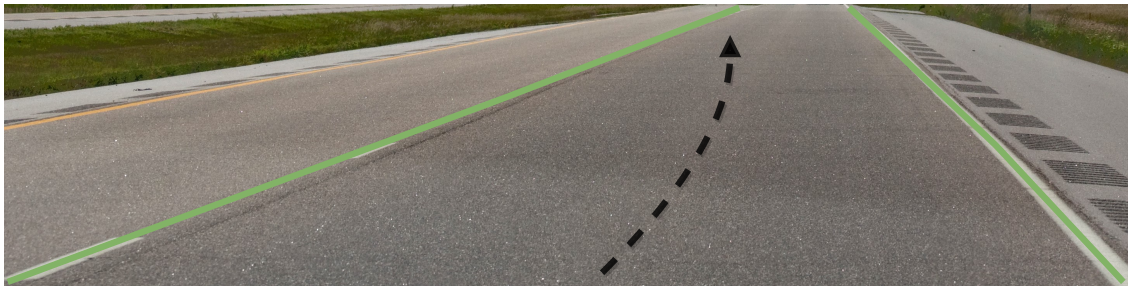


Unintended Lane Departure Prediction using Neural Networks

Master's thesis in System, Control and Mechatronic / Complex Adaptive Systems

Rasmus Jonsson, Anton Kollmats

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis EX051/2018

Unintended Lane Departure Prediction using Neural Networks

Rasmus Jonsson
Anton Kollmats

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems- and Control
Chalmers University of Technology
Gothenburg, Sweden 2018

Unintended Lane Departure Prediction using Neural Networks
Rasmus Jonsson
Anton Kollmats

Supervisor: John Dahl, Chalmers & Zenuity
Supervisor: Gabriel Rodrigues de Campos, Zenuity
Examiner: Jonas Fredriksson, Electrical Engineering, Chalmers
Master's Thesis EX051/2018
Department of Electrical Engineering
Division of Systems- and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: To be determined

Unintended Lane Departure Prediction using Neural Networks
Rasmus Jonsson
Anton Kollmats
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Even though great improvements have been made in terms of traffic safety in the last decades, many accidents still occur on a daily basis. In the United States alone, for instance, roughly 30,000 fatal accidents occur every year. As highlighted in different reports, fatal accidents are often due to human error. Inattention, disregard of traffic rules or poor situational awareness are common causes. Unintentional lane departures, for instance, can easily lead to collisions with oncoming vehicles.

In this thesis, we have focused on the problem of how to accurately predict unintentional lane departures.We have used a Neural Network approach, based on an extensive study of network architectures, in order to capture unintentional lane departure behaviors. The included architectures were: Multilayer Perceptron (MLP), Long Short-Term Memory network (LSTM), Temporal Convolutional Network (TCN) and Multi-Channel Deep Convolutional Neural Network (MC-DCNN). To develop a threat assessment system for lane keeping assistance, we leveraged over 4,000 hours of real vehicle data collected by professional drivers to train the networks.

By building upon previous results derived from simulated data, this thesis is an additional step towards data driven lane keeping assistance in realistic conditions. We demonstrate that a data driven approach is feasible for predicting unintentional lane departures at least 0.5 seconds into the future, with a recall of 80.28 % and a precision of 80.71 %. We demonstrate that none of the network architectures significantly outperform the others. We also investigated sources of errors in hardly predictable situations, and analyzed the trade-offs between the network structure and prediction performance.

Keywords: Unintentional lane departures, Lane keeping aid (LKA), Lane keeping assistance, Threat assessment, Neural networks, Deep learning, Signal classification.

Acknowledgements

First of all, we would like to thank John Dahl for the incredible support in supervision and the engagement in our work. Gabriel Rodrigues de Campos also deserves credit for all his inputs throughout the project, essentially functioning as an extra supervisor. We would also like to thank Zenuity for having us and supporting us, especially Olle Brännlund and Anders Dahlbäck for helping us with the initial data wrangling. Finally we would like to thank our examiner Jonas Fredriksson for taking on the project.

Rasmus Jonsson & Anton Kollmats, Gothenburg, June 2018

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

Though great improvements have been made in terms of traffic safety in the last decades, many accidents still occur on a daily basis. In the United States, roughly 30,000 fatal accidents occur every year, whereof 2016 alone counted for more than 34,439 fatal crashes [1]. Moreover, as highlighted in different reports, many of such accidents are partially or even completely due to human error. For instance, [2] reports that nine percent of all fatal traffic accidents were due to distracted drivers. To reduce the occurrence of such type of traffic accidents, several "Advanced Driving Assistance Systems" (ADAS) have therefore been developed in the last decades and today there exist a wide range of solutions such as Adaptive Cruise Control (ACC), blind spot indicator, collision avoidance systems and traffic sign recognition, just to mention a few of them. A major challenge for active safety systems is how to accurately predict future events, typically referred to as threat assessment in the literature. Threat assessment methods are necessary in order to assess dangerous situations, and the result from an accurate threat assessment can be used in the decision making of whether or not to activate an ADAS intervention. For a thorough review on threat assessment design challenges and methodologies, see [3].

In cases where the driver is fatigued, drowsy or fails at the driving task, dangerous situations can appear. A typical dangerous situation is when the driver makes an unintentional lane departure which can easily result in a collision with an oncoming vehicle or other obstacles. The authors of [4] approached the problem of predicting unintended lane departures with Artificial Neural Networks (ANNs). In their study, data from fatigued drivers from a simulation environment was analyzed and the method resulted in accurate predictions of unintentional lane departures. However, such work was only based on simulations and thereby neglects important sources of uncertainty inherent to realistic environments. For instance, the performance of typical sensors such as cameras, lidars and radars, is often dependent on weather conditions, range, speed or the visibility of lane markers, which yields a more complex problem when compared to a deterministic simulated environment.

1.2 Problem definition

In this thesis we focus on the problem of how to accurately predict unintended lane departures and, in the positive cases, to determine when the vehicle will leave the lane, see Figure 1.1 for an illustration. More precisely, we want to determine if the distance from the vehicles' front corners to the nearest lane marker (d in Figure 1.1) converges to zero within a given prediction horizon T_{horizon}. In order words, the research question behind this work can be formulated as: *for what time horizon can we accurately predict unintentional lane departures*?

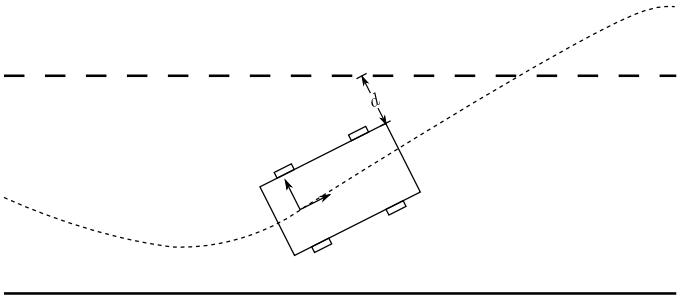


Figure 1.1: The problem under consideration is defined as predicting whether the distance from the vehicle to the closest lane marker, d, unintentionally converges to zero within a prediction horizon T_{horizon}.

1.3 Purpose

The purpose of this thesis is to investigate how one can make accurate predictions of unintentional lane departures. Handling this exact problem, many vehicles already come equipped with Lane Keeping Assistance (LKA) systems consisting of sensors, a threat assessment and decision making system as well as a control unit. An overview of the whole system is illustrated in Figure 1.2.

Our purpose in this work is to develop a threat-assessment and decision system based on the realistic data, that can be used in the future to decide when to trigger the vehicle control system for an evasive maneuver. As mentioned in the background, the authors of [4] make accurate predictions of unintentional lane departures, using a prediction model based on neural networks and simulator-driven data. In addition to that, ANNs have been successfully applied to tasks that are similar from a technical standpoint, such as human activity recognition [5], natural language processing [6] and time series classification [7]. Due to the potential of neural networks and its large usage in recent years, we have decided to investigate an ANN based solution for our problem using realistic data.

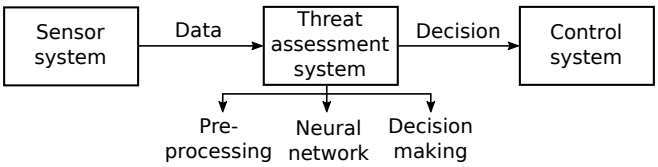


Figure 1.2: An overview of a lane keeping assistance (LKA) system, where the sensor system senses the surroundings, the threat assessment system takes the intervention decision and the control system controls the vehicle.

1.4 Scope and boundaries

The dataset available for our research was recovered with different XC70 Volvos that were driven by professional drivers. In this thesis, we will focus on scenarios where a Lane Keeping Assistance (LKA) intervention could avoid accidents. To complete the scope, we present in the sequel a list of assumptions and boundaries for our study:

1. Only straight roads are considered.
2. The vehicle speed should be higher than 60 km/h.
3. Objects other than the vehicle and the road are disregarded.

2
Theory

In this section we introduce theoretical concepts and definitions regarding artificial neural networks and statistical methods that are relevant for the understanding of our approach.

2.1 Artificial neural networks

The inspiration behind Artificial Neural Networks (ANNs) as computational models comes from the structure of animal brains. Animal brains are composed of millions of interconnected cells called neurons, that communicate by means of electrical signals. If a neuron is sufficiently stimulated by other neurons, it will activate and pass a signal on to its neighbors. The signals propagate over junctions called synapses, whose degree of connection may vary. The stronger the synaptic connections, the stronger the propagated signal. For further information on biological neural networks, see [8]. Inspired by these concepts, the authors of [9] proposed the McCulloch-Pitts neuron as a mathematical model for the artificial neuron.

2.1.1 The McCulloch-Pitts neuron

The McCulloch-Pitts neuron is the basic building block of artificial neural networks. Like animal brains, ANNs are networks of interconnected computing cells. The synapses are represented by a set of weights that determine how sensitive a neuron is to input from a given connection. The propagation through the neuron is represented by an activation function. For a given combination of weights and activation functions, the McCulloch-Pitts neuron maps a set of inputs to an output. More precisely, for a given input vector x_i, the McCulloch-Pitts neuron yields an output given as

$$y = g\left(\sum_i w_i x_i\right), \tag{2.1}$$

where w_i is a weight vector and g the activation function of the neuron, while its output y is commonly referred to as the activation. In terms of the animal brain, the activation is analogous to the signal that is forwarded to the next neuron. Figure 2.1 shows a typical illustration of a McCulloch-Pitts neuron with a two-dimensional input vector.

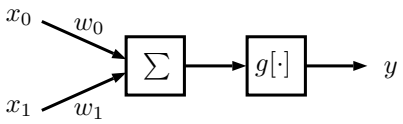


Figure 2.1: The McCulloch-Pitts neuron for a two-dimensional input vector. This unit is the fundamental building block of artificial neural networks.

2.1.2 Feedforward networks

A typical feedforward network has a structure as illustrated in Figure 2.2. Here, the circles represent McCullough-Pitts neurons, the encircled variables are the respective outputs and the connections between neurons are indicated by the arrows. Networks where each neuron of a layer has a unique connection to every neuron in the previous layer are referred to as a fully connected network. The

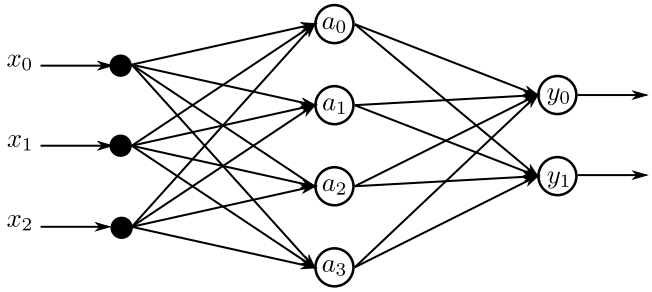


Figure 2.2: A typical feedforward network with three inputs, two outputs and a hidden layer with four neurons. In this example, $a_k^{[0]} = x_k$, $a_k^{[1]} = a_k$ and $a_k^{[2]} = y_k$, where k runs over the number of neurons in each layer.

network takes an input example x_k, $k = 0, 1, \ldots, N$ and N as the number of inputs. The state of the neurons in the middle layer (the intermediate layers are commonly referred to as hidden layers) is calculated by weighting the input(s), adding a bias and passing it through an activation function defined such that the following holds:

$$\begin{aligned} z_j^{[l]} &= \sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}, \\ a_j^{[l]} &= g^{[l]}(z_j^{[l]}). \end{aligned} \tag{2.2}$$

Here $a_j^{[l]}$ denotes the output of neuron j in layer l (also called activation) and $w_{jk}^{[l]}$ is the weight between $a_k^{[l-1]}$ to $z_j^{[l]}$. The activation function is denoted by g and the input $z_j^{[l]}$ is the weighted sum of the neuron inputs plus its bias $b_j^{[l]}$. In a standard feedforward network, the neuron inputs are the weighted outputs of the previous layer. In the case of Figure 2.2, we have $a_k^{[0]} = x_k$, $a_k^{[1]} = a_k$ and $a_k^{[2]} = y_k$, where k runs over the number of neurons in each layer. Note that the biases are set to zero and are not included in the illustration.

It is worth mentioning that when the activation function g is nonlinear and the network has one hidden layer that has a finite but large enough size, there exists a set of weights such that the neural network can estimate any continuous function on a compact subset of $\mathbb{R}^n$, as proven in [10].

2.2 Training networks

Neural networks are trained to emulate an underlying target function, given a set of known input/output pairs. Generally speaking, a well trained network is one that generalizes to make correct predictions on data that it has not been trained with. In training, the dataset is usually split into two subsets: one set used for training and the remaining for testing. As explained in [11], a neural network is trained by feeding it examples and adjusting its weights towards better performance. The inputs of neural networks are sometimes referred to as samples, but for the remainder of this thesis *we will refer to network inputs as examples in order to distinguish them from time samples* of vehicle signals. Note that during training, a small part of the training set is kept and is referred to as the validation set and is used for validation during the training phase. Theses set are further elaborated in Section 3.2.1.

The training phase validation is useful because it shows how the network is progressing during training. In particular, the validation set shows whether the network generalizes well or not, since the validation set is not used for for the backpropagation. Therefore, even if the training performance continues to increase the validation performance might decrease, which is a sign of poor generalization (see the brief discussion in Section 2.4). If the performance is sufficiently high, it might be advantageous to stop training as overfitting commences. This is referred to as early stopping. It is common practice to train the network in **epochs**, which include a number of steps (updates of the weights). After each epoch, the validation set is used to evaluate the network.

To quantify the performance of a network, a loss function is used to measure the discrepancy between the target values and the predictions. The weights are then iteratively updated by shifting them in the negative direction of their gradient.

2.2.1 Loss functions

Neural networks are trained to produce a minimal loss, sometimes referred to as a cost. The loss is calculated by comparing the network outputs to the target value (i.e., the expected output) by means of a *loss function*. One of the most common loss functions is the Mean Squared Error (MSE). For simplicity, the following equations are written for one-dimensional outputs. This yields that the MSE loss function is given as

$$J_{\text{MSE}}(\hat{y}, y) = \frac{1}{M} \sum_{i=1}^{M} \left(y^{(i)} - \hat{y}^{(i)} \right)^2, \tag{2.3}$$

where M is the number of samples, $\hat{y}^{(i)}$ and $y^{(i)}$ are the predicted output and the target for the ith sample, respectively. However, a general consensus is that the cross entropy is a better loss function for classification problems, see for example [12, 13]. In particular, and given that we have a binary classification problem, the binary cross entropy the loss can be computed as

$$J_{\text{BCE}}(\hat{y}, y) = -\frac{1}{M} \sum_{i=1}^{M} \left[y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)} \right) \log \left(1 - \hat{y}^{(i)} \right) \right]. \tag{2.4}$$

2.2.2 Backpropagation

Backpropagation is a gradient-based method for minimizing the loss of a neural network. Using the notation of (2.2), backpropagation revolves around finding the gradient of a loss function J with respect to the weights $w_{jk}^{[l]}$. Once the gradient is known, a minimization algorithm such as gradient descent is typically used to find a set of weights that minimize the loss, see Section 2.4.6. According to [11], the popularity of backpropagation stems from its conceptual simplicity, computational efficiency and the fact that it often produces good results.

While general versions of backpropagation exist [11, 14], in this thesis we focus on backpropagation in the case of traditional multi-layered networks. For such networks, and considering the notation of (2.2), the backpropagation algorithm can be summarized by the following recurrence relations:

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^{[l]}} &= \frac{\partial J}{\partial a_i^{[l]}} g^{[l]\prime} \left(z_i^{[l]} \right) a_j^{[l-1]}, \\ \frac{\partial J}{\partial a_m^{[l-1]}} &= \sum_n \frac{\partial J}{\partial a_n^{[l]}} g^{[l]\prime} \left(z_n^{[l]} \right) w_{nm}^{[l]}, \end{aligned} \tag{2.5}$$

where $g^{[l]\prime}$ is the derivative of activation function $g^{[l]}$. These relations provide the means of recurrently calculating the total gradient, given that the network output is known. Thus, one must first pass the examples as inputs through the network (forward pass) before the error can be propagated back (backward pass). In practical terms, for a network of L layers, the boundary values of the recurrence would be the inputs and outputs $x_j = a_j^{[0]}$ and $\hat{y}_j = a_j^{[L]}$, respectively.

Proof. The weight update aims at finding the rate of change of the error J when an arbitrary weight in the network is changed and everything else is kept fixed. To derive an expression for that, we momentarily assume that the partial derivative of J with respect to the output of an arbitrary neuron, $a_i^{[l]}$, is known. From the chain rule and (2.2), we obtain

$$\frac{\partial J}{\partial w_{ij}^{[l]}} = \frac{\partial J}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}} = \frac{\partial J}{\partial a_i^{[l]}} \frac{\partial}{\partial w_{ij}^{[l]}} \left[g^{[l]} \left(z_i^{[l]} \right) \right] = \frac{\partial J}{\partial a_i^{[l]}} g^{[l]\prime} \left(z_i^{[l]} \right) \frac{\partial z_i^{[l]}}{\partial w_{ij}^{[l]}}. \tag{2.6}$$

Using the expression for $z_i^{[l]}$ from (2.2), the final factor from (2.6) becomes

$$\frac{\partial z_i^{[l]}}{\partial w_{ij}^{[l]}} = \frac{\partial}{\partial w_{ij}^{[l]}} \left[\sum_n w_{mn}^{[l]} a_n^{[l-1]} \right] = a_j^{[l-1]}, \tag{2.7}$$

where the last equality holds as everything except $w_{ij}^{[l]}$ is kept constant. Substituting equation (2.7) back into (2.6), we obtain the first expression of (2.5) such that:

$$\frac{\partial J}{\partial w_{ij}^{[l]}} = \frac{\partial J}{\partial a_i^{[l]}} g^{[l]\prime} \left(z_i^{[l]} \right) a_j^{[l-1]}. \tag{2.8}$$

Note that to derive the above we assumed knowledge of Js derivative with respect to the output of an arbitrary neuron. Once again, using the chain rule, we establish a relation between the outputs of consecutive layers as follows:

$$\frac{\partial J}{\partial a_m^{[l-1]}} = \sum_n \frac{\partial J}{\partial a_n^{[l]}} \frac{\partial a_n^{[l]}}{\partial a_m^{[l-1]}}. \tag{2.9}$$

Substituting $a_i^{[l]}$ from (2.2) and using the fact that everything is kept fixed, we obtain the second expression of (2.5) as:

$$\frac{\partial J}{\partial a_m^{[l-1]}} = \sum_n \frac{\partial J}{\partial a_n^{[l]}} \frac{\partial}{\partial a_m^{[l-1]}} \left[g^{[l]\prime} \left(z_n^{[l]} \right) \right] = \sum_n \frac{\partial J}{\partial a_n^{[l]}} g^{[l]\prime} \left(z_n^{[l]} \right) w_{nm}^{[l]}, \tag{2.10}$$

which concludes the derivation. □

2.3 Network structures

Other ways of improving the performance of a network is to leverage prior knowledge about the dataset into its design. Time dependency, for instance, is often incorporated by using Recurrent Neural Networks (RNNs).

2.3.1 Recurrent neural networks

RNNs belong to a class of ANNs that form directed cycles. RNNs model sequential relationships by taking advantage of temporal correlations between neurons. Specifically, they are used to model the following scenario. Suppose a sequence $x = \{x_0, x_1, \ldots, x_T\}$ and a corresponding output $y = \{y_0, y_1, \ldots, y_T\}$ and assume that we want to learn a mapping $f : x \to y$. An RNN exploits a hidden state h_t that is not only dependent on the current input x_t but also relies on the previous hidden state h_{t-1}. We can express the hidden state h_t as

$$h_t = g(h_{t-1}, x_t), \tag{2.11}$$

where g is the nonlinear activation function. The hidden state h_t enables the network to store information about the whole sequence. In summary, RNNs can use the hidden states as memory in order to capture long-term dependencies. Suppose the following RNN model which, as illustrated in Figure 2.3, is similar to the ANN with additional intermediate connections. The forward pass of the network can then be expressed as:

$$\begin{aligned} h_t &= g^{(1)}(w_{hh} h_{t-1} + w_{xh} x_t + b_h), \\ y_t &= g^{(2)}(w_{hy} h_t + b_y), \end{aligned} \tag{2.12}$$

where $g^{(\cdot)}$ are nonlinear activation functions, w_{hh} the weights between the hidden states, w_{xh} the weights from the input to the hidden states and w_{hy} the weights from the hidden states to the outputs. Moreover b_h and b_y represent the corresponding biases for the hidden state and the output.

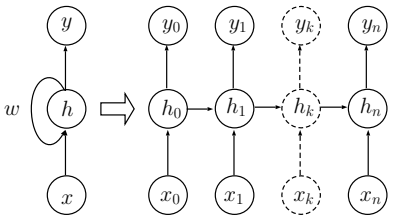


Figure 2.3: To the left is a simple recurrent neural network (RNN) layer. The layer is unfolded to the right to illustrate the additional information flow introduced by the intermediate connections. This is what makes the difference between the feedforward network and the RNN architecture.

The backpropagation of the RNN layer is done similarly as in Section 2.2.2 but the derivatives are taken with respect to w_{hh}, w_{hx} and w_{yh} and altered with a time component. For a full derivation of the RNN backpropagation the reader can refer to [15].

The simple RNN suffers from the exploding gradient problem which can be solved by thresholding the gradient. It also suffers from the vanishing gradient problem and therefore the network is difficult to train, as shown in [16]. These problems can however be tackled by the long short-term memory (LSTM) or the gated recurrent unit (GRU) cells presented in the following sections.

2.3.2 Long short-term memory

The Long Short-Term Memory (LSTM) cell was invented by the authors of [17] in order to address the issues of vanishing gradients. The idea behind LSTM is that the gradient has its own flow through the chains of cells, which allows it to propagate through the network without interruption, see Figure 2.4.

The LSTM cell is composed of four different gates that regulate the flow of signals through the cell. The i gate, known as the input gate, determines whether to write to the internal cell, the f gate determines how much of the internal cell is forgotten, the o gate determines how much to reveal to the cell, and g gate determines how much to write to the cell. How the gates are determined can be seen in the following equation:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}. \tag{2.13}$$

In the above equation we use a shorthand for applying different activation functions to different parts of the vector. The W matrix incorporates weight matrices that are linear mappings from the hidden state h_{t-1} and input x_t to the different gates. The internal state c_t and the output state h_t is then given as

$$\begin{aligned} c_t &= f \odot c_{t-1} + i \odot g \\ h_t &= o \odot \tanh(c_t) \end{aligned} \tag{2.14}$$

where $\odot$ is the entrywise product and $\sigma(x)$ is the logistic sigmoid function $\frac{1}{1+e^{-x}}$. An illustration of the cell can be seen in Figure 2.4.

The internal state c_t prevents the gradient from vanishing. As seen in Figure 2.4, when backpropagating with respect to c (from c_t to c_{t-1}), the only "interruption" of the gradient flow is the single entrywise multiplication by the forget gate f_t. Whereas normally, however, the backpropagation involves repeated multiplications with small activations and it is the riddance of this that prevents the gradient from vanishing. In the LSTM cell we do not repeatedly multiply the same weight matrix as in the simple RNN, which is the reason for the exploding or vanishing gradient. Another reason for why the gradient does not vanish or explode is that the gradient is only propagated through a single nonlinear activation function. For further details refer to [17].

[1] This figure is inspired by `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

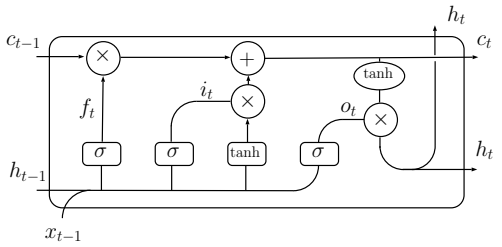


Figure 2.4: An illustration of the LSTM cell[1]. The hidden state h_{t-1} and the internal state c_{t-1} comes from the previous cell together with the input x_{t-1}. The different inputs are then processed as illustrated, where σ is the sigmoid function, tanh is the hyperbolic tangent function, to produced the new outputs h_t and c_t.

2.3.3 Gated recurrent unit

The authors of [18] present a simpler alternative to the LSTM cell that still has the property of adaptively remembering and forgetting information. The authors construct the cell in the following way. A *reset gate*, denoted as r_j, is introduced and given as

$$r_j = \sigma([w_{xr}x]_j + [w_{hr}h^{(t-1)}]_j + [b_r]_j), \tag{2.15}$$

where σ denotes the sigmoid function, x the input vector, h_{t-1} the previous hidden state, and w_{xr} and w_{hr} are the weight matrices for the input and previous states, respectively. The *update gate* z_j, which determines how the hidden unit is updated, is computed as

$$z_j = \sigma([w_{xz}x]_j + [w_{hz}h^{(t-1)}]_j + [b_z]_j), \tag{2.16}$$

while the activation of the hidden unit $h_j^{(t)}$ is given by

$$h_j^{(t)} = z_j \odot h_j^{(t-1)} + (1 - z_j) \odot \tilde{h}_t, \tag{2.17}$$

where $\odot$ denotes the entrywise product and

$$\tilde{h}_j = \tanh([w_{xh}x]_j + [w_{hh}(r \odot h^{(t-1)})]_j + [b_h]_j). \tag{2.18}$$

For the sake of clarity, a brief description of the cell's principles is given as follows. When the reset gate goes to 0, the hidden state is forced to ignore the previous hidden state and align itself with the current input. This enables the cell to drop any information which is not relevant for the future, using a simpler structure than the LSTM cell.

2.3.4 Temporal convolutional networks

Convolutional Neural Networks (CNNs) are a class of feature extraction structures invented by [19]. The essential part of the CNN structure is that a filter is convolved over the inputs, where the filter is a set of weights that the network learns by itself. In our case, with a dataset of time dependent signals, we use an architecture with one dimensional filters, which is referred to as a Temporal Convolutional Network (TCN). The signals have the structure $a_i = [a_1, \ldots, a_N]$, where N is the length of the signals. The signals are then convolved with a filter of size M such that:

$$\begin{aligned} z_j^{[l+1]} &= b_j^l + \sum_{m=1}^{M^{[l]}} w_m^{[l,j]} * a_{i+m-1}^{[l,j]}, \\ a_j^{[l+1]} &= g^{[l+1]}(z_j^{[l+1]}), \end{aligned} \tag{2.19}$$

where l represents the layer index, g the activation function, and b_j and $w_m^{[l,j]}$ the bias term and weight respectively for a feature map j.

2.4 Training enhancement

Neural networks are considered to be hard to train and several techniques have been developed to improve the networks' convergence. In this section we introduce several techniques for preprocessing, regularization and optimization as methods for convergence improvement.

The authors of [20, 21] explain the phenomena of under- and overfitting which are common causes of poor generalization. Underfitting occurs when the chosen model is too simple to capture the underlying target function. On the other hand, overfitting occurs when an overly complex model is chosen. A sign of overfitting is that the model performs well on the training samples but fails to generalize to similar but unseen data.

Another phenomena that makes neural networks hard to train, especially deeper architectures with several layers, is the vanishing and exploding gradient problem. The authors of [22] go into details on how this effects the training. The problem of vanishing gradients can be summarized as follows. As it can be seen in (2.5), backpropagation is essentially a repeated multiplication of activation functions whose derivatives are typically smaller than one. If the weights are small, the total gradient in (2.2) can therefore vanish in the case of deep network architectures, which in turn causes the weights to stagnate. If, on the other hand, the weights are very large, the problem of exploding gradient appears. In this case the weight factor in (2.5) dominates the gradient, leading to large fluctuations in the updates.

In the following chapters, we describe several methods to tackle the above mentioned problems.

2.4.1 Input normalization

As stated in [11], an input signal with a non-zero mean would promote weight updates in a given direction more than others, resulting in low efficiency. The authors also point out that normalization of the input values increases the rate of convergence. Likewise, inputs with equal variance increase the rate of convergence. Zero mean and unit variance is obtained by the following transformation

$$x_i' = \frac{x_i - \mathrm{E}[x_i]}{\sqrt{\mathrm{Var}(x_i)}}. \tag{2.20}$$

Have $\mathrm{E}[x]$ and $\mathrm{Var}(x)$ denote the mean and variance for each input variable x_i in the training set, respectively. Using the mean and the variance of the training set, the same transformation has to be applied for any future inputs to the network (including the test set). Consequently, we must assume that the training set captures the underlying distribution wellso that future inputs follow the same mean and variance. In certain cases, as pointed out in [11], some input variables are likely to be more important than others and could be scaled so to increase the rate of convergence.

2.4.2 Activation functions

A key component in neural networks is the activation function. They are typically nonlinear functions and a wide array of functions exist. In this work, we introduce and evaluate some of the most popular ones.

The sigmoid function was one of the first activation function used for neural networks. It can be seen in equation (2.21) and it squashes the output from the neuron into the range $(0, 1)$.

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.21}$$

As previously mentioned, deep networks require repeated multiplications of the activation derivative. In the case of the sigmoid, the derivative is small and therefore the vanishing gradient problem is inherent during backpropagation. The authors of [23] demonstrate that the Rectified Linear Unit (ReLU) activation function makes it easier to train deep networks. The ReLU activation function is given in (2.22). Since the gradient is always 1 for $x \geq 0$, the gradient is less prone to vanish.

$$f(x) = \begin{cases} 0 & \text{for } x < 0, \\ x & \text{for } x \geq 0. \end{cases} \tag{2.22}$$

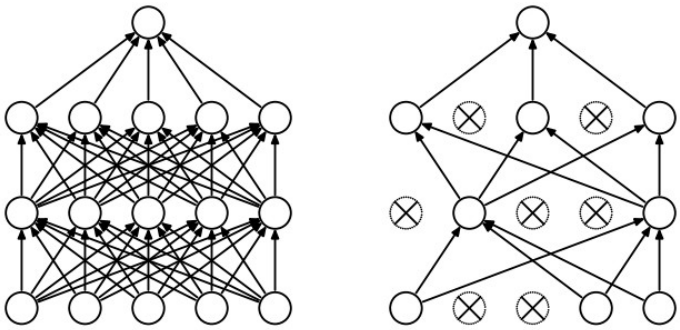
For faster training and higher classification results, the author of [24] invented the Exponential Linear Unit (ELU) which can be seen in equation (2.23). ELU has the same properties as ReLU for dealing with the vanish gradient problem but introduces an exponential part for $x < 0$. The exponential part enables the ELU function to push the input to the next layer towards a zero mean. The outcome is similar to the one of batch normalization (see Section 2.4.4), but requires fewer parameters.

$$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0, \\ x & \text{for } x \geq 0. \end{cases} \tag{2.23}$$

2.4.3 Dropout

Dropout is a technique that was first proposed in [25] and is applied to networks to avoid overfitting or co-adaptation of the units. As explained in [26], co-adaption means that several neurons have learned a complex relationship which can prevent the individual improvement of the neuron. The inspiration of applying dropout originates from theories on the role on sex in evolution. Among the most highly evolved organisms, sexual reproduction is a major part of the evolution. Sexual reproduction is a way of break up the co-adaptions of the genes, where some gene subsets get (randomly) chosen and others are dropped. In an analog way, by making random selections of neurons, their individual performance can be increased.

Dropout is implemented by randomly dropping a percentage of the neurons during the training phase. A neuron is dropped with the probability p during forward and backpropagation, see Figure 2.5 for an illustration of the dropout technique. Dropout can be added layerwise with an individual probability. For a discussion around the details of the dropout technique, the reader can refer to [26].



(a) Standard Neural Net (b) After applying dropout.

Figure 2.5: The difference between a standard neural network during training compared to a network with dropout. Figure taken from [26].

2.4.4 Batch normalization

The training of deep neural network architectures (i.e., networks with several hidden layers) is complicated because weights are simultaneously updated. During training, the weights change in order to: (1) learn the properties of a new example and (2) adjust for the fact that the previous layers are learning too. The latter has the consequence that a hidden layer may perceive the same example in two completely different ways because the weights of the layers before it has changed. In turn, this slows down the learning process.

The authors of [27] have proposed a solution to this problem which revolves around the normalization of the inputs of every layer in a similar manner to the one described in Section 2.4.1. The authors propose that the layer inputs are normalized after every batch of data, meaning that the distribution of each layer's input remains relatively constant. This allows for higher learning rates and, in turn, to faster convergence, see [27].

2.4.5 Weight initialization

If the starting weights are too small, the layer outputs will vanish in deeper layers. On the other hand, if they take too high values, the signal will grow quickly during the forward pass, making the outputs oscillate back and forth between extreme values. By initializing the weights in a favorable manner, a significant boost to the convergence speed can be achieved. Two common weight initialization schemes are the Xavier [22] and He [28] initializers.

The Xavier is designed for and performs well with sigmoidal activation functions. The weights are sampled from the distribution defined by

$$\mathcal{D}\left(0, \frac{2}{N_{\text{in}} + N_{\text{out}}}\right), \tag{2.24}$$

where N_{in} and N_{out} are the number inputs and outputs to a layer, respectively. In the above, $\mathcal{D}$ is either the normal or uniform distribution, see [22] for further details.

The He initializer is designed for ReLU activation functions and gives a significant improvement for those types, as explained in [28]. The He initializer is a tweaked version of the Xavier initializer, where the weights are instead sampled from the distribution given by:

$$\mathcal{N}\left(0, \frac{2}{N_{\text{in}}}\right), \tag{2.25}$$

where N_{in} is the number of inputs to the layer and $\mathcal{N}$ is the normal distribution. See [22] for further details.

2.4.6 Optimizers

Optimizers for neural networks determine how weights in the different layers are updated. The simplest one is gradient descent:

$$w^{(t+1)} = w^{(t)} - \eta \frac{\partial J}{\partial w}, \tag{2.26}$$

where η is a chosen step length. Variable η is one of the networks tunable hyperparameters that can affect the rate of convergence. The gradient descent can be done in two ways: (1) take one or several training examples at random (the latter is called a batch) and calculate the error or (2) propagate all training examples. The first method is called Stochastic Gradient Descent (SGD) and the second is referred to as Gradient Descent (GD) learning. SGD has slower convergence when the gradient is large in one dimension and lower in another. It is possible to circumvent this by adding a momentum term to the SGD. This reduces the chance of converging to local minimum as well as the tendency for oscillations along the direction of descent. See [29] for further information. The analogy of the method is to imagine pushing a ball down a hill where the ball accumulates momentum until it reaches its terminal velocity. In order to incorporate momentum, the above equation can be modified to

$$\begin{aligned} v_{t+1} &= \gamma v_t + \eta \frac{\partial J}{\partial w}, \\ w^{(t+1)} &= w^{(t)} - v_{t+1}, \end{aligned} \tag{2.27}$$

where γ is a number between 0 and 1 that decides the impact of the momentum term.

2.5 Evaluation metrics

In order to evaluate the performance of different network architectures, relevant metrics are necessary. Here, we introduce the confusion matrix concept together with the accuracy, recall and precision metrics.

2.5.1 Confusion matrix

A statistical overview of how well the network is able to classify lane departure events can take the form of a confusion matrix, which is a table mapping the predicted class versus the true class. Given a Lane Departure Event (LDE), if the classification is correct it is called true positive (TP) and false negative (FN) otherwise. If we have a non lane departure event, if it is predicted as a non lane departure event it is called true negative (TN) and false positive (FP) otherwise. An example can be seen in Table 2.1.

Table 2.1: Confusion matrix of lane departure events (LDE) and non lane departure events (non-LDE). When using the matrix, the entries ("true positives", and so on) will be replaced by the corresponding number of cases.

		Ground truth	
		LDE	**Non-LDE**
Prediction	**LDE**	True positives	False positives
	Non-LDE	False negatives	True negatives

2.5.2 Metrics

From the confusion matrix presented before several performance metrics can be derived. The rate of correct predictions (both LDEs and non-LDEs) is reflected by the accuracy, defined as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}. \tag{2.28}$$

However, accuracy alone does not usually give the entire picture. In our case, it is expected that the number of non-LDEs signficantly outnumbers the number of LDEs under regular driving circumstances. Therefore, we introduce the recall, precision and false positive rate as follows:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{False positive rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{2.29}$$

Recall indicates how many lane departures the network was able to predict among the true number of departures, i.e., how many of the actual cases it catches. Precision indicates how many of the positive predictions were correct and, intuitively, a high precision indicates that the LDEs are not caught by chance in a random flurry of positive predictions. The false positive rate is the rate of falsely predicted LDEs divided by all non-LDEs, and gives an appreciation of how often the system reacts when it is not supposed to.

Hence, if the recall is high and the precision is low the network is overly prone to accurately predict an event. On the other hand, if the recall is low and precision is high the network rarely predicts an event correctly.

3
Method

In this chapter we will describe our design concepts and machinery for automated detection of unintentional lane departures. We propose here a threat assessment system (TAS) for lane keeping assistance (LKA) applications. A simplified illustration of a TAS and its sub-parts is shown in Figure 3.1. To achieve the results presented in this thesis, we evolved trough the following steps: (1) selecting relevant data, mainly concerning lane departure events (LDEs); (2) preprocessing the data; (3) selecting and training neural networks; (4) evaluating the neural networks' performance.

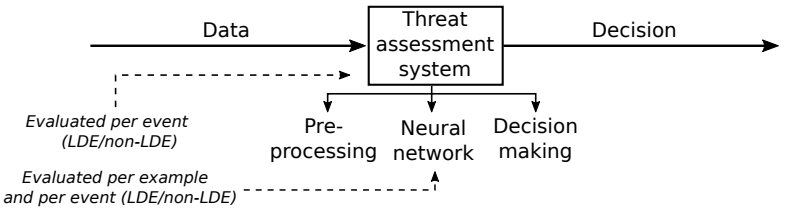


Figure 3.1: Overview of our proposed threat assessment system (TAS). Its components include data preprocessing, a neural network (NN) and a decision making block (e.g., decision threshold). A suitable NN was found by evaluating several networks on both training examples as well as more generic driving, while the TAS as a whole was evaluated on generic driving only.

3.1 Dataset and preprocessing

The dataset analyzed in this work was generated by a fleet of several Volvo XC70 driven by professional drivers for over more than 4000 hours. The drivers were given no special driving instructions, but only requested to drive normally. To simplify our setup, we focused on a subset of the data where the conditions were favorable, and, for training purposes, we chose primarily situations where the vehicle is about to leave the lane. Both matters are explicitly discussed in the following subsections. Moreover, we will begin with a short description of the tools used to handle data in a structured manner.

3.1.1 Toolchain and data pipeline

Throughout our work we established several stepping stones that allowed us to work with the data in a incremental manner. This enabled us to progressively adjust our methodology as we learned more about the dataset. This resulted in a number of tools and the most significant ones are shown in Figure 3.2 and summarized as follows:

- The *signal extractor* enables us to extract a subset the dataset, as explained in Section 3.1.2, significantly reducing storage needs.
- The *splitter* divides the dataset as described in Section 3.2.1. In addition to that, it turned out to be a natural place to perform input normalization as described in Section 2.4.1.

The remainder of the pipeline concerns training and evaluation. The training block relies on a *loader* module, which is responsible for the final step of converting the data to a format that is suitable as an input for a neural network. The results are produced by evaluating the network itself, but also by using a *simulator* that considers a wider dataset. The datasets are further described in Section 3.2.1. The simulator operates in a similar fashion to the loader with respect

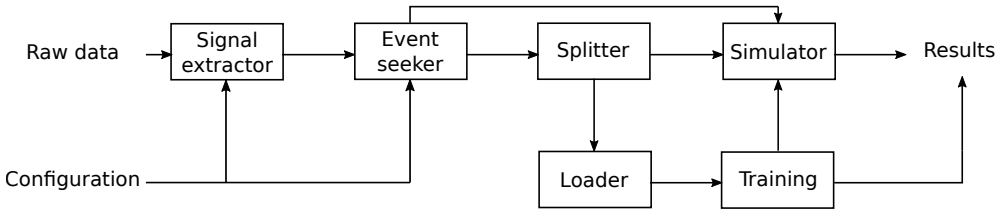


Figure 3.2: Overview of the used data pipeline. The leftmost two components were used for extracting relevant subsets of the data, while the rightmost four were used for preprocessing, training and evaluation.

to the inputs, with the exception that it is able to consider a wider range of situations. This is further elaborated in Section 3.2.4 and Section 3.3.

3.1.2 Signal selection

The dataset contained a wide array of signals. To simplify our problem, we focused on only a few of them. We selected the same signals as the ones used in [4], as well as additional signals that could provide further insights on the lane departing maneuvers.

Table 3.1 shows the signals that were extracted from the data logs and available for use. The lane marker estimates had been recovered in advance using the vehicle cameras and came coupled with an undisclosed measure of quality. Some of the signals were used directly as inputs to the networks, while others were used to identify relevant scenarios in the overall dataset. We describe this in the following sections.

Table 3.1: Signals parsed from the log data. Some of the signals were used for finding relevant data, while others were used as inputs to the neural networks.

Signal	Symbol	Comment
Vehicle yaw rate	ψ	-
Vehicle yaw	θ	Derived from ψ and l_1 or r_1 using numerical integration.
Vehicle speed	v	-
Vehicle acceleration	a	Derived from v using finite differences.
Steering wheel angle	α	-
Gas pedal position	γ	-
Indicator lights	λ	-
Left lane marker estimates	l_0, l_1, l_2, l_3	Polynomial coefficients of the 3rd degree,
Right lane marker estimates	r_0, r_1, r_2, r_3	measured in the rest frame of the vehicle.

3.1.3 Lane keeping assistance conditions

Since the data mainly includes normal driving behaviors, we simplified the problem by: (1) removing low quality data and (2) removing situations that would be unreasonably difficult to handle. Therefore, we defined a set of conditions to be fulfilled for our threat assessment system to be active. More precisely, we introduced an artificial "LKA availability signal" that was toggled when the following criteria are satisfied:

1. The longitudinal speed of the vehicle is greater than $v_{\min}$.
2. The road is straight: the radius of curvature is greater than $R_{\min}$ meters.
3. The width of the road is constant; the width does not change more than $\Delta w_{\min}$ meters.
4. The estimates of the lane markings are of maximum quality.

The above conditions and the specific values were chosen incrementally as we examined results and located sources of poor performance. Likewise, the specific values for $v_{\min}$, $R_{\min}$ and $\delta w_{\min}$ were chosen empirically. In practice, these conditions can be thought of as measures to: (1) avoid urban traffic, (2) avoid voluntary curve cutting scenarios, (3) avoid inconsistencies in the lane marker estimates and to avoid transitions from one to two lane segments (4) ensure that the system has reliable information.

Let $\mathcal{D}_{\text{LKA}}$ denote the subset of data where the LKA availability signal holds true. This will be further elaborated in Section 3.2.1 and in particular in Figure 3.3.

3.1.4 Lane departure events

We defined an LDE as when one of the front corners of the vehicle intersects a lane marking. Secondly, the lane departure events were further subdivided into two categories: intentional and unintentional. An *unintentional lane departure event* was defined as one that fulfills the following criteria:

- One of the front corners of the vehicle intersects a lane marking (i.e., a lane departure event).
- The indicator lights have not signaled lane change within T_{before} seconds prior to the lane departure event.
- The vehicle does not complete a lane change within T_{after} seconds after the lane departure event. A lane change is completed when more than half the lateral width of the vehicle has crossed the marking.

Given the above definition, the remaining lane departure events were classified as *intentional lane departure events*. For practical reasons, we shall define *the event window of an unintentional LDE*s as T_{before} seconds prior and T_{after} seconds after the event itself.

3.1.5 Input normalization

As touched upon in Section 3.1.1, all signals were normalized to zero mean and unit variance. This was motivated by the principles described in Section 2.4.1.

3.2 Finding a network

The inputs of neural networks are sometimes referred to as samples, but for the remainder of this thesis *we will refer to network inputs as examples in order to distinguish them from time samples* of vehicle signals.

Several network architectures and prediction horizons were considered. As pointed out in Section 2.2, we will refer to the neural network inputs as *examples* to distinguish them from time *samples* in the context of vehicle signals. All of the networks were fed examples of similar characteristics, namely, sets of short time series of some or all signals described in Section 3.1.2.

To train and make an initial evaluation of the networks' performances, we used the type of data split described in Section 2.2. The networks were trained, validated and tested on examples extracted from the neighborhood of LDEs. Intuition lead us to believe that this would be the most efficient way of training, since that is where ambiguous situations are likely to arise. This intuition is supported by [14]. This way of evaluating networks was useful for comparing various network architectures.

Thereafter, we increased the prediction horizons for the best network and measured its performance. In order to correctly assess the performance, we had to introduce a more realistic way of evaluation: simulation on generic data that included non-LDE driving and excluded data occurring after LDEs or after situations where an LKA system would have intervened.

3.2.1 Splitting data

As explained in Section 2.2, it is necessary to use separate subsets for training and evaluation purposes. The evaluation is done in steps and require several data partitions: (1) one set for *validation* during training, used for early stopping; (2) one for *testing* the network after training and (3) one for the *simulation* designed to evaluate the threat assessment system as a whole.

Let the training set be denoted by $\mathcal{D}_{\text{train}}$, the validation set by $\mathcal{D}_{\text{val}}$ and the test set by $\mathcal{D}_{\text{test}}$, be such that

$$\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{test}} = \{W : W \in \mathcal{D}_{\text{LKA}}\}, \tag{3.1}$$

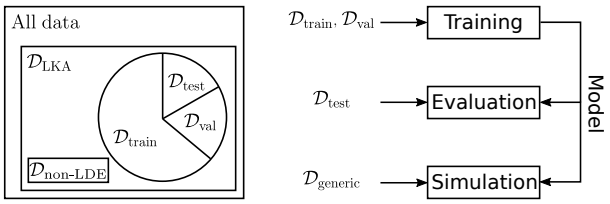


Figure 3.3: The left hand part of the figure represents the set of all log data considered, and divided into subsets. $\mathcal{D}_{\text{LKA}}$: the conditions in Section 3.1.3 are fulfilled; $\mathcal{D}_{\text{train}}$: used for explicit training, $\mathcal{D}_{\text{val}}$: data used for validation and $\mathcal{D}_{\text{test}}$: used for evaluation/simulation together with $\mathcal{D}_{\text{non-LDE}}$ and $\mathcal{D}_{\text{generic}}$. The right-hand part of the figure shows how the data is used for the different modelling steps. The specifics are detailed in Section 3.2.3 and Section 3.3.

where W denotes the event window of an unintentional LDE and $\mathcal{D}_{\text{LKA}}$ is defined as the set of data where the LKA signal holds true (see Section 3.1.3). Effectively, the union above contains all unintentional lane departure events that our threat assessment system aims to predict. Additional data is necessary for evaluating the complete threat assessment system on generic driving, see Section 3.3. Therefore, an additional dataset, denoted by $\mathcal{D}_{\text{generic}}$, is defined as

$$\mathcal{D}_{\text{generic}} = \mathcal{D}_{\text{test}} \cup \mathcal{D}_{\text{non-LDE}}, \tag{3.2}$$

where $\mathcal{D}_{\text{non-LDE}}$ is composed of data where the LKA is active but no LDEs occur. $\mathcal{D}_{\text{non-LDE}}$ contains 6,5 % of the total driving and have not been used for training the networks. The dataset compositions and their use are illustrated in Figure 3.3.

3.2.2 Formatting input examples

As mentioned above, the networks were fed time series examples taken from the signals described in Section 3.1.2. More specifically, the examples featured a selection of signals of length N_{lags} including the LKA availability signal described in Section 3.1.3.

During the training and evaluation of the networks we relied on examples taken from the neighborhood of the LDEs, taken from the sets $\mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{val}} \cup \mathcal{D}_{\text{test}}$. In other words, we swept a *sample window* of size N_{lags} over the event windows described in Section 3.1.4. The target outputs of the examples were assigned ones T_{horizon} seconds prior to the LDE and onwards. Earlier than that, the target values (i.e., the expected values) are assigned 0. This endeavor is depicted in Figure 3.4.

Given a sample period of T_{period}, the number of generated examples is given as

$$M = \left(\frac{T_{\text{before}}}{T_{\text{period}}} - N + 1 \right) N_{\text{LDE}}, \tag{3.3}$$

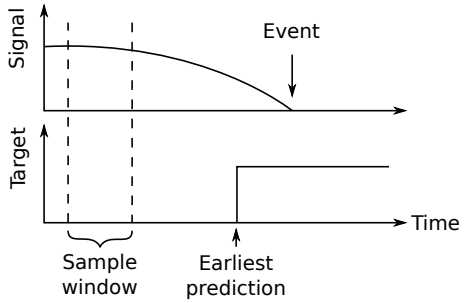


Figure 3.4: Illustration of signal to sample construction. Each LDE window (see Section 3.1.4) is augmented with a target signal. In this case it is a boolean set to true shortly prior to the event. A *sample window* is swept over the entire event window, and each instance of the sample window is turned into a network example. The present signal and target values are taken at the sample windows right-hand boundary.

where N_{LDE} denotes the number of unintentional lane departure events, and T_{before} is defined as the number of seconds prior to the lane departure event in the sample window. See Section 3.1.4. Note that the loader block in Figure 3.2 is responsible for providing the network with the examples.

3.2.3 Architecture selection

We used intuitions gained from Section 2.2 and Section 2.3 to derive different network architectures. Then, we evaluated the networks and tuned them to improve performance. In a first step, we use the losses obtained during training and validation to quickly discard unfit architectures (see Section 2.2).

While the architectures are comparable with each other using the test loss obtained when feeding the network the examples from $\mathcal{D}_{\text{test}}$, it is nevertheless difficult to get a feeling for the true performance. To counter this, we used the metrics described in Section 2.5.1. Since the network outputs a floating point value between 0 and 1 (and not a boolean), we introduced a *decision threshold* τ. If the network outputs a value greater than τ we consider that as a positive case and a negative one otherwise. In other words, an LKA system based on the threat assessment in Figure 3.1 would make the decision to intervene if the network output exceeds τ. With the above metric, the network predictions and targets of each example contributed to an entry in the confusion matrices.

In order to train the networks, we generated examples from the set $\mathcal{D}_{\text{train}}$. The set $\mathcal{D}_{\text{val}}$ was used throughout the training as an independent set for evaluating the performance during training. This allowed us to interrupt training if the network started to overfit on the training data.

As described in Section 2.5.2, a well performing network is one where recall and precision are high and balanced. This definition constituted the basis for selecting one of the architectures described below. They were all implemented in Keras [30] and ran on a computer with an Intel i7 CPU, 32 GB RAM and Geforce GTX 1050 Ti GPU.

3.2.3.1 Multilayer perceptron

The Multilayer Perceptron (MLP) is a feedforward network of at least three layers of neurons. The MLP is a general function approximator as described in Section 2.1.2. As a baseline, we used the MLP described in [31]. Thereafter, we investigated several MLP variations including a number of different enhancement techniques (see Section 2.4). For example, we evaluated different activation functions such as sigmoid, Rectified Linear Unit (ReLU) and Exponential Linear Unit (ELU), regularization techniques such as dropout and batch normalization, as well as layers sizes ranging from 64, 128 and 256 neurons in each layer. The general structure of the MLPs is illustrated in Figure 3.5.

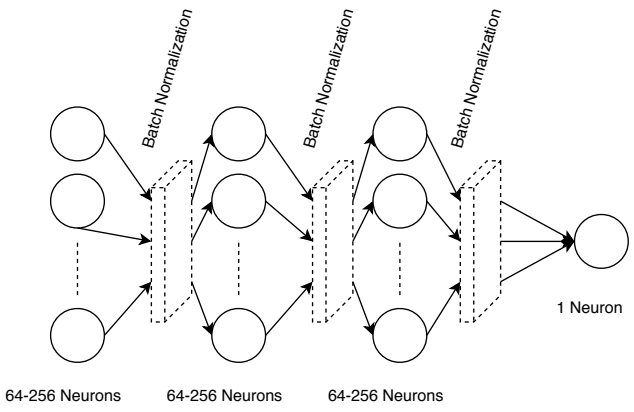


Figure 3.5: Multilayer perceptron structure with 64, 128 or 256 neurons in the first layers with optional batch normalization layers in between, connected to a single neuron for prediction.

3.2.3.2 Long short-term memory network

Since the data is has time series format, we can infer the time dependency by introducing a recurrent neural network in the form of a Long Short-Term Memory (LSTM) based network, combined with a fully connected output layer. This network is illustrated in Figure 3.6.

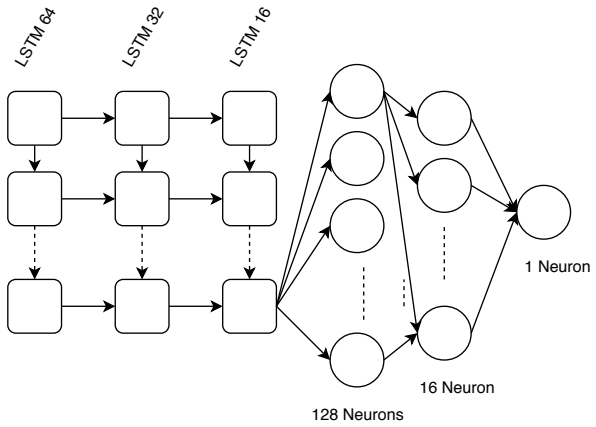


Figure 3.6: Long Short Term Memory network with 3 stacked layers of LSTM cells connected to a feed forward network for predictions.

3.2.3.3 Temporal convolutional network

Another way to infer time dependency is to utilize Temporal Convolutional Networks (TCN). Temporal convolutional networks convolve a filter over the time domain (as opposed to the spatial domain, as commonly done for images). We took inspiration from the fully convolutional network given in [31], and propose the architecture described in the Figure 3.7.

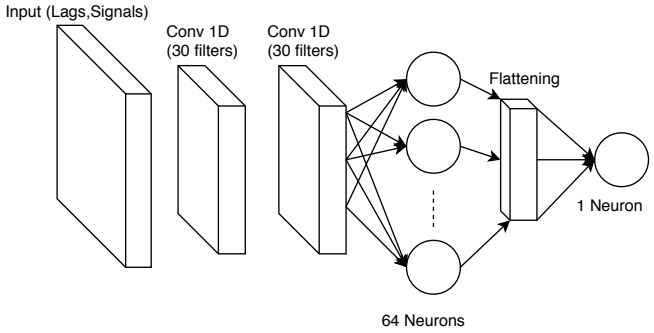


Figure 3.7: Temporal convolutional architecture with 2 convolutional layers (kernel size of 2) for feature extraction and a MLP for prediction.

3.2.3.4 Multi-channel deep convolutional neural networks

Multi-Channel Deep Convolutional Neural Networks (MC-DCNN), first proposed in [7], separates each multivariate time series into several univariate time series. Each channel of univariate time series has an independent CNN layer for feature extraction. The feature extraction layers are then flattened and concatenated and feed to a MLP. The structure can be seen in Figure 3.8.

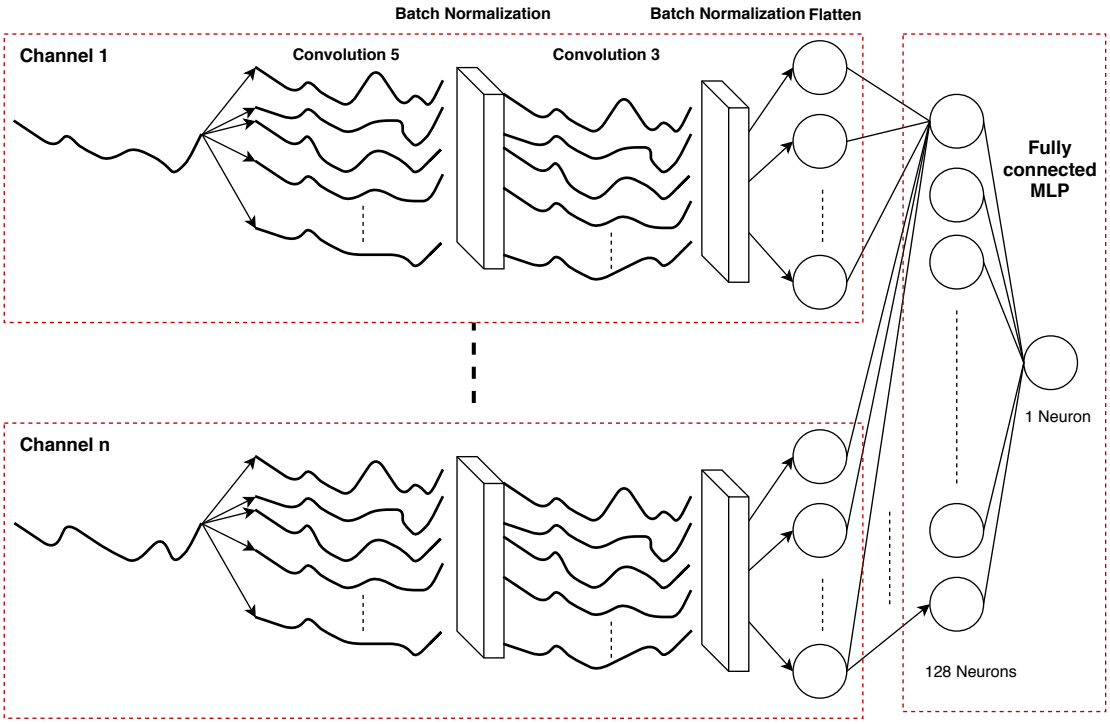


Figure 3.8: An overview of our proposed neural network. Convolution 5 and 3 refer to a kernel size of 5 and 3, respectively. The architecture and figure are inspired by the work of [7].

3.2.4 Longer prediction horizons

Once we identified the most suitable architecture, we increased the prediction horizons (see Section 3.2.2) and investigated how the recall and precision was affected. For training purposes, this simply meant running the data through the pipeline once more (see Section 3.1.1) with a different horizon. In order to preserve the proportions of the two target classes in the event window, however, this made the time scales incompatible for evaluation using the metrics described in Section 3.2.3.

As mentioned in Section 3.2, we introduced a more realistic evaluation method that resulted in the simulator described in Section 3.1.1. The simulator aims to mimic generic driving, by including all kinds of data (as opposed to only data from the LDE neighborhood) and by discarding segments of data that occur just after LDEs or situations where an LKA system would have intervened. The amount of data discarded after each such stoppage is T_{cooldown} seconds.

In the earlier performance evaluation we looked at the network output for each example. During simulation we used a looser definition of what represented a successful intervention by looking at each event instead. The confusion matrix content was derived using the following definitions:

1. We count a prediction as a true positive when it is made $T_{\text{horizon}}T_{\text{tolerance}}$ seconds prior to the event.
2. When a prediction is made and there are no events within $T_{\text{horizon}}T_{\text{tolerance}}$ seconds it is counted as a false positive.
3. If the network does not predict a lane departure $T_{\text{horizon}}T_{\text{tolerance}}$ prior to the event it is counted as a false negative.
4. When there is no event occurring and the threat assessment system does not predict an event, it is counted as a true negative.

For each of the true positives, the simulator measured the time difference between an ideally triggered intervention and when it actually triggered.

Even though the simulator was capable of simulating on all kinds of data, its primary purpose was to compare various prediction horizons without having trouble with the different time scales. It was therefore sufficient to use only $\mathcal{D}_{\text{test}}$ for this evaluation. As we shall see later, the same approach will be used for evaluating the complete threat assessment system.

It is paramount to note that the outputs (recall, precision, etc) produced by this type of evaluation cannot be directly compared with the types of metrics produced by the evaluation scheme used for finding a good network architecture, see Section 3.2.

3.3 Designing the threat assessment system

The previous sections we have described the different components of the threat assessment system, see Figure 3.1 for an overview. In summary, we have outlined:

- The dataset and how to preprocess it;
- The two evaluation schemes for evaluating networks. One for comparing architectures and one for comparing prediction horizons.
- How the network output is thresholded and used for decision making.

To assess the performance of the overall system we used the simulator in the same way that we used it to evaluate the networks for various prediction horizons, see Section 3.2.4. Instead of evaluating the system using only network input examples, all containing LDEs, we used more generic driving taken from the set $\mathcal{D}_{\text{generic}}$. This set included *all forms of driving* present in the dataset for which the LKA conditions were fulfilled.

4
Results and discussion

4.1 Dataset

The dataset contained over 4,000 hours of driving in total. In Section 3.1.3 we described the conditions for which our threat assessment system (TAS) was applicable. These conditions reduced the dataset to approximately 1,000 driving hours. Within this set, an extraction of 16,978 unintentional Lane Departure Events (LDEs) (see Section 3.1.4 and Section 3.2.2) were used for the training and evaluation stages. When addressing longer prediction horizons (Section 4.2.4), we used differently sized event windows which resulted in gradual reductions of the number of LDE cases down to sizes as low as 14,000.

Through experimentation, we determined that only a subset of the available signals contributed significantly to the network performance. These are the yaw rate, speed, acceleration and the two lower order coefficients of each lane marker estimate. See Section 3.1.2 for details.

4.2 Networks

To derive the best network model we evaluated the network on accuracy, recall and precision, as described in Section 3.2.3. We used a prediction horizon of $T_{\text{horizon}} = 0.5$ seconds and an output threshold of $\tau = 0.5$. In this section, we select the best network, present prediction examples from the dataset and provide a deeper analysis of difficult scenarios.

4.2.1 Different architectures

The different networks described in Section 3.2.3 have been trained and evaluated on our dataset, see Section 4.1. All networks were trained with 100 epochs, 10,000 steps per epoch, a batch size of 20 and with early stopping. The Multilayer Perceptrons (MLPs) were trained with several configurations: with 3 layers each containing 64, 128 or 256 neurons; with or without batch normalization; using sigmoid, Rectified Linear Unit (ReLU) or Exponential Linear Unit (ELU) as activation functions.

As seen in Table 4.1, the Long Short-Term Memory (LSTM) network, Temporal Convolutional Network (TCN) and Multi-Channel Deep Convolutional Neural Network (MC-DCNN) did not contribute to any performance increase. All networks were trained and evaluated using a sample window of size $N_{\text{lags}} = 10$ (see Section 3.2.2). Therefore, we conclude that a simple MLP architecture is the best choice, even if this choice may seem surprising considering the success that the more advanced architectures have had in, e.g., image classification or human activity recognition. One possible explanation is that our input space is of much lower dimension when compared to the input spaces the other works.

Table 4.1: Overview of different network results.

	MLP	**LSTM**	**TCN**	**MC-DCNN**
Accuracy	0.8948	0.8933	0.8941	0.8933
Recall	0.8028	0.7643	0.7924	0.7851
Precision	0.8071	0.8277	0.8113	0.8135

4.2.2 Selected network architecture

The network chosen according to Section 3.2.3 for the TAS had the following architecture: 3 layers with 64 neurons in each, using ELU as the activation function and connected to a single output neuron with sigmoid as activation function. The network was regularized with dropout of probability 0.3 between each layer, and we used a threshold $\tau = 0.5$ for the output. A summary of all predictions made for the test set $\mathcal{D}_{\text{test}}$ can be seen in the confusion matrix given in Table 4.2.

Table 4.2: The confusion matrix for the selected network evaluated on the test set which corresponds to the MLP in Table 4.1.

		Ground truth	
		LDE	**Non-LDE**
Prediction	**LDE**	27012	7248
	Non-LDE	5576	80722

Figure 4.1 shows a few selected scenarios taken from the test set $\mathcal{D}_{\text{test}}$ (see Section 3.2.1) and using the chosen network. The scenarios were chosen to illustrate the weaknesses and strengths of the network. The bottom panels show the network outputs and target outputs (Section 3.2.2), while the top panels depict the road for the scenarios under consideration. The decision threshold τ was set to 0.5, see Section 3.2.3.

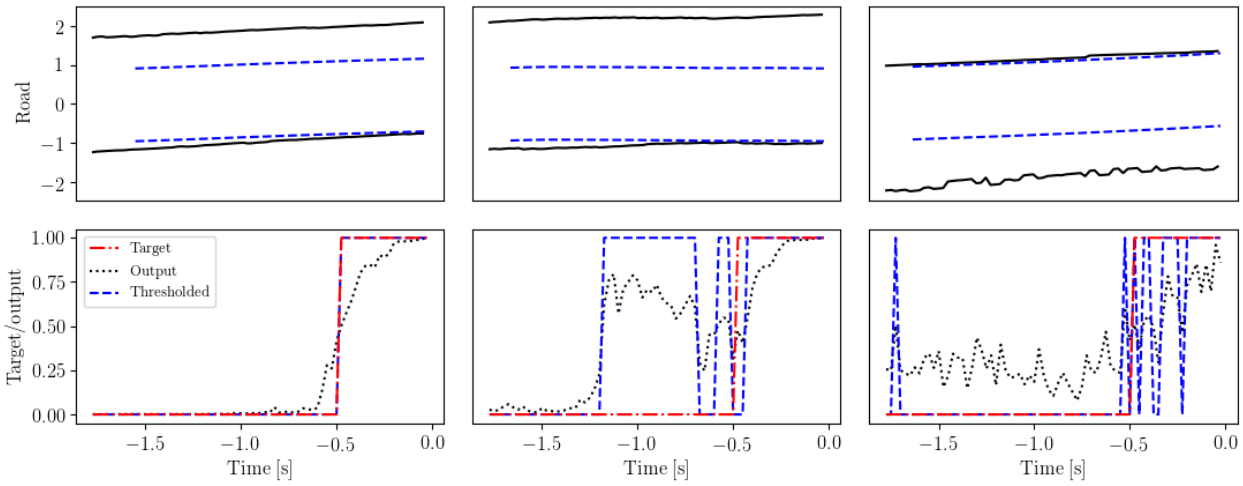


Figure 4.1: Top panels: the solid lines show the position of the lane markers, while the dashed lines show the front corners' positions of the vehicle, headed towards the right. Bottom panels: the dash dotted line shows the target value; the dotted line shows the output value and the dashed line shows the output when the output threshold is set to 0.5.

The leftmost panels show a case that the network performs well. The lane marker estimates are relatively smooth and the front corners of the vehicle (marked in blue) approach it at a relatively large yaw angle. This was typically observed in high performance scenarios.

The center panels show a case where the vehicle approaches the lane marker, but the characteristics of the road suddenly change so that our system fails to identify it as a lane crossing. One could argue, however, that this is a borderline case where an intervention would have made sense in practice but labeled as a false positive due to our hard definition of an LDE. This issue could have been avoided by defining a "degree" of lane departure, for example, which would lead to a softer boundary. Possible implementation of this idea include the "fuzzification of the lane markers/front corners", perhaps using Gaussians. A softer measure like that would also eliminate the obvious issues of estimating a step function.

The rightmost panels illustrate two issues. The first is that the rightmost lane marker has a quite erratic behavior, indicating a poor estimate. While acceptable if nothing else is available

(poor measurements are better than none), in this case the vehicle is about to exit on the left-hand side of the road and it is questionable whether the estimate of the right marker is relevant in determining this at all. The obvious solution for this is to use a threat assessment system that focuses on the closest lane marker only. It would solve the problem of removing irrelevant noise while simultaneously allowing for a smaller network architecture. Since the lane markers are only estimates of the real world, they are themselves stochastic processes and therefore prone to estimation errors, which may make it more difficult. The second issue is that the vehicle essentially drives parallel to the lane marker, i.e., with a small yaw angle. This issue was significant enough that we decided to analyze it further in the next section.

4.2.3 Error analysis

To determine possible sources of errors, we studied the characteristics of some of the signals described in Section 3.1.2 for each example. In particular, we studied the mean values of the yaw angle and vehicle speed as well as the maximum discontinuities in the lane marker estimates, and compared the network's output to the target output.

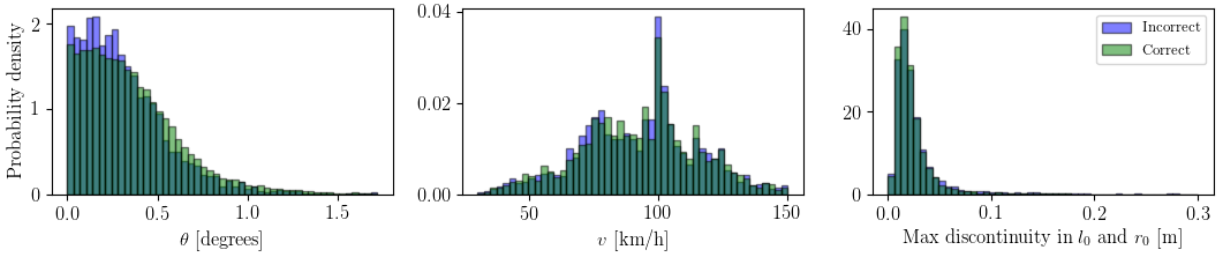


Figure 4.2: The distributions of the mean yaw angle, vehicle speed and zero order coefficients of the lane marker estimates for correct and incorrect network outputs. The means are calculated over the same sample windows. Note that the axis of the densities are scaled differently. The dark-green signal is where the incorrect and correct distribution coincides.

See now Figure 4.2. This figure indicates that cases with low yaw angles may be difficult to correctly identify. To verify this, all cases where the yaw angle $\theta \leq 0.33°$ were removed from the test set $\mathcal{D}_{\text{test}}$. With this adjustment, the network achieved a significant performance increase, accuracy increased from *0.8948* to *0.9343*, recall from *0.8028* to *0.8484* and precision from *0.8071* to *0.9005*. The confusion matrix is reported in Table 4.3.

In the other two cases, speed and lane marker discontinuities reveal no obvious insights. In fact, discontinuities above 0.1 m occur very seldom and therefore any contribution to the error should be minimal. Partially, this can be attributed to the removal of cases where the discontinuity exceeded 0.3 m, see Section 3.1.3.

Table 4.3: Confusion matrix for predictions made without low yaw angles towards the lane marking.

		Ground truth	
		LDE	**Non-LDE**
Prediction	**LDE**	11642	1287
	Non-LDE	2080	36253

4.2.4 Longer prediction horizons

We also tested the selected network for longer prediction horizons using the methodology described in Section 3.3. In short, to make a just evaluation for different horizons we now consider true positives to be when an intervention is made within $T_{\text{tolerance}} = 0.25$ seconds from the target. The interval was chosen to minimize difficulties of learning a step function.

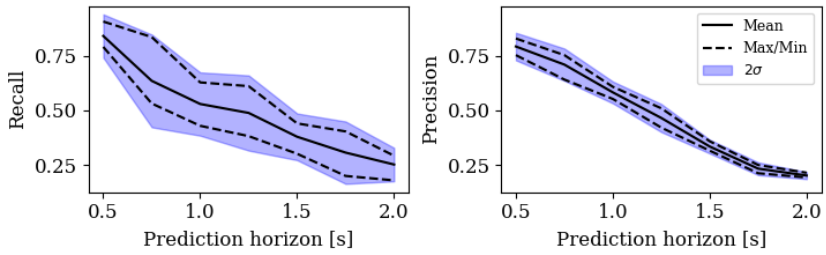


Figure 4.3: The final network model evaluated for for different prediction horizons.

As expected, the performance drops for longer horizons. However, since we have not made a thorough investigation non-MLP architectures for larger horizons, we cannot exclude the possibility that other network architectures could have been more beneficial in these cases, such as those proposed in Section 4.2.

It is also possible that the way we formatted examples (see Section 3.2.2) has not been explored thoroughly enough. For example, we look at fixed number of consecutive time samples, but it is possible that other ways of selecting data from the signals would have been more advantageous. For example, it might have been more interesting to construct examples with fewer samples but with some spacing in between in order to remove possibly redundant inputs. In a similar sense, we could have constructed examples incorporating several time scales at once: one capturing short term behaviors and another capturing long term behaviors. Similar fusions of different time scales have been used successfully for predicting crowd flows, for instance, as in [32].

4.3 Evaluating the threat assessment system

To evaluate the threat assessment system we ran a simulation on generic driving, as described in Section 3.3. Table 4.4 shows the resulting confusion matrix, containing both unintentional and intentional events, as these could not be distinguished from non-LDE cases.

Table 4.5 summarizes the results by using the metrics defined in Section 2.5.2.As seen in Table 4.4, the false positive rate is low which means that the driver may only be exposed to a few bothering false alarms.

Table 4.4: Confusion matrix for predictions made using the threat assessment system on the dataset $\mathcal{D}_{\text{generic}}$ (see Section 3.2.1). Note that this matrix cannot be compared with the confusion matrices in Section 4.2, since the evaluation methodology is different. See Section 3.3.

		Ground truth	
		LDE	**Non-LDE**
Prediction	**LDE**	2647	1117
	Non-LDE	394	8939168

Table 4.5: A summary of the simulation results presented in Table 4.4

Recall	**Precision**	**False positive rate**
0.8704	0.7032	0.000125

The false positive rate, as described in Section 2.5.2, indicates how often the driver is exposed to unnecessary interventions, i.e., how ”annoying” the system is. Our result of 0.000125 indicates that the system, on average, makes an intervention every 200 seconds.

Finally, Figure 4.4 shows the distribution of intervention triggering times for correctly predicted unintentional LDEs. See Section 3.3 for details.

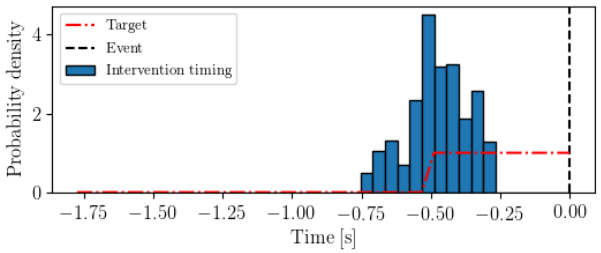


Figure 4.4: Distribution of intervention timings for true positives. The dash dotted line shows the target and the black line shows the event timing. The bars show the probability density of intervention timings, i.e., the discrepancy between the actual and targeted times of intervention.

Note that the dataset used to obtain these results include all forms of driving. As is expected, a dataset generated to mimic generic driving contains far more non-LDEs than actual LDEs. Therefore, a network trained on a dataset containing a relatively balanced number of LDEs and non-LDEs is likely to produce a greater number of false positives. A solution to this could be to select difficult cases and continue training the network on these, specifically. This dataset also includes LDEs that we defined as intentional, see Section 3.1.4. Since the network was not designed to handle such cases, intentional and unintentional events may look indistinguishable from the network point of view, which is likely to affect these results. Therefore, it is necessary that a complementary system is used to separate the two types of events. A simple solution may be to deactivate the LKA while the driver is signaling a lane change, but more sophisticated solutions such as using a camera monitoring the awareness of the driver can also be considered.

4.3.1 Further work

Based on the results presented in this Chapter, different research avenues for future developments seem pertinent.

For instance, a different model architecture that we did not have time to try is an autoencoder model such as the one in [33], which could potentially extract important features from the historic signals by concatenating it with the current state of the signals.

Future work should also include a correct distinction of intentional from unintentional events. An in-vehicle camera to monitor the driver awareness could be used, for instance, or a human-based annotating system to properly identify the different cases.

Finally, it could be pertinent to have different networks identifying lane departures for the two different road sides for better performance. By separating such events, the network would have a simpler case to predict, even if a drawback can be reduction of training data. Another solution might be to use a time series regression instead of a classificator. While one can utilize bigger parts of the dataset, it is however commonly known that it is hard to identify a regression model for prediction purposes using a dataset where relevant elements are rare, which may be the case for lane departures.

5
Conclusion

In this work we proposed a threat assessment system for lane keeping assistance (LKA) using artificial neural networks. Unlike many works in the literature using simulated data, we developed and validated our algorithms with realistic data which represents a substantial advancement towards data driven LKA systems under realistic conditions. We compared several state-of-the-art network architectures and investigated performance for longer prediction horizons. We have demonstrated that a data driven approach is feasible for predicting unintentional lane departures at least 0.5 seconds into the future. We also identified several areas where the data driven approach perform poorly, which can be used to identify future research efforts in this field.

Bibliography

[1] National Highway Traffic Safety Administration, "Quick facts 2016," 2017.

[2] ——, "2016 fatal motor vehicle crashes: Overview," 2017.

[3] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *ROBOMECH Journal*, vol. 1, no. 1, pp. 1–14, 2014.

[4] J. M. Ambarak, H. Ying, F. Syed, and D. Filev, "A neural network for predicting unintentional lane departures," in *2017 IEEE International Conference on Industrial Technology (ICIT)*, 2017, pp. 492–497.

[5] J. Yang, M. N. Nguyen, P. P. San, X. Li, and S. Krishnaswamy, "Deep convolutional neural networks on multichannel time series for human activity recognition." in *IJCAI*, 2015, pp. 3995–4001.

[6] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *arXiv preprint arXiv:1708.02709*, 2017.

[7] Y. Zheng, Q. Liu, E. Chen, Y. Ge, and J. L. Zhao, "Time series classification using multi-channels deep convolutional neural networks," in *International Conference on Web-Age Information Management*. Springer, 2014, pp. 298–310.

[8] L. Squire, F. Bloom, N. Spitzer, L. Squire, D. Berg, S. du Lac, and A. Ghosh, *Fundamental Neuroscience*, ser. Fundamental Neuroscience Series. Elsevier Science, 2008.

[9] 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.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[11] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 9–50.

[12] K. Murphy, *Machine Learning: A Probabilistic Perspective*, ser. Adaptive computation and machine learning. MIT Press, 2012.

[13] P. L. Bartlett, M. I. Jordan, and J. D. McAuliffe, "Convexity, classification, and risk bounds," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156, 2006.

[14] Y. Lecun, *A theoretical framework for back-propagation*. IEEE Computer Society Press, 1992.

[15] G. Chen, "A gentle tutorial of recurrent neural network with error backpropagation," *arXiv preprint arXiv:1610.02583*, 2016.

[16] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Trans. Neur. Netw.*, vol. 5, no. 2, pp. 157–166, 1994.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[19] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[20] K. Burnham and D. Anderson, *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*. Springer New York, 2003.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[22] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," vol. 9, 2010, pp. 249–256.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*. USA: Omnipress, 2010, pp. 807–814.
[24] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *ArXiv e-prints*, 2015.
[25] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.
[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
[27] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *ArXiv e-prints*, 2015.
[29] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.
[30] F. Chollet *et al.*, "Keras," 2015.
[31] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 1578–1585.
[32] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction." 2017.
[33] N. Laptev, J. Yosinski, L. E. Li, and S. Smyl, "Time-series extreme event forecasting with neural networks at uber," in *International Conference on Machine Learning*, 2017.