detect all scenarios. On the other hand, the resulting instances feature clear and stable sensor values over the time of the event.

2.3 Peculiarities of the Parameters Measured on the Road

Among other variables, the longitudinal and lateral distances of vehicles on the road and their heading angle are measured by the ego vehicle. This measurement uses the ego cars centre as origin. Figure 2 shows these three parameters measured on a curvy road with respect to the vehicle on its left lane. The longitudinal and lateral distance are marked d_{lon} and d_{lat} and measured with respect to the ego car's driving direction. The dashed lines represent the lane markers. The angle α of the 'Le' car shows the difference in orientation to the ego car.

For vehicles driving on a straight road, a lateral distance to the left of the ego vehicle and a heading angle of zero show a vehicle driving on the left lane with the intention to stay there. If the Le vehicle intents a lane change, the lateral distance shrinks until it approaches zero, which would imply its position in front of the ego vehicle. During the manoeuvre, the Le car's heading angle is directed towards the ego car's lane, thus to the right.

If however the road is curved, as shown in Figure 2, these assumptions are no longer true. In the figure the Le vehicle is heading to the right and its lateral position is not even in front of the ego vehicle but to its right. However, the Le vehicle does not visibly intend to change to the ego lane, it simply follows the road.

In order to interpret this kind of scenario in the correct way, the road geometry has thus to be taken into account. Otherwise an algorithm like a NN is not able to make use of the measured positions.

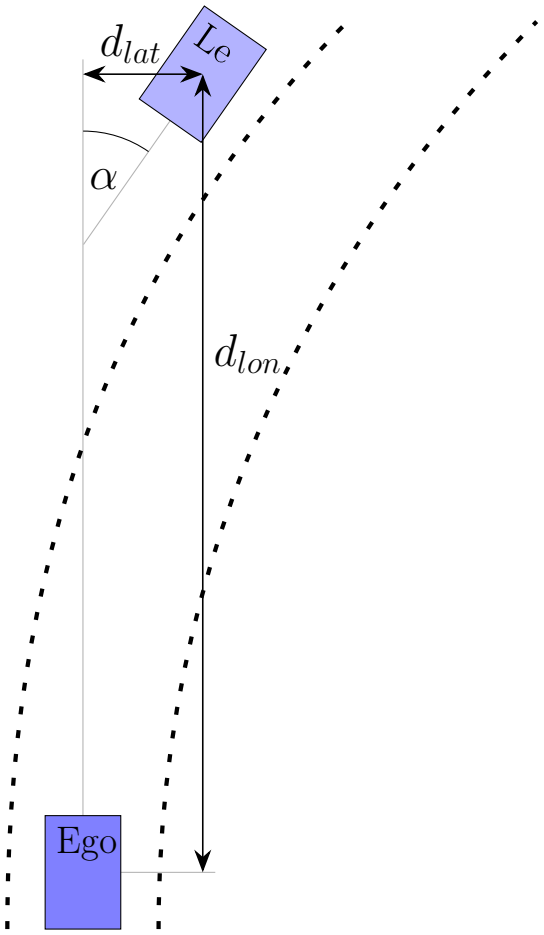


Figure 2: Positions of Vehicles on the Road as Measured by the Ego Vehicle

3 Fundamentals of Feed-Forward Neural Networks

This chapter summarizes the necessary mathematical concepts the thesis uses. It provides a brief overview of the Neural Network algorithms used to gain the results in Section 6.

3.1 Properties of Neural Networks

NN are data-driven, self-adaptive methods to approximate a function [6]. This means, they do not need a specification of the underlying model, which makes them a flexible method to describe a system. NN can approximate any function with arbitrary accuracy [6]. In the case of cut-in scenarios on the road, this is an advantage, as models involving e.g. more passive drivers as well as rather aggressive ones soon become complex.
The nonlinear nature of NN allows for modelling naturally nonlinear systems [7]. Their adaptivity is of advantage, as it allows for a fast adjustment to changing conditions. So if for example in the future the average driving behaviour changes, possibly due to new regulations or a rising number of (semi-) autonomous vehicles, the NN can be retrained without being rebuilt.
In neural network computing, two approaches to train a NN are present: supervised and unsupervised learning. In supervised learning, the network is either presented the correct answer (teacher-based) or a measure of its performance (reinforcement learning). Based on this input, it can adjust its internal structure to improve its operation.
In unsupervised learning, the network is only given the data itself and a learning rule, but no specification how its output should look like. The learning rule for those algorithms allows them to learn the familiarity or similarity of the training data. This learning paradigm is e.g. used to create topological maps from the input data.
This thesis will use teacher-based supervised learning to adapt an input-output mapping function to predict cut-ins. The specific network to be used is a Multilayer Perceptron.

3.2 Model of a Neuron

The neurons used for computations are inspired by biological neurons. A Neuron in the brain gets excited if the signals it gathers at its receptors exceed a certain intensity level. These signals usually have their origin in the activity of other neurons. If it gets excited, it will send out signals itself to be collected by other neurons. The mathematical adaptation of neurons was introduced in 1943 by McCulloch and Pitts [8].

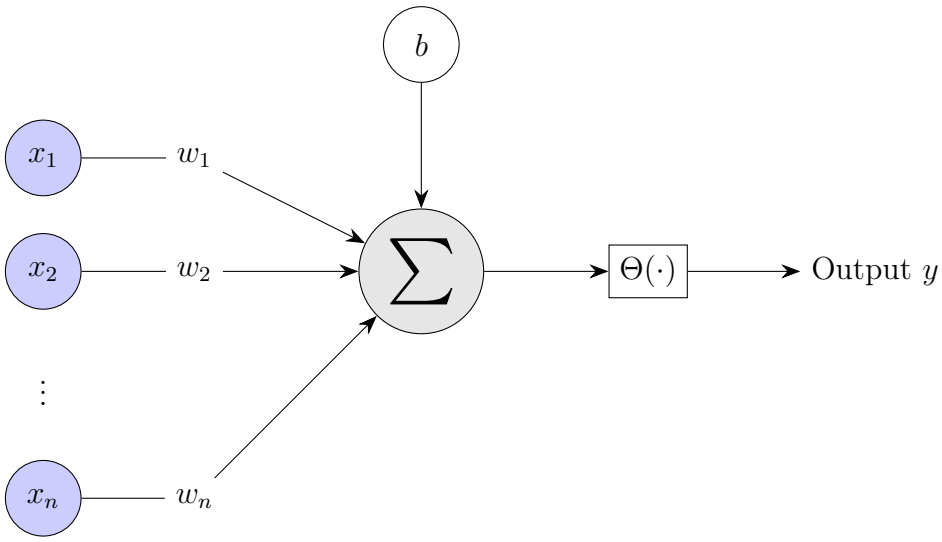


Figure 3: Computational Model of a Neuron

A schematic for the computational model of such a neuron is shown in Figure 3. The input signals x_i are each multiplied with the corresponding weight w_i, the results are added up and a bias value b is added. The result hereof is processed with a so-called activation function $\Theta(\cdot)$.
Therefore

$$y = \Theta\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{1}$$

Often a binary output is desired. In this case, popular choices for $\Theta(\cdot)$ are a step- or sigmoid function. The $sign(x)$ function, along with the two sigmoids $tanh(x)$ and the logistic function $\frac{1}{1+e^{-x}}$ are shown in Figure 4. If a function like $sign(x)$ or $tanh(x)$ is chosen over e.g. the step (Heaviside) function, the binary output becomes $[-1, 1]$ instead of $[0, 1]$.

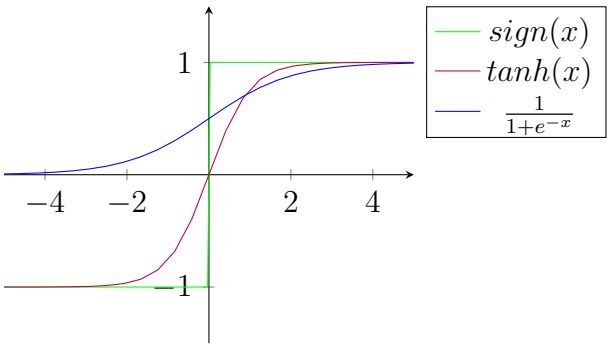


Figure 4: Example of sigmoid functions

The computation of such a binary neuron might be interpreted as the position of a point relative to a hyperplane. This hyperplane is defined by the weights and the bias value; the output tells if the input lies on one side of the hyperplane or the other.

3.3 Multilayer Perceptron

The NN structure used in this thesis is a MLP. In this form of network, the neurons are organized in different layers. The values to be evaluated are presented to the NN via the input layer, the results are represented by the output layer. All layers in between the input- and output layer are called hidden layers. Each neuron is connected to neurons in the next layer, but not to neurons in its own layer. Every connection is represented by a weight value. Figure 5 shows such a NN. Here, the grey circles represent neurons and the arrows represent weights. The input layer does not consist of computational neurons, the squares only represent placeholders for the input values. Each square will receive one specific feature, for example a vehicle's position. The bias values are omitted in the figure. The output neuron tells if a cut-in will happen or not.

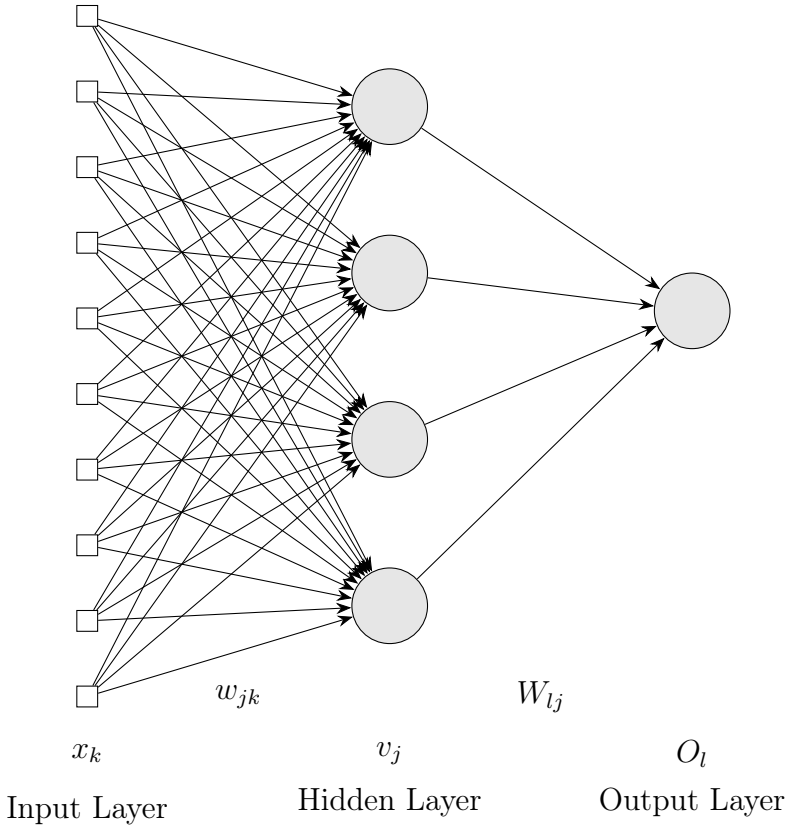


Figure 5: Fully-Connected Multilayer Perceptron

When using Equation 1 in this context, each input pattern $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is processed to the first hidden layer by

$$v_j = \Theta\left(\sum_k w_{jk} x_k + b_j\right) \tag{2}$$

Where v_j are the values, which the neurons in the hidden layer will pass on to the next layer, Θ is the activation function, w_{jk} are the weight values from the kth input neuron to the jth hidden neuron and b_j are the bias values for each neuron in the hidden layer. For the output layer, using Equations 1 and 2 it looks like so:

$$\begin{aligned} O_l &= \Theta\left(\sum_j W_{lj} v_j + B_l\right) \\ &= \Theta\left(\sum_j W_{lj} \Theta\left(\sum_k w_{jk} x_k + b_j\right) + B_l\right) \end{aligned} \tag{3}$$

With W_{lj} the weights between the jth neuron from the previous layer and the lth output neuron and B_l the bias values for each neuron in the output layer.
The second line of Equation 3 is specific for the output of a perceptron with one hidden layer like the one in Figure 5. MLPs might have any number of hidden layers, the computation of each layer is done similarly to the procedure shown in Equation 2 and 3.
When reviewing the interpretation of a single neuron as a hyperplane, every neuron in the hidden layer is such a hyperplane. In a case like in Figure 5 with exactly one hidden layer, the output can be viewed as the position of an input $\vec{x}$ relative to j hyperplanes. Thus, a perceptron with two input neurons, three hidden neurons and one output neuron is capable of classifying a set as sketched in Figure 6.
In the simplified example in Figure 6, the class of black dots relates to a cut-in while the class of white dots does not. The relation to a cut-in can be determined from the features x and y, which are the inputs to the network. The two classes can't be separated with a straight line in the plane, so more than one hidden neuron is needed to separate the classes without faults. The hyperplanes defined by the hidden neurons are drawn as the red lines. As one can see in Figure 6, the class of a sample can be determined by the sample's position relative to those hyperplanes.
The purpose of network training is to find an appropriate set of such hyperplanes, which is equivalent to finding a set of weight values.

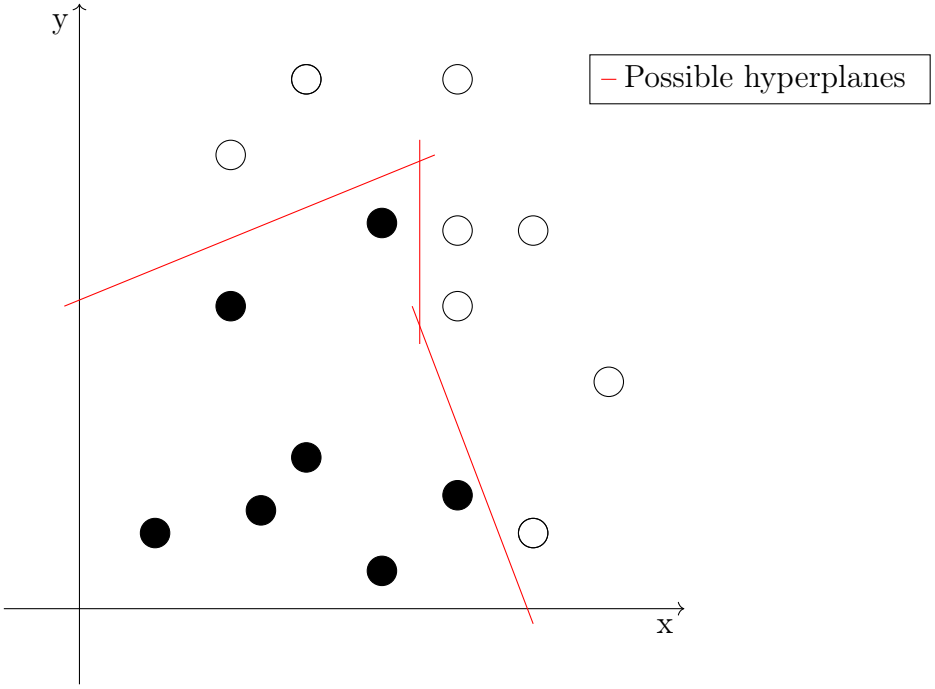


Figure 6: Classification with Hyperplanes

3.4 Backpropagation

To achieve classification of a particular set of input vectors with a perceptron, one needs to adjust the weights and biases, so the corresponding hyperplanes separate the different classes of inputs. Typically this is done by training the NN. The training set consists of N pairs of input pattern $\vec{x}^{(\mu)}$ and desired output $\vec{d}^{(\mu)}$ for the respective pattern where $\mu \in \{1 \dots N\}$ is the index of the pattern. The input patterns in this work consist of selected features from fused sensor data; the desired output shows if the sample is related to a cut-in or not.

The error function of the NN is defined as the sum of squared errors of the output neurons with $O_l^{(\mu)}$ the output generated by the network:

$$E^{(\mu)} = \frac{1}{2} \sum_l \left(d_l^{(\mu)} - O_l^{(\mu)} \right)^2 \tag{4}$$

The average squared error is

$$E_{av} = \frac{1}{N} \sum_{\mu=1}^{N} E^{(\mu)} \tag{5}$$

In order to update weights and biases with a rule like

$$w_{xy} = w_{xy} + \delta w_{xy} \tag{6}$$

where w_{xy} represents any weight or bias, we define δw_{xy} as

$$\delta w_{xy} = -\eta \frac{\partial E}{\partial w_{xy}} \tag{7}$$

with learning rate η and E an error function defined as in Equation 4 (on-line mode) or 5 (batch mode). Equation 7 represents a gradient descent on the error function, thus by applying the updates to the weights, a minimum of this function is being approached. In on-line mode, the weights and biases are updated after each presented training pattern; in batch mode all the training patterns are processed first and the average error is computed before the weight updates are calculated.

These updates are first computed for the output layer, where the input to the error function is directly defined as the difference of the computed output and the desired output. With the partial derivatives of the output layer on hand, the updates are computed for the hidden layer before the output layer. Then these partial derivatives are used for the updates of the weights and biases of the hidden layer beforehand and so forth. When all the update values are computed, they are applied to all the weights and biases.

The fact that the error correction is computed backwards through the network (see Figure 7), gives this method the name *backpropagation*. This method was discovered independently by Williams et al, Parker and LeCun in the mid-1980s [9, 10, 11].

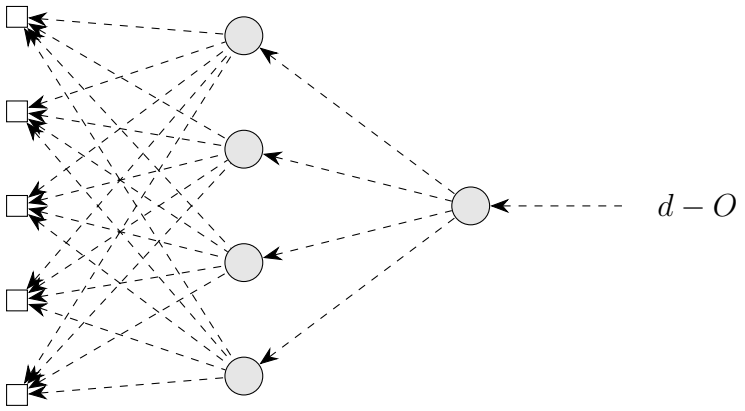


Figure 7: Signal Flow in the Backpropagation Algorithm

3.5 The Problem of Overfitting and Regularization

This section depicts overfitting and an approach to minimise its influence in NN called regularization. Overfitting is observed in the results in Section 6.1. In Section 6.2, the regularization approach presented here is applied.

3.5.1 Under- and Overfitting

As mentioned in Section 3.3, given a binary classification problem, a MLP intends to find a set of separating hyperplanes between the two classes of input provided. Another way of interpreting this property is as a curve-fitting algorithm: The NN tries to find a nonlinear function that separates the two input classes. The free parameters in this curve fitting tasks are the weights of the MLP.
If the number of these free parameters is too small to represent the properties of the nonlinear characteristics of the input data, the curve generated by the network will not fit well. This is called underfitting.
If on the other hand the number of free parameters is much higher than needed to reflect the shape of the boundary between the two classes of the input data, the network might output a curve, that fits the training data well, but fails in generalizing the data presented.
Examples for those cases are shown in Figure 8. Here, the thick dots have been used to generate the curve approximation and the opaque dots represent additional points in the distribution. In the underfitting example on the left, the curvefitting algorithm has few parameters to tweak and can not represent the data well. On the far right, in the overfitting example, the algorithm has many parameters to tweak and manages to fit the training data without errors. But when looking at the opaque dots, it performs worse than the fit shown in the middle panel. This one has a medium number of parameters to adjust, it doesn't manage to fit the training data perfectly. But its overall error on the shown datapoints is smaller than in both other cases.

3.5.2 Regularization

The free parameters for the curve-fitting in NNs are the weights. For MLPs, a high number of free parameters is easily achieved with many hidden units. As shown in Section 3.5.1, the complexity of the curve fitted to the data on hand should not be too high. So the error function for backpropagation can be extended by a complexity penalty term:

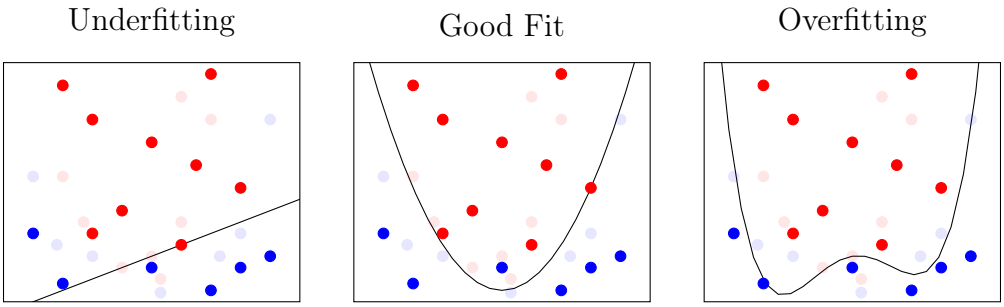


Figure 8: Example of Under- and Overfitting. (Opaque dots represent data not used in training)

$$E = \frac{1}{2}\sum_{l}(d_l - O_l)^2 + \lambda\frac{1}{2}\sum_{w\in\mathbb{W}} w^2 \tag{8}$$

Where the first part is the quadratic error as specified in Equation 4, $\mathbb{W}$ is the set of all weights in the network and λ is the regularization parameter. For $\lambda = 0$ the network output will not differ from the regular backpropagation from Section 3.4. The bigger λ is chosen, the more importance is given to the magnitude of the weights. This magnitude correlates to the complexity of the fitted function, so a low overall magnitude helps to achieve low complexity. If the regularization parameter is chosen too big, the network will not be able to fit the presented data well, and will suffer from underfitting.
With this manipulated error function, using Equation 7:

$$\begin{aligned} E_S &= \frac{1}{2}\sum_{l}(d_l - O_l)^2 \\ E_R &= \lambda\frac{1}{2}\sum_{w\in\mathbb{W}} w^2 \\ E &= E_S + E_R \\ \delta w_{xy} &= \eta\cdot\left(\frac{\partial E_S}{\partial w_{xy}} + \frac{\partial E_R}{\partial w_{xy}}\right) \\ &= \eta\cdot\left(\frac{\partial E_S}{\partial w_{xy}} + \lambda w_{xy}\right) \end{aligned} \tag{9}$$

The addition to the standard backpropagation algorithm consists therefore of scaling each weight with $(1 - \eta\lambda)$ before adding the derivative of the quadratic error function. The regularization term in the error function allows the network to set all parameters it assumes to be unnecessary to zero [7]. If λ is not chosen too big, the parameters essential for the network to operate are still allowed to be significant in size.

Based on the thought, that the regularization eliminates all unneccessary weights, one change to the formula can be made: There are parameters in the network that are important for the classifier, namely the biases. These are unique for every layer, can't add noise (as they are fixed values) and ensure that the hyperplanes constructed by every neuron don't need to go through the origin. Accordingly, the bias parameters are not regularized along with all the other weights.

4 Conceptual Considerations

This section gives an insight in the different approaches taken into regard for this thesis. It wants to clarify on possible options for the network algorithm, the dataset generation and the specification of the desired target, that the algorithm shall produce.

4.1 Definition of the Target for the Classification Task

Maas et al define the target for their classification by taking the samples 6 seconds before the cut-in is performed as positive, the samples 8 to 6 seconds before as negative [12]. In this thesis, a similar approach is considered; here the cut-in event is defined to be the moment the target vehicle crosses the lane marker. A specified number of samples before this event is taken as positive, the rest as negative.
Figure 9 shows an example of the target vector over time along with sketches of the scenario on the road corresponding to different parts of the target vector. The yellow triangle marks the cut-in instance.

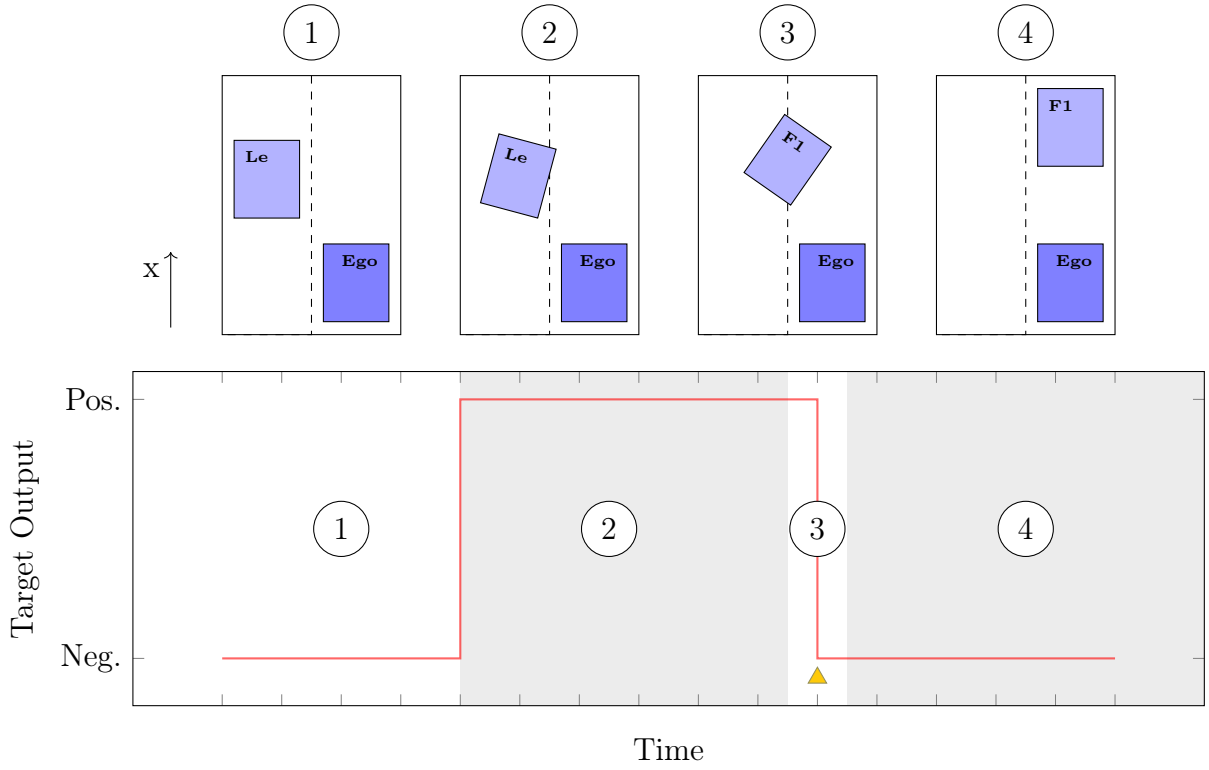


Figure 9: Showcase Target Vector and Sketches of Corresponding Road Scenarios. The Yellow Triangle marks the cut-in event, x marks the driving direction of the vehicles.

Long before a cut-in, the target vehicle and the ego vehicle both drive on their own lanes. In figure 9, the corresponding scenario and Part of the target vector are marked with (1). As this behaviour does not correlate with a cut-in, the corresponding target vector is negative.
During the timespan immediately before the cut-in, the target vehicle moves closer to the ego car's lane. This motion should be detected by the NN. The target vector for this timespan is positive, indicating that a cut-in will happen soon. The respective scenario and target vector section are marked with (2).
The cut-in itself is the moment the target vehicle crosses the lane marker. At this event the target vector goes from positive to negative. The cut-in event is marked with (3) in Figure 9.
When the target vehicle has passed the lane marker, it becomes the leading vehicle to the ego vehicle. This means, starting from the cut-in event, the ego vehicle is driving behind the target vehicle. Such a situation should be classified negative by the NN, thus it gets a negative target value assigned. The respective scenario and target vector part are marked with (4) in Figure 9.
The falling flank to this algorithm is clearly indicated by the cut-in event. The rising flank on the other hand is a flexible bound. There is no clear indication from the scenario on the road, when this bound should be placed.
In this thesis, two approaches to assign the rising flank are taken. One is to assign it when the target vehicle's parameters exceed certain boundaries, e.g. the distance to the lane marker becomes smaller than a threshold. The thresholds are explained in more detail in Section 4.4. Section 6.4 contains test results gained with a target vector, that is constructed following this principle.
The other approach to assign the rising flank is to take a specified number of seconds before the cut-in. This approach allows to determine how soon in advance a NN can predict a cut-in with arbitrary precision. The results in Sections 6.5 and 6.6 are gained with such a target vector.

4.2 Selecting a Network Structure

In this thesis, a MLP has been chosen to accomplish the task of predicting cut-ins. Machine learning is considered to be suited for this task because of its similarity to human learning [13]. Also, other tasks in self-driving cars, as for example car-following models, could be handled using machine learning [2, 1, 3].
Table 1 shows machine learning algorithms taken into account in this thesis.

	SVM	FFNN	FFNN (t.d.)	RNN
low complexity structure	●	●	◑	○
easy to train	●	●	◑	◑
time dependence	○	○	●	●
neural representation	○	●	●	●

● satisfied
◑ partially satisfied
○ not satisfied

Table 1: Properties of different algorithmic concepts

A SVM is a function approximator that uses an input set marked with the classes it should separate. It then computes the minimum of an error function defined on misclassification of this data. This algorithm serves the same purpose with a similar approach than a MLP, but is not a neural network.
A MLP, also called FFNN, uses computational neurons grouped in layers to perform a classification task. It differs from the SVM in the calculations, that are performed to classify the input data and minimize the classification error. Details about its operation can be found in Section 3.3.
In their standard form, both SVM and FFNN don't take time series data into account, they represent classifying algorithms for classes of single, unconnected data points. The time dependent variant of FFNN differs from the standard algorithm by taking a number of previous samples into account as well. If $\vec{\xi}(t)$ is the vector of features at time t, and n previous samples are taken into account, the input to the network at time t looks as follows:

$$\vec{x}(t) = [\vec{\xi}(t), \vec{\xi}(t-1), \vec{\xi}(t-2), \ldots, \vec{\xi}(t-n)] \tag{10}$$

Some training algorithms for a FFNN with such an input take the time dependence into account as well.
A RNN is a neural network containing circles in its structure. As the input data is processed sequentially, these loops act as a memory and allow for the network to make predictions based on the input data and the outputs of the recurrent units. RNN can be equally mighty in describing dynamical systems as state-space equations are [7]. Being a representation of dynamical systems, instabilities might occur, which makes the training of RNN more difficult than the training of FFNN.

Maas et al compare different algorithms to predict cut-ins on simulated data, they achieve the best results for SVM and FFNN [12]. Both this algorithms are standard algorithms in machine learning and have a lower complexity than time-dependent FFNN and RNN. Being standard algorithms, there exist a multitude of extensions for both. The time-dependent FFNN and the RNN are considered an extension or specialized variant of the MLP, so if the task can be solved with a MLP, these algorithms offer room for improvement.
Being standard algorithms, FFNN and SVM are easy to train with their respective standard training methods. Concerning time-dependent FFNN and RNN, the time dependence of the input data needs to be taken into account, e.g. via backpropagation through time. In the case of the RNN, possible instabilities need to be taken into account as well [7].
The time-dependent FFNN and the RNN additionally offer a memory to the network, which allows to take the time-dependence of the data into account. Both Maas et al and Dogan et al find that incorporating the time-dependence is beneficial to the task [12, 13].
All algorithms taken into consideration in Table 1 but the SVM are neural models. So results gained with a MLP, as presented in this work suggest that time dependent FFNN and RNN also offer the possibility to solve the given task. With the SVM no conclusions can be drawn concerning the other three algorithms.
Maas et al states that a MLP can solve this classification task with data out of a simulator [12]. It is yet to examine if this also works with real captured data.

4.3 Selecting the Features to Input to the Neural Network

The Features to give to the NN as inputs have to be chosen prior to training. In this thesis, two approaches to selecting the features have been taken. They differ in the overall number of input values to the neural network. Those alternatives are discussed below.

4.3.1 Large Quantity of Features

The motivation to give a large number of features to a NN is to have the network attach the appropriate importance to each input. In theory, a learning algorithm would do so by giving very small weight values to all those features it considers unnecessary. One downside to this method is the required overall number of weight values and resulting

weight updates in training, which will increase the computation time compared to fewer input features.
The features to input to the dataset were chosen based on the consideration if they might be useful for the task of perceiving and evaluating a cut-in scenario. Table 2 shows the features selected for the dataset. It will be used in the tests in Section 6.1 and Section 6.2. All the vehicles which the ego car's sensors can detect are stored in the object structure. First closest in lane marks the vehicle in front of the ego vehicle, in Figure 1 it is marked 'F1'. Pahhist contains a 'snail trail' of positional data.

Entity	**Features**
Object	Longitudinal and Lateral Position Speed and Acceleration Heading Angle Type of Vehicle, Width, Height Lateral Distance from Middle of Ego Lane
First Closest in Lane	Speed and Acceleration Heading Angle Longitudinal and Lateral Position Type
Ego Vehicle	Longitudinal Speed Longitudinal Acceleration Yaw Rate
Pahhist	Each Entry

Table 2: Contents of the Dataset with many Features

4.3.2 Specific Features

As an alternative to the datasets with many different features, that bases on the philosophy to have the network figure out all the correlations by itself, the features fed to the network can be chosen in advance based on their definitive relation to the cut-in. Such a dataset allows for a deeper understanding of the NNs performance, since the connection between the input data and the target value is known. When picturing the scenario sketched in Figure 1, one factor with a clear correlation is the lateral position of the targeted vehicle. Another correlation can be seen when plotting the heading angle of the target vehicle over the sampling time: before cutting in, the vehicle is headed towards the ego cars lane. While driving on its own lane it's heading straight in the

direction of lane at most instances. As the road may be curved, the bare lateral position and angle have to be interpreted in relation to the road geometry. The vehicle positions and angles on a curved road are shown in Figure 2. To detect a cut-in, it is beneficial to know the target vehicles position and angle with respect to the lane marker of the ego car's lane. So the road geometry is interpreted using the position of the lane maker to the right and left of the ego lane.

Entity	**Features**
Vehicle to the Right	Longitudinal and Lateral Position Heading Angle
Vehicle to the Left	Longitudinal and Lateral Position Heading Angle
Ego Vehicle	Lateral offset to Center of Lane
Lane marker Positions Right and left	All available coefficients

Table 3: Contents of the Dataset with selected Features

Table 3 shows the features chosen in this dataset. It contains positional data and angles for the cars to the right and left and the ego vehicle. Additionally this network takes the lane marker positions. These are stored as coefficients for a polynomial depending on the longitudinal position. All those coefficients are given to the NN.

4.4 Using Thresholds to Ensure a Correlation of the Input Features to the Cut-In

The input features of the vehicle to the right and left from Table 3 are compared to thresholds. Such a comparison shows that these features correlate with a cut-in. It also allows to compare different cut-in scenarios by checking how soon before the event the same variable value is reached.
The threshold for the lateral and longitudinal distance specifies a maximal distance to the lane marker, the target vehicle needs to fall below ahead of a cut-in. For the heading angle, the threshold ensures the target vehicle is heading towards the ego car's lane.
Due to the lateral distance and angle being measured with respect to the ego vehicles driving direction, the measurements on a straight road differ from those on a curved road in an otherwise identical scenario (see Section 2.3). Thus, the road geometry has to be taken into account.

The compliance with all specified thresholds can be evaluated per sample. This method is not able to detect cut-ins in the recorded data with arbitrary precision. If the thresholds are conservative, not all cut-ins are detected by this algorithm. On the other hand, if the thresholds are more generous, they apply in other samples than those related to cut-ins as well.
Accordingly, the thresholds are chosen such that they apply to most cut-ins while being restrictive enough to exclude regular driving.

5 Implementation of the Dataset Generation and the Networks

In this section, a presentation of the realisation of the necessary elements to predict cut-ins with a NN is given.
The training dataset needs to be deliberately composed, because the recorded data contains only a small number of cut-ins compared to other traffic scenarios. In the following, the mechanism how to accomplish a training dataset composition, from which a NN can learn to predict cut-ins, is explained.
Additionally the characteristics of the implemented NN and the procedure to evaluate the performance of the trained networks are presented.

5.1 Composing and Preparing the Training Dataset

For network training, a matrix-like structure is desired. Every column in the dataset represents a feature of the data, every row is a sample at a discrete timestep. So for each training step, one row can be given to the NN.

Feature 1	Feature 2	Feature 3	...	Target Value
$F_1(t_0)$	$F_2(t_0)$	$F_3(t_0)$	...	$T(t_0)$
$F_1(t_1)$	$F_2(t_1)$	$F_3(t_1)$	...	$T(t_1)$
$F_1(t_2)$	$F_2(t_2)$	$F_3(t_2)$	...	$T(t_2)$
...	...	...	...	...
$F_1(t_{n-1})$	$F_2(t_{n-1})$	$F_3(t_{n-1})$	...	$T(t_{n-1})$
$F_1(t_n)$	$F_2(t_n)$	$F_3(t_n)$	...	$T(t_n)$

Table 4: Structure of the Dataset for Training

To generate such a dataset, the features are chosen in advance. Approaches to choosing features are described in Section 4.3.
Cut-in scenarios are located in the logged data and copied to the training dataset. This selection is necessary, because the cut-in scenarios only account for a small fraction of the recorded data. When training on a dataset heavily biased towards one target class, the network will not learn to classify the other class correctly, so the training dataset needs to contain both classes in similar magnitudes.
The structure of a scenario to copy to the training dataset is sketched in Figutre 10. The part to cut out of the dataset evolves around the cut-in event's marker t_i. This marker is

either taken from the recorded data or defined by scanning the logs for instances where a vehicle from an adjacent lane becomes the leading vehicle to ego. The respective algorithms are explained in greater detail in Section 2.1 and Section 2.2.
Four parameters are set to define the structure of the scenario for the training dataset: a, b, x_a and x_b. Table 5 gives an overview of those parameters. b and a are used to define the start s and end e of the excerpt from the recorded data by taking the respective number of samples before and after the cut-in. x_b and x_a define the range around the cut-in marker to assign a positive target value to. Using this assignment, the target value states if there will be a cut-in in the following x_b samples or if there has been one less than x_a samples ago. The four parameters a, b, x_a and x_b are constant for all cut-ins to incorporate into the dataset.

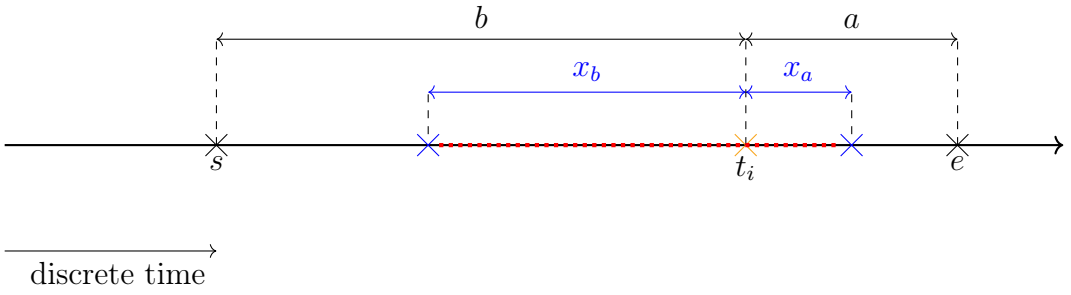


Figure 10: Extracted Part of the dataset for a cut-in event

Parameter	Description	Size
b	the number of samples to go back in time before t_i to reach the start of the scenario	positive
a	the number of samples to go forth in time after t_i to assign the end of the scenario	positive
x_b	the number of samples to go back in time before t_i to mark the start of the positive target value annotation	$0 \leq x_b \leq b$
x_a	the number of samples to go forth in time after t_i to mark the end of the positive target value annotation	$0 \leq x_a \leq a$

Table 5: The Input Parameters to the Dataset Generation Algorithm

In conclusion, a dataset with selected features is built by taking a specified range of samples around the t_i value. Another, smaller range of samples is assigned a positive target value. All the other samples have a negative target value. This algorithm allows to

vary the training datasets in terms of length around the cut in scenario and distribution of the target values within the scenario.
If two cut-in instances, t_{i1} and t_{i2} have less than $b+a$ samples between them, the ranges to cut from the original dataset overlap. Furthermore, if they have fewer than b samples between them, the second excerpt from the log will always contain both cut-in markers. This case is pictured in Figure 11. This situation is handled such that the overlapping sets of samples are merged so the part to cut from the record starts with s_1 and ends with e_2. Using this measure avoids to add some samples to the training dataset twice. In the bigger ecxerpt containing both t_{i1} and t_{i2}, the samples around both cut-in instances have to be marked according to the parameters x_b and x_a. These annotation ranges are displayed with a red, dotted line in Figure 11.

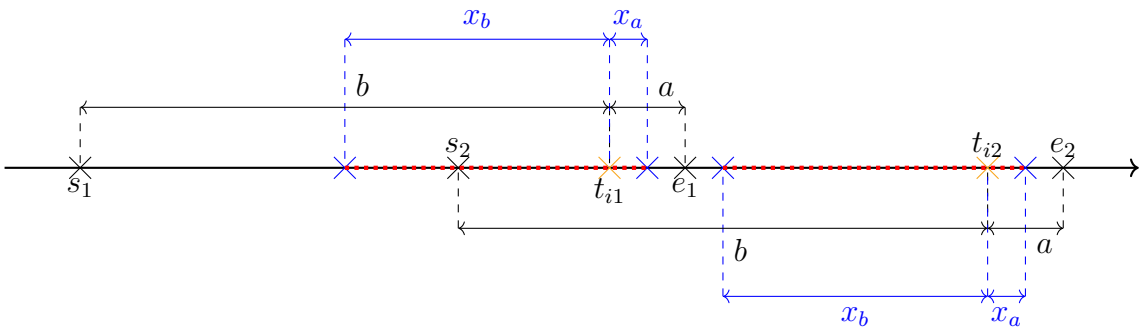


Figure 11: Doubles in the Dataset

In training and validation, all excerpts from the recorded data are connected together to one continuous stream of data.

5.2 Chosen Network Characteristics

This section specifies the implementational details of the FFNN used in this thesis.
The NN can be initialized with 1, 2 or 3 layers and a variable number of neurons per layer. The only exception is the output layer, which always contains only one neuron in order to perform the binary classification task.
Each neuron uses $tanh(x)$ as its activation function. The network training is performed batch-wise, so all the training set is processed by the network first before calculating the average error and then updating the weights. The error function used is the average squared error as defined in equation 5.
The weight updates are calculated with the standard backpropagation algorithm as described in Section 3.4.

Available hyperparameters of the network are the learning rate η and the steepness of the activation function by introducing a parameter β and using $tanh(\beta \cdot x)$.
The initial weights are generated using a uniform distribution with a mean of zero and a variance reciprocal to the number of connections of each neuron [7].

5.3 Procedure to Evaluate Network Performance

In Section 6, NN are trained to achieve different classification goals using different training dataset and parameters. The performance of the trained networks is evaluated and analysed.
The standard process in testing is visualized in Figure 12. The tests are carried out with randomly generated weights. Those weights are saved to provide reproducibility. Using these initial weights and the training dataset, the network is trained to learn the given task from the dataset. In the training process, the weights are continuously adapted to reduce the network's error; the mathematical procedure can be found in Section 3.4. After training, the weights are saved as well. As shown in Figure 12, the weights are no longer changed when processing validation data in the network. So, with the trained weights on hand, the network's performance can be evaluated on different sets of validation data.
The errors on the training set and on each validation set are calculated and saved. The percentage of errors to dataset length serves as measure of performance.

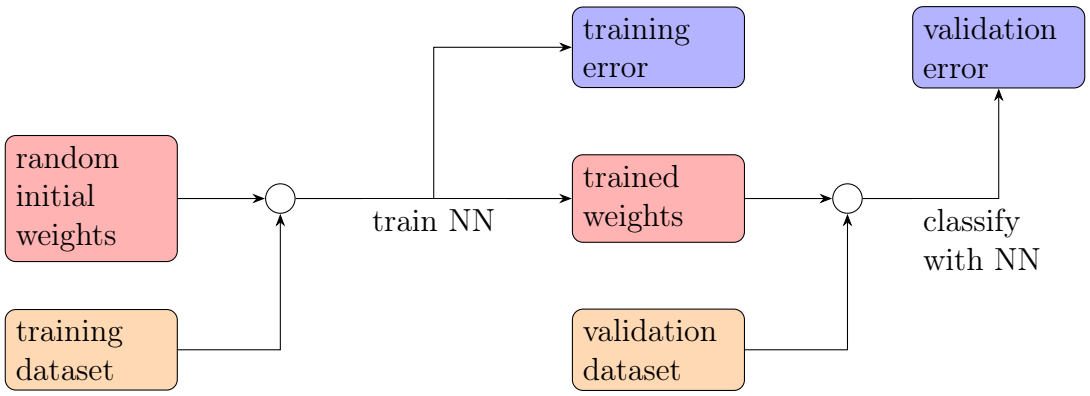


Figure 12: Visualization of the Standard Process for Network Training and Evaluation

As the trained weights are saved, the network can also be trained further without retraining from the initial weights.

6 Training Networks and Analysing their Performance

This section aims to both present and interpret the conditions and findings of the tests conducted during this thesis. Different approaches to selecting the features, generating the ground truth and assigning the classes the network should distinguish between have been used. The specific methods chosen are outlined for each test.

6.1 Analysing the Performance of Networks Trained to Predict Cut-Ins with a Dataset based on Flags from the Log and Many Features

This test series aims to detect cut-ins without predicting them. A possible use-case for a detection-only algorithm is the postprocessing of recorded data, as mentioned in Section 1.1.
The tests are carried out with the input features specified in Table 2, using the approach to take many features as inputs to the NN. The NN are expected to find the relations of the features in the dataset by themselves and adjust the weight values according to the importance of the respective features.
The cut-in events are specified using the marker that can be found in the logged data. This marker gets assigned during driving based on the sensor inputs at the respective timestep. The algorithm is explained further in Section 2.1. If the marker is set in consecutive timesteps, only the first one is used to indicate the cut-in event.
In training, the networks are presented only cut-in scenarios taken individually from the logs and combined to generate one big training dataset. These cut-in scenarios contain some negative samples before and after the event in addition to the positive samples directly around the cut-in event. The numbers of positive and negative samples in the dataset are equal in size.
In multiple tests with different assignment methods and varying hyperparameters (learning rate, number of hidden neurons etc.), the output consistently shows a high number of false positives and high validation errors. Two main reasons are expected to cause the high number of false positives. The first one is, that limiting the training dataset to exclusively cut-in scenarios excludes the network from learning any other situation on the road. The other reason why the network does not perform well is the choice of ground truth in the training dataset. When using the marker from the logs, the target

vector already contains false positives itself. In further examination, many of these false positive scenarios turned out to be lane change manoeuvres of the ego car.
A high number of validation errors along with a poor ability to generalize to unknown data are characteristic indicators for overfitting. This seems likely considering a high number of free parameters is available to the network, which arises due to the high number of input values and a number of hidden units in the same magnitude.

6.2 Evaluating the Effect of Regularization on Training with a Dataset Based on Flags form the Log

The networks trained in Section 6.1 were not able to classify data they had not seen in training. This behaviour might result from overfitting, adapting a complex function to the training data so precisely, that it performs worse on the distribution of all possible input data.
In order to check if overfitting is the problem on hand, in this test the network is forced to use a simpler function on the data. A regularization parameter is introduced, that scales the impact of a penalty on large weight values to the network. The higher this parameter is, the higher the penalty, thus the network is likely to keep the weights smaller in absolute value. Smaller weight values directly contribute to a less complex function approximation.
The dataset for this test is the same as for the previous test from Section 6.1: It contains only cut-in scenarios taken from the logs and stacked together. It takes the big set of input features, listed in Table 2. The number of positive and negative marked samples are of similar size. The target value, if a sample is to be classified true or false, is assigned corresponding to the cut-in marker in the logs described in Section 2.1.
In testing, the network structure and all hyperparameters but the regularization parameter are kept fixed. The regularization parameter is varied in a range from 0 to 4000. For each value of regularization parameter, multiple runs with different randomized initial weights are conducted, to ensure the end result does not depend on the initial conditions. The errors of the networks after being trained for the same number of iterations are saved. Figure 13 shows the training- and validation errors plotted over the value of the regularization parameter.
A rising training error for a bigger regularization parameter is an expected behaviour, as the goal of the regularization is to approximate the training set with a less precise function. When using this technique, the regularization parameter has to be chosen in

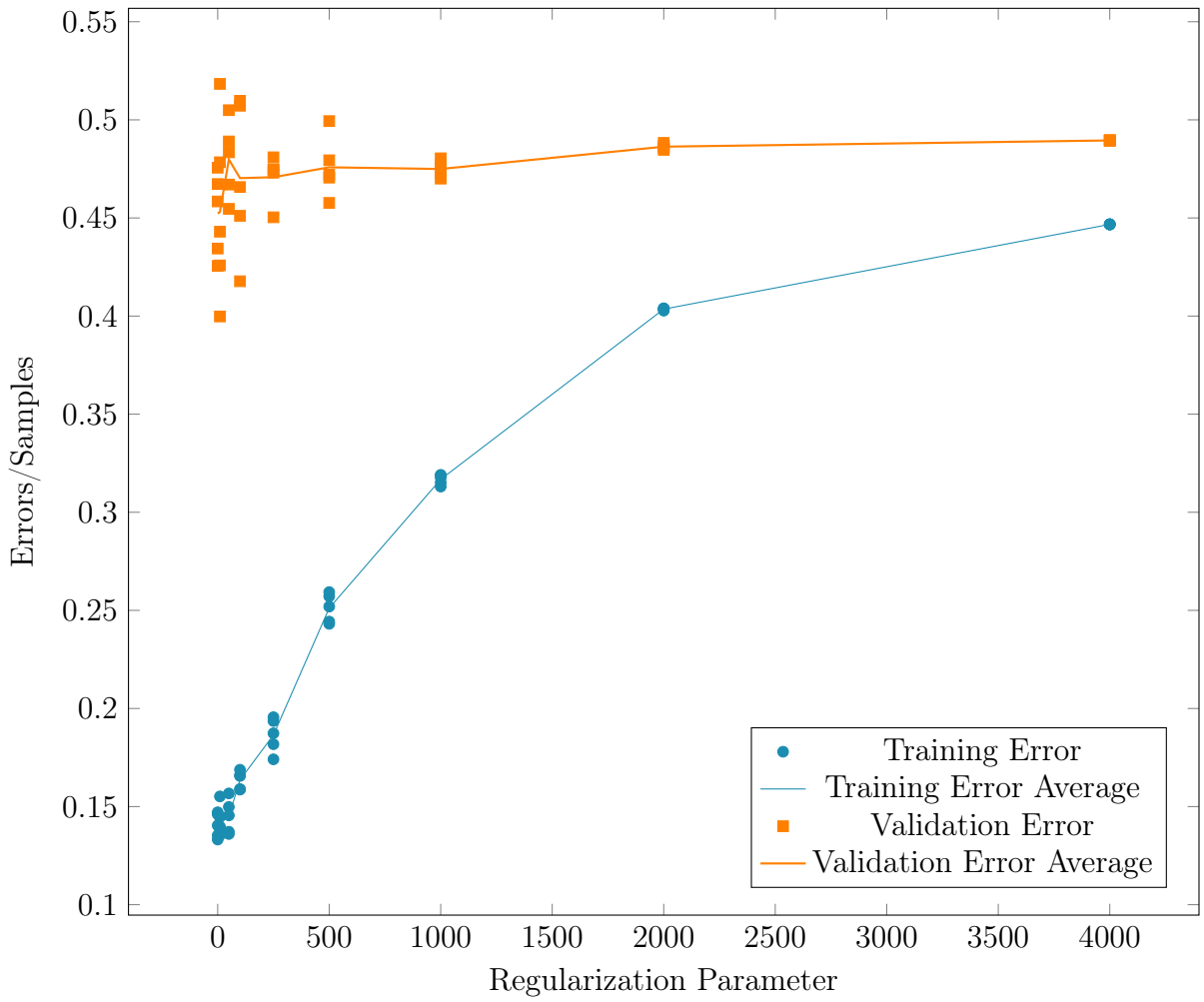


Figure 13: The training and validation error subject to the regularization parameter. The training error rises for larger values of the regularization parameter, the validation error does not significantly change.

such a way, that the effect of training is still visible on the training set. If chosen too big, the network only focuses on keeping the weights small, but no longer on fitting the function to the given data. This behaviour is shown by the network for the larger values of the regularization parameter in the plot.
When observing the orange curve, the validation errors, in Figure 13, two things can be seen: The errors are rather high, just shortly below 50%, and they do not change significantly over different values of the regularization parameter. For the regularization to show the desired effect, the validation error would need to drop for some value of the parameter. The value for the regularization parameter would be expected to be between 0, where the regularization has no effect and a value, that still guarantees useful training. As such a drop in the validation error cannot be observed, this method did not help to solve the problem of low performance experienced in the test from Section 6.1.

6.3 Performance of a Network when Trained to Detect Cut-Ins with Selected Features

This test aims to state if a neural network is able to distinguish the samples before a cut-in has happened from those after. The cut-in is defined to be the instance in which the target car crosses the lane marker to the ego lane.
The input parameters to the network consist most importantly of the positions and angles of the vehicles to the left and right of the ego vehicle. The position of the ego vehicle is input as well. This allows the network to detect lane changes performed by the ego vehicle. A full list of input features can be found in Table 3.
The ground truth data for the target value is found by scanning the logs for cut-ins. A cut-in is found in the recorded data, if a vehicle on an adjacent lane becomes the leading vehicle to ego. A description of the respective algorithm can be found in Section 2.2.
The training set consists of samples around the cut-in instances cut from the logs and stacked together. To detect a cut-in by separating the samples before the event from those after, all samples before the cut-in event are marked to be negative, the samples

after the event are positive. The respective values for a, b, x_a and x_b as parameters to the algorithm from Section 5.1 are chosen as follows:

$$b = 3\ sec$$
$$a = 2\ sec$$
$$x_b = 0\ sec$$
$$x_a = 2\ sec$$

With this data on hand, a network has been trained. Its output on parts of the validation data is shown in Figure 14. The validation data in this test was similar to the training data in its structure. Contrary to the training data, the network has not seen the contents of the validation set in training.

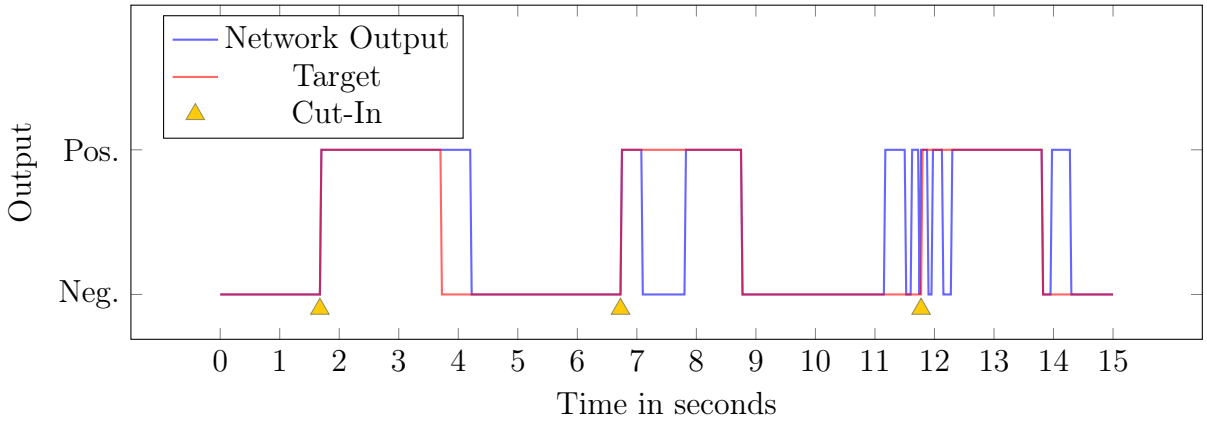


Figure 14: The network is able to distinguish between the samples up to 3 sec before the cut-in and the samples 2 sec after the cut-in. Altough, some samples are classified wrongly.

Figure 14 shows the network's capability to detect the marked cut-ins. The network's output is plotted in blue, its target is shown in red. Cut-in events are marked with yellow triangles. The network's output approximates the target curve visibly. Some samples are classified wrongly, this is expected to result from insufficient training. When trained longer, with optimized hyperparameters and possibly a more extensive training dataset, the precision of the network is assumed to increase.
The chosen set of features seems to allow for a detection of cut-ins, as the network was able to determine the samples before a cut-in from those after.

6.4 Performance of a Network when trained to Predict Cut-Ins with Selected Features

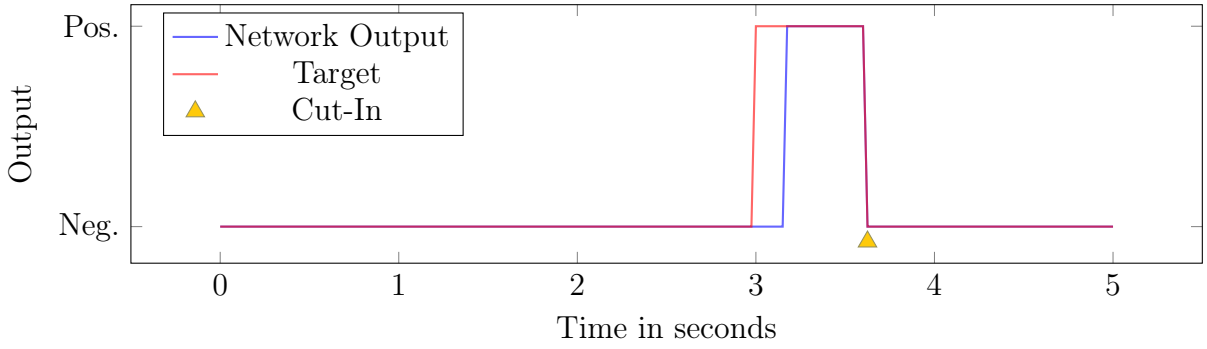


Figure 15: The network predicts the cut-in, but it does so later than desired.

The previous test, outlined in Section 6.3, showed the network's ability to differentiate between the samples before a cut-in event and those after. When aiming to predict cut-ins, the network should be able to differentiate the samples in a period shortly before the cut-in event from both those before this period and after the event.
Accordingly, the time shortly before the cut-in needs to be different from the time long before the cut-in when comparing the input features to the NN. This correlation of the input features to the cut-in event can be shown by comparing the input features' values with thresholds, as introduced in Section 4.4.
The cut-in events for this test are found by scanning the logs for instances, where vehicles on adjacent lanes become the leading vehicle to the ego vehicle. A description of the algorithm used for this task can be found in section 2.2.
The training dataset is composed from all the samples within 5 seconds around these cut-in events. All those samples, that comply with thresholds on the lateral distance and the angle of the target vehicle, are marked positive. All other samples are marked negative. So the network is expected to find the same correlation as the thresholds on the input features do. This also means that there is no unified time interval in advance of the event in the target value.
The features include values of the ego car, the positions of the vehicles to the right and left and the road geometry. A full list can be found in Table 3.
An excerpt of the output of a network trained with this dataset is shown in Figure 15 as the blue line. The red line represents the target value for the network, the yellow triangles mark the cut-in events. The network finds the cut-in, even though a little later than the truth value is assigned.

This result suggests that it is possible to detect the cut-ins in advance based on the data given to the network. The network is able to differentiate between the samples that suffice the thresholds chosen and the samples that don't.
Usage of the trained network on whole logs showed several false positives though. When reviewing the composition of the training dataset, it becomes apparent that the network never learned what other scenarios, apart from cut-ins, look like. This is because the training dataset consists of cut-in scenarios only. All negative samples known to the network lie either directly before or after a cut-in. Thus, the misclassification is likely to be reduced when introducing more negative examples to the training dataset.

6.5 Varying the Time in Advance for the Prediction and Analyzing its Effects on Accuracy

It is possible to predict cut-ins in advance, when assigning the classes to differentiate based on an accordance with thresholds on the features. This is the result of the test in section 6.4. In this test series, it is evaluated how soon the cut-ins can be detected in advance and how the accuracy of the network output is affected by a sooner prediction. As in the previous test, the input features consist of data from the ego car, the positions and angles of the cars to the right and left and the road geometry. A full list can be found in Table 3.
The cut-ins are found in the logs by scanning them for corresponding scenarios. A cut-in is found, if a vehicle on an adjacent lane changes to be the vehicle in front of the ego car. The respective algorithm is described in Section 2.2.
The positive target value is assigned to samples before the cut-in event. In doing so, the timespan before the event is of variable length throughout the test series. This timespan is further called t. The overall training dataset length is three times t. This comes from taking $3 \cdot t$ at each cut-in: t negative samples, t positive samples right before the cut-in and t negative samples directly after the cut-in. Figure 16 shows the proportions of the data taken for the training dataset at a cut-in in relation to t.
To generate such a dataset with the algorithm described in Section 5.1, the parameters are chosen as follows:

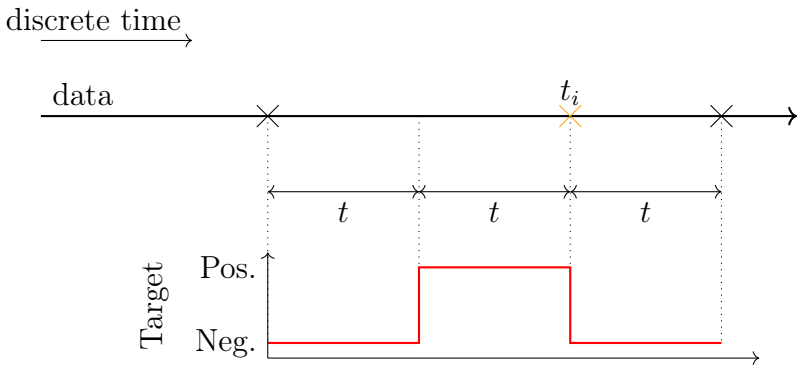


Figure 16: Extract from the record to add to the training dataset with respect to t. The thick black arrow symbolizes the logged data. t_i marks the cut-in instance. The black crosses mark the beginning and the end of the excerpt. On the bottom, the target vector is shown as the red curve.

$$b = 2 \cdot t$$
$$a = t$$
$$x_b = t$$
$$x_a = 0$$

Multiple datasets with varying t-parameter are constructed and a NN is trained for each one. The hyperparameters and initial weights are the same for all of these networks.
Each network is trained for 300 training iterations. After that, the output of the network on a 'checking dataset' is evaluated. This checking dataset does not change between input datasets. It consists of multiple whole log files, which have previously been taken to extract scenarios to the training dataset from. This means, some samples of this dataset have been shown to the network in training, some others haven't.
For each t-Parameter, the training is then iterated in the following way: Out of the checking dataset, the four most remarkable false positives are added to the training dataset. The network is trained again with the extended training dataset. For this new training, the saved initial weights are used, so it is trained 'from zero'. This training is continued for 300 training iterations as well.
Table 6 shows accuracies reached by this approach for different values of t. The accuracy is defined as the percentage of correctly classified samples out of all samples in the dataset. It also shows the number of iterations in adding false negatives needed to reach that accuracy. The values after dashes for the false negatives, false positives and true

positives show the respective numbers when the network has been trained with 1000 training iterations.

t	Accuracy	TP	FN	FP	Iterations needed
1	98%	5/8	3/0	5/3	2
1.5	97.03%	6/8	2/0	8/7	3
2	96.54%	7/8	1/0	18	3
2.5	95.99%	8	0	21	4

Table 6: Accuracies Reached for Different Times in Advance of the Cut-In

A first observation from the results presented in the table is the decreasing accuracy when using a larger t. The accuracy declines even though more iterations in adding false positives are conducted. Although it's not reflected in the table, the accuracy rises in each such iteration. These accuracies are calculated on the checking dataset. So reaching an arbitrary accuracy seems to be harder, the bigger t gets.
This decreasing accuracy is also mirrored in a higher number of false positives for a bigger value of t. Again not captured by the table, the number of misclassified samples drops when adding more former false positives to the training dataset. When continuing training, the number of misclassifications declines as well. It has to be noted that even after 1000 training iterations, the decline in training error does saturate.
Figure 17 shows the network output on the same dataset for two different networks. The network in the top panel has been trained to detect a cut-in 1 sec before it happens, the one in the bottom panel has been trained to detect it 2.5 sec in advance. The red line shows the respective target value, the blue line represents the networks output. The cut-in events are marked with yellow triangles.
Both networks succeed in detecting both cut-ins shown in the time interval approximately in the requested time. False positives are apparent in both panels; the network trained on 2.5 sec t shows more false positives than the other one. All shown false positives are considerably more narrow than the true positives. The false positives consist of only one or very few samples at a time whereas the true positives consist of a clearly larger number of samples at a time. This behaviour concerning false positives can be observed throughout the test after the networks have been trained.
All in all, the smaller t is chosen, the more precise does the network output get with the same effort in training. Every network taken into consideration throughout this test is able to detect cut-ins. The larger t, the more false positives are found by the

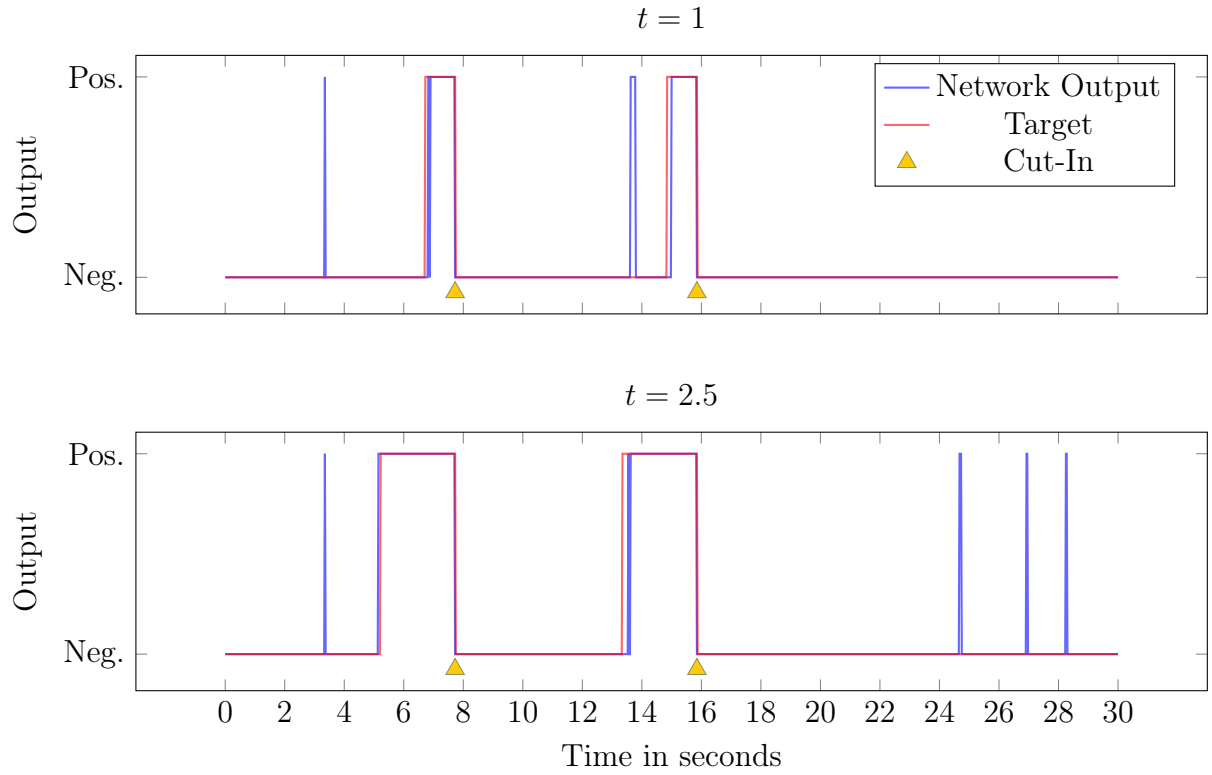


Figure 17: Two different networks predicting the cut-ins at different times in advance. The network in the top panel is designed to detect the cut-in 1 second in advance, the one in the bottom panel is designed to detect it 2.5 sec in advance. The network in the bottom panel shows more false positives than the one in the top panel.

network and more negative examples are needed in training to reach comparable precision. Throughout the tests, the false positives that stay even after long training periods are distinguishable from the true positives by the amount of consecutive samples they contain.

6.6 Evaluating the Performance when the Dataset for the Network contains Random Scenarios in Addition to Positives

In the previous section, the precision of the cut-in detection could be improved by adding the false positives to the training dataset. This section evaluates, if it is possible to improve the performance with the less laborious approach of adding random scenarios to the training dataset in order to incorporate negative scenarios. These randomly selected parts are likely to contain negative samples, as the ratio of non-cut-ins to cut-ins in the logs is very high.

The basis training dataset, containing the cut-in scenarios is designed by cutting excerpts of 7.5 seconds length from the logs. The cut-in events are found by checking for adjacent vehicles that change to become the leading vehicle in the recorded data. This method is explained in depth in Section 2.2. In these excerpts, the first and last third or samples is marked negative and the middle third is marked positive. As positive samples are marked 2.5 sec before the cut-in, the parameters to the dataset generation algorithm from Section 5.1 are:

$$b = 5 \ sec$$
$$a = 2.5 \ sec$$
$$x_b = 2.5 \ sec$$
$$x_a = 0$$

The basic training dataset corresponds to the dataset marked 2.5 seconds in advance from the previous section.

Additionally, randomly taken parts of the same length, 7.5 seconds, are added. They are marked appropriately, so if they contain a cut-in, the 2.5 seconds in advance are marked positive.

Two training datasets are generated, one with an approximately equal amount of cut-in scenarios and random scenarios, the other with a ratio of 1:2 of the cut-ins to the random excerpts. They will be referred to as the smaller and bigger dataset based on the total number of samples. Two NN are trained for 300 training iterations with the

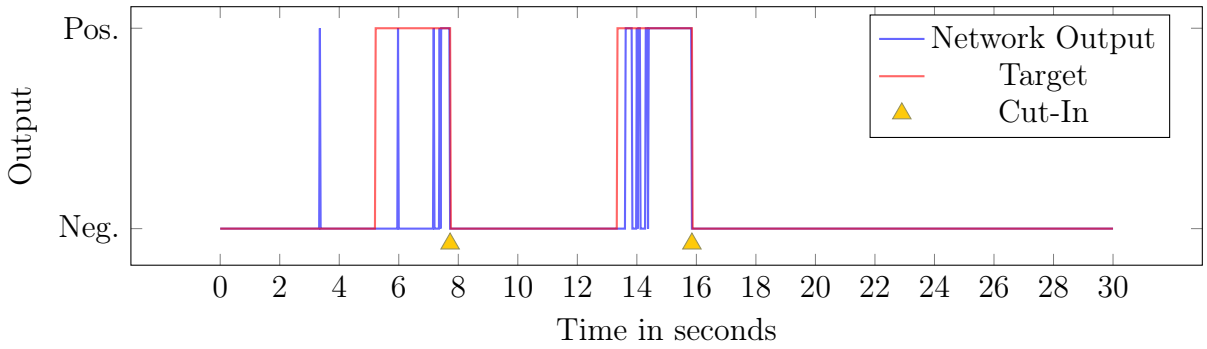


Figure 18: The network is supposed to predict the cut-ins 2.5 sec in advance. There are nearly no false positives, but the detection of the first cut-in only happens ca. 0.5 sec in advance.

same initial weights and the different training datasets. Additionally both are trained for 1000 training iterations.

Figure 18 shows the output of a network, that has been trained on a dataset generated as mentioned above as the blue curve. The target value for the network is plotted in red, the cut-in events are marked with yellow triangles.

The network detects the first cut-in barely, the second one considerably later than desired. When continuing training, the precision of the prediction improved though. For the bigger training dataset, the number of false positives was smaller, but the time to learn to classify the true scenarios was greater.

So it is possible to train the network to predict cut-ins and avoid false positives when adding random false training scenarios. In comparison to adding the false negatives, as practised in Section 6.5, the network takes longer to train under the same conditions to reach the same performance. This is likely to result, because when adding the false negatives, the training set contains precisely those scenarios that are hard to classify for the network.

When adding random samples, the relative magnitude of these corner cases is only reached, when a reasonably big overall number of negative samples is added. In this case though, as seen in the comparison between the two datasets in this test, the network has a hard time to learn the positive scenarios. The more samples of one class are included in the training dataset, the more likely the network output is heavily biased towards this class.

7 Discussion

In Section 6.2 a less complex function did not improve the networks' performance. The reason for the poor performance of the networks in the tests from Sections 6.1 and 6.2 is assumed to be something else than overfitting. A very likely cause is the algorithm used to find the positive examples in the logs. In the tests with high validation errors, the ground truth for the training set has been generated by the algorithm that detects cut-ins using only samples up to the cut-in. Thus, it contains false positives. The algorithm is described in detail in Section 2.1. On the other hand, the tests using the algorithm that checks for cars on adjacent lanes changing to be the vehicle in front of the ego vehicle showed significantly better performance. More details about this algorithm can be found in Section 2.2. This comparison supports the claim, that the first algorithm is not suitable to generate the ground truth for the dataset generation.
In Section 6.3, the NN was able to detect the cut-ins in the recorded data. However, one property of the generated training dataset is worth considering: the absence of the target vehicle after a cut-in event. As the vehicle to the left and right are included in the training dataset, the target vehicle is present up to the moment of the cut-in. In that moment it becomes the leading vehicle, which is not part of the input features. For this reason the task given to the NN is considered rather easy.
The results from Section 6.3 have not been tested on whole logs, so there is no knowledge of false positives the network would find. On the other hand, in order to train a neural network to recognize cut-ins and minimize the number of false positives, the training dataset could be extended by negative scenarios. This method is used in Section 6.5.
The tests that succeeded in predicting cut-ins presented in Sections 6.5 and 6.6 have not been tested on a validation set. The dataset used to evaluate their performance contained samples used in training. For the interpretation of the results, this has to be taken into consideration. The network's accuracy on an unknown dataset is likely to be lower than on the dataset used.
In all tests, the MLP's output has been used as a binary indicator if a cut-in is upcoming or not. Alternatively, the output of the continuous activation function can be interpreted as the network's confidence in the generated result. For the $tanh(x)$ function used in this thesis, a value close to 1 or -1 would imply, the network is very sure about its classification. A value near 0 would imply, the network is rather undecided.

8 Conclusions

This section gathers the findings of this thesis work. The results and methods used are viewed and interpreted in a broader context. Possible directions how to proceed in the matter presented in this thesis are given.

8.1 Summary of the Findings in this Thesis

In this thesis, a MLP trained with backpropagation has been used to predict cut-ins. The validity of this approach could be verified for predictions up to 2.5 seconds in advance of the event. The sooner the prediction should be made, the smaller the precision of the MLP gets. So the earliest possible prediction depends on the performance requirements for the algorithm.
The results of the experiments showed false positives in the prediction task. However, the number of the false positives declined during longer training. They were also much shorter than the true positives, which implies they could be filtered out.
An alternative use-case for the NN presented in this thesis is to detect cut-in scenarios, without announcing them in advance. This capability might be used to postprocess recorded data at VCC.
The tests in this thesis showed the need for good ground truth in the training data for the NN. When the real cut-ins are assigned in the training data with a greater precision, the training performance improves.
The results also suggest, that the training can be improved, when the training data contains non-cut-in scenarios as well. Including the scenarios that are hard to classify for the network yielded the better result in this thesis. On the other hand, including random negative scenarios needs a smaller amount of work.

8.2 Concepts for Further Research on and Improvement of a Neural-Network based Cut-In Prediction

In this thesis, training efficiency has not been taken into account. The training durations and results are likely to improve when optimizing e.g. the hyperparameters on hand or the network structure regarding the number of neurons and/or layers.
In the future, the test series can be extended to show at what point the input values before a cut-in become indistinguishable from all the other non-cut-in data.

An in-depth analysis of possible input features can be carried out to find out what parameters correlate in the most significant way with the cut-in. The results of this analysis can be used to minimize the feature-space of the NN in order to optimize training and output accuracy.
The cut-in related behaviour of different vehicle classes is likely to differ. While cars need a comparatively short longitudinal distance to the ego car in order to perform a cut-in, a truck likely needs a longer distance. Motorcycles are known to sometimes drive on a lane marker to move faster in a traffic jam. The algorithm presented in this thesis does not take different behaviours of different vehicle classes into account. This might be a useful addition to the algorithm.
As mentioned in previous chapters, the algorithm used in this thesis work is not time dependent. However, the input data to the network is a time series. Even tough the prediction could be accomplished with a FFNN, the classifying algorithm is likely to benefit from an internal state or a variant of memory. To achieve this, time-dependent Variants of FFNN or RNN might be used.

References

[1] A. Khodayari, A. Ghaffari, R. Kazemi, and R. Braunstingl, "A modified car-following model based on a neural network model of the human driver effects," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 42, no. 6, pp. 1440–1449, 2012.

[2] S. Panwai and H. Dia, "A reactive agent-based neural network car following model," in *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pp. 375–380, IEEE, 2005.

[3] M. Tanaka, "Development of various artificial neural network car-following models with converted data sets by a self-organization neural network," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 10, pp. 1614–1630, 2013.

[4] "MIT moral machine," January 2017. Date of page view.

[5] T. I. G. I. for Ethical Considerations in Artificial Intelligence and A. Systems, "Ethically aligned design - version 1," January 2017. Date of page view.

[6] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451–462, 2000.

[7] S. Haykin, *Neural Networks: a comprehensive foundation.* Prentice Hall, 2^{nd} ed., 1999.

[8] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[9] R. Williams, D. Rumelhart, and G. Hinton, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–538, 1986.

[10] D. B. Parker, "Learning-logic: Casting the cortex of the human brain in silicon," 1985.

[11] Y. LeCun, "Une procédure d'apprentissage pour réseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks)," in *Proceedings of Cognitiva 85, Paris, France*, pp. 599–604, 1985.

[12] N. Maas, I. D. Schramm, L. Louis, and T. Rehder, "Empirische evaluation von prädiktionsmethoden am beispiel der vorhersage eines spurwechsels aufgrund der verkehrssituation," in *Entscheidungen beim Übergang in die Elektromobilität*, pp. 195–209, Springer, 2015.

[13] Ü. Dogan, J. Edelbrunner, and I. Iossifidis, "Autonomous driving: A comparison of machine learning techniques by means of the prediction of lane change behavior," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pp. 1837–1843, IEEE, 2011.

[14] D. F. Specht, "Probabilistic neural networks," *Neural networks*, vol. 3, no. 1, pp. 109–118, 1990.

[15] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, vol. 37, no. 3, pp. 328–339, 1989.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[17] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," *Advances in neural information processing systems*, pp. 943–949, 1998.

[18] M. Dougherty, "A review of neural networks applied to transport," *Transportation Research Part C: Emerging Technologies*, vol. 3, no. 4, pp. 247–260, 1995.

[19] Y.-C. Chiou, "An artificial neural network-based expert system for the appraisal of two-car crash accidents," *Accident Analysis & Prevention*, vol. 38, no. 4, pp. 777–785, 2006.

[20] B. Morris, A. Doshi, and M. Trivedi, "Lane change intent prediction for driver assistance: On-road design and evaluation," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 895–901, IEEE, 2011.

[21] C. Wöhler and J. K. Anlauf, "Real-time object recognition on image sequences with the adaptable time delay neural network algorithm—applications for autonomous vehicles," *Image and Vision Computing*, vol. 19, no. 9, pp. 593–618, 2001.

[22] M. R. Meireles, P. E. Almeida, and M. G. Simões, "A comprehensive review for industrial applicability of artificial neural networks," *IEEE transactions on industrial electronics*, vol. 50, no. 3, pp. 585–601, 2003.

[23] G. D. Magoulas and M. N. Vrahatis, "Adaptive algorithms for neural network supervised learning: a deterministic optimization approach," *International Journal of Bifurcation and Chaos*, vol. 16, no. 07, pp. 1929–1950, 2006.

[24] J. Hunt and G. Lyons, "Modelling dual carriageway lane changing using neural networks," *Transportation Research Part C: Emerging Technologies*, vol. 2, no. 4, pp. 231–245, 1994.

[25] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the theory of neural computation*, vol. 1. Addison-Wesley, Redwood City, CA, 1991.