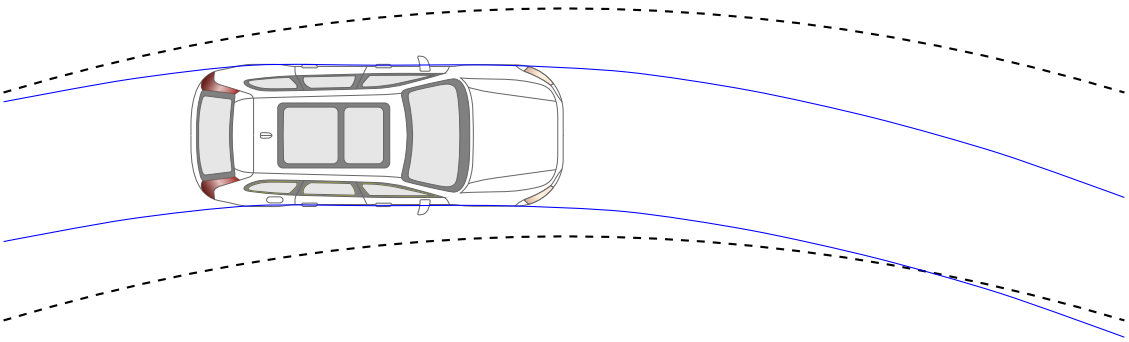


A Lane Departure Detection System Based on Uncertainty Aware Machine Learning

Master's thesis in Complex Adaptive Systems

JESPER LARSSON
MATTIAS SJÖSTEDT

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

A Lane Departure Detection System Based on Uncertainty Aware Machine Learning

JESPER LARSSON
MATTIAS SJÖSTEDT

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

A Lane Departure Detection System Based on Uncertainty Aware Machine Learning
JESPER LARSSON
MATTIAS SJÖSTEDT

Supervisor: John Dahl, Zenuity AB
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2020
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

A Lane Departure Detection System Based on Uncertainty Aware Machine Learning

JESPER LARSSON
MATTIAS SJÖSTEDT
Department of Electrical Engineering
Chalmers University of Technology

Abstract

In the effort to reduce traffic-related accidents and fatalities, the use of autonomous active safety systems has become increasingly important. A key component in autonomous active safety systems is how to accurately predict potential future threats. In this thesis, we use uncertainty aware machine learning models for lane departure detection by performing time series regression, on real-world automotive data, to predict the future lateral relative distance to lane markers. We propose a decision making algorithm based on the estimated probability of departure and evaluate its performance in terms of the true-positive-rate and false-positive-rate. The performance is compared to baseline machine learning models, as well as a kinematic model that is common in the automotive industry. The results indicate that the uncertainty aware model, despite delivering well-calibrated uncertainty estimates, does not improve the decision making performance, when compared to the baseline machine learning model. Moreover, we show that the ensemble-based models can be distilled to recover a computationally cheaper model while still retaining most of the predictive performance of the full ensemble.

Keywords: Collision avoidance, Lane departure detection, Gaussian output networks, Ensembles

Acknowledgements

First and foremost we would like to thank our Zenuity supervisor John Dahl for his great support during the thesis and for teaching us to always question what may seem obvious. We would also like to thank our academic supervisor Professor Lennart Svensson for his invaluable input and for keeping us on track throughout the thesis. Finally, we would also like to thank Gabriel Rodrigues de Campos for helping us improve the quality of our report.

Jesper Larsson & Mattias Sjöstedt, Gothenburg, June 2020

Contents

List of Figures

List of Tables

Nomenclature

ANN Artificial Neural Network

BLR Bayesian Linear Regression

CAS Collision Avoidance System

CVM Constant Velocity Model

DM Decision Making

FLR Frequentist Linear Regression

GMLP Gaussian Multilayer Perceptron

GMLPE Gaussian Multilayer Perceptron Ensemble

KL Kullback-Leibler

MCD Monte Carlo Dropout

ML Machine Learning

MLP Multilayer Perceptron

MLPE Multilayer Perceptron Ensemble

SBTM Single-Behaviour Threat Metrics

TA Threat Assessment

TLC Time-to-Lane Crossing

1 Introduction

Recently, autonomous active safety systems have become an important cornerstone in the pursuit of reduced traffic-related fatalities. New sensor technology and cheaper computational power have accelerated the development of highly effective and advanced collision avoidance systems. By utilizing collision avoidance systems, fatality-prone accidents, due to driver mistakes or inattentiveness, could be prevented. One of the major challenges in the development of an active safety system is how to accurately predict future threats so to assist the driver correctly.

Essential to automated safety, and especially for Collision Avoidance Systems (CAS), are real-time, highly robust Threat Assessment (TA) methods. Discerning the correct threat level using a classical engineering approach is typically hard, especially in a possibly high dimensional input space provided by the extensive sensor suite of modern cars. Moreover, due to individual driving skill and style, how to precisely distinguishing safe or unsafe driving behaviours remains non-trivial.

Furthermore, the system performance is directly dependent on hardware and software limitations as well as the driver's behavior. These factors combined give rise to predictive uncertainties. Without any information on the predictive uncertainty, an autonomous intervention can only be based on the threat level predicted by the TA model. However, if the predictive uncertainty can be estimated, this information could be included in the decision making process of whether the predicted threat level warrants an autonomous intervention. By including the predictive uncertainty in the decision making process, it may be possible to increase the performance of the collision avoidance system.

In order to estimate the future threat level an accurate prediction model is required. A possible prediction model for this purpose is a Machine Learning (ML) based model, that approximates a function relating the input state to an output state. These relations are purely inferred from data in the form of a training set. An uncertainty aware ML approach however also relates a confidence measure to this output state. With the recent rise and success of deep machine learning techniques in numerous fields, it is of interest to evaluate whether an uncertainty aware ML approach could be used as a basis for a TA system. By using an uncertainty aware ML model, it may be possible to capture the complex dynamics and approximate the threat level using a sufficiently large, excited and annotated dataset for this specific purpose.

1.1 Collision Avoidance

A collision avoidance system aims to avoid or mitigate collisions. These types of systems are commonly referred to as Advanced Driver-Assistance Systems (ADAS) and they all require some sort of TA in order to properly function. A simple example of a CAS today is blind spot detection, that monitors threats in the driver's blind

spot. This system reduces the collision risk by simply informing the driver whether a lane change is unsafe at the moment.

Adaptive cruise control is another example of a CAS. Such a system keeps a safe distance to the vehicle in front by adjusting the set speed as well as braking when necessary. This is done by using sensors to monitor the car in front of the ego vehicle. Compared to blind spot monitoring, such a system is more powerful as it actually actuates the car when engaged. Similar systems, such as Lane-Keeping Assistance (LKA) and Lane-Centering (LC) systems actuate by steering to avoid lane departures.

1.2 Threat Assessment

The purpose of TA is to predict how a given scenario will evolve over time. Despite the multitude of existing approaches and methods, TA systems can be divided into model-based and data-driven methods

There exist a large variety of different model-based threat assessment systems. According to [1] the model based methods can be divided into: Single-Behaviour Threat Metrics (SBTM), optimization methods, formal methods and probabilistic methods. Examples of such methods are presented in [2]–[6]. Amongst these methods, this thesis is limited to SBTM methods for performance comparisons.

In general a SBTM predicts the future threat based on a specific assumptions on driver actions, i.e. keeping constant velocity or braking with constant acceleration. An important metric in this matter is Time to Collision (TTC) [7]. Given two moving objects, TTC is calculated as their relative distance divided by their relative velocity. Similarly, in the spatial domain, a constant lateral velocity model can be used to predict the distance to the lane marker [8].

Data-driven methods, such as deep neural networks are very versatile. Recent development has resulted in models with impressive performance in complex tasks such as image classification [9], object detection [10] and natural language processing [11]. Deep neural networks have also shown promising results when applied to TA systems [12].

Despite being powerful and versatile, modern deep neural networks do not necessarily output well-calibrated uncertainty estimates [13]. Here, the term *well-calibrated* refers to that the predicted uncertainty complies with the true uncertainty. Reliable uncertainty information is beneficial in a safety critical application in order to correctly assess the situation. By including a calibrated uncertainty measure in the Decision Making (DM), it might be possible to increase the performance of the CAS.

1.3 Project Description

This thesis aims to develop an uncertainty aware TA system capable of detecting unintended lane departures or run-off-road events before they occur. The system is

based on a data-driven model that predicts the lateral position of the ego vehicle, and the associated predictive uncertainty, at a future time horizon H.

A prediction model, based on an Artificial Neural Network (ANN), is trained for time series regression on real-world automotive data to predict the lateral distance to the lane marker as well as the associated predictive uncertainty.

The prediction model is then used as a basis for threat assessment to detect lane departures. The predicted lateral position as well as the associated predictive uncertainty is merged into a single threat level metric, used to make decisions of whether an autonomous intervention is needed.

Moreover, in real applications computational power is limited in an on-board computer, and the resources must be shared by numerous processes. Therefore, an attempt to reduce the computational complexity of the tested models using ensemble distillation is evaluated.

In essence this project can be divided into three distinct parts: i) preparation of data used for training and evaluation; ii) implementing uncertainty aware ML models, and iii) finally evaluating the implemented models.

1.4 Related Work

In recent years, a lot of progress has been made in uncertainty estimation of data-driven methods. Various successful approaches exist and are here summarized.

Lakshminarayanan *et al.* [14] suggest training an ensemble of identical ANNs , where each ensemble member outputs the parameters of a Gaussian distribution, namely the mean and the variance. The individual networks are each trained on the full dataset, but will learn slightly different things due to random initialization. These models are able to learn the variance from the data by maximizing the likelihood of the Gaussian output parameters using the negative log likelihood loss function. Therefore, no explicit target has to be provided for the uncertainty estimate. This is the main uncertainty aware ML approach used throughout the thesis.

Bui *et al.* [15] proposes using Deep Gaussian Processes (DGP) for regression. A DGP is a layered Gaussian process (GP) that resembles ANNs with infinitely wide hidden layers [16]. When using multiple layers, it is intractable to perform exact Bayesian inference. Instead, the authors use stochastic expectation propagation as a Bayesian approximation [17].

Dropout during training is a common method to prevent over-fitting the training data. However, it may also be used during test time to construct an ensemble based on a single network. This is called Monte Carlo Dropout (MCD) and according to Gal and Ghahramani [18] the technique approximates Bayesian posterior inference. The predictive distribution is obtained by performing a number of forward passes through the network while sampling different dropout realisations.

Blundell *et al.* [19] describe a possible approach of Bayesian Neural Networks (BNNs) that uses weights represented by probability distributions instead of weights repre-

sented by a single value. In this case, every weight is defined by its mean and variance, resulting in an increase in the number of model parameters by a factor of two. However, getting the exact Bayesian posterior for the large number of weights of a neural network is intractable and a variational approximation is utilized instead. The authors suggest using a cost function based on minimising the Kullback Leibler divergence between the weight distributions and the Bayesian posterior on the weights. Here, the unknown Bayesian posterior is expressed in terms of the known prior on the weights and likelihood of the data using the Bayes rule. This cost function regularizes the weights by penalizing too low variances. Such a network can, using this cost function, be trained using backpropagation and the method has therefore been named *Bayes by backprop*. However, training this network is more expensive than a standard network with the same architecture as the probability distributions on the weights require more parameters. Much like MCD, the model output is obtained by sampling multiple realisations of the same network in order to leverage the weight distributions and get an uncertainty estimate.

Pearce *et al.* [20] present a Bayesian approximation method applicable in classification and regression. They use a method called Randomized Maxmimum a Posteriori Sampling (RMAPS)[1] and show that this can be used to calculate the true Bayesian posterior given that the parameter likelihood is known in advance. Knowing this is generally not possible, and an approximate method for obtaining the posterior, denoted by the authors as Anchored Ensembling, is presented. Unlike Bayes by Backprop [19], the weights are not represented by probability distributions. For this method, the predictive output is represented by sample mean, sample variance and a data noise estimate.

Moberg *et al.* [21] present an ANN trained to output mean and variance representing a Gaussian distribution. The predicted Gaussian variance as well as the activations of the last hidden layer, are then used to perform Bayesian Linear Regression (BLR). Having access both to the predicted Gaussian variance and BLR gives an estimate of both the epistemic and aleatoric uncertainty. While already giving good results in terms of predictive Negative Log-Likelihood (NLL), even better results can be achieved by using a BLR ensemble.

1.5 Contributions

The main contribution of this thesis is the creation and evaluation of an uncertainty aware data-driven TA system, and can be divided as follows:

- Implementation of state of the art uncertainty aware ML models on real-world automotive data for time series regression
- Formulation of an uncertainty aware lane departure detection criterion
- Investigation of the properties and performance of the proposed detection criterion

[1]Note that the authors refer to this as RMS, but due to the overlap with the more common interpretation, Root Mean Square, we refer to it as RMAPS.

- Comparison of the performance of the proposed TA method with respect to baseline ML models as well as a kinematic model, commonly used in the automotive industry

1.6 Outline

The remainder of this thesis follows the theory, method, result and conclusion format. The relevant background theory for regression, ANN, uncertainty estimation and theory relevant for the implemented models is presented in Chapter 2. In Chapter 3 pre-processing, annotation and usage of the dataset is presented, along with model implementation details and model training setup. Furthermore, the model evaluation procedure concludes the method chapter. The evaluation results are presented in Chapter 4, starting with regression performance. This is followed by evaluating the predictive uncertainty estimates. Performance in a TA setting then concludes the result chapter. Finally, a discussion and conclusions for regression performance and TA evaluation are presented in Chapter 5.

2 Theory

The methods used throughout the thesis are tailored for an uncertainty aware regression task. This chapter presents fundamentals for regression as well as theory regarding ML methods that are capable of estimating uncertainty. Different kinds of uncertainty and how the uncertainty may be calibrated is also described.

2.1 Regression

A simple regression task aims to infer the relationship between some input data $\mathbf{x}$ and a target $\mathbf{y}$. This may be expressed as

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

where $\hat{\mathbf{y}}$ is the model estimate of $\mathbf{y}$. For the simplest one dimensional linear case, the data may be visualised on a coordinate system where x represents the input data and y the target value. Assuming that there is a linear relationship between these, a line

$$\hat{y} = w_0 + w_1 x \tag{2.2}$$

that fits the available data would be calculated. An example of such a regression model fit to data of some sort is presented in Fig. 2.1. The linear model seems suitable in the example, but relationships exist where more complex models are required to get an acceptable mapping. For such cases, more weights can be used in order to represent a higher order polynomial such that

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + ... + w_n x^n = f(x,\mathbf{W}). \tag{2.3}$$

A polynomial of higher order could be better at fitting the data, although it may also suffer from over-fitting the data points instead of generalizing. This may result in a model that is good for the training data, but that deviates much from the true value when exposed to new unseen data.

2.1.1 Frequentist Linear Regression

Frequentist Linear Regression (FLR) is a common way to perform regression [16]. It is based on the concept presented above and may be optimized using different error metrics such as the absolute or the squared error. The squared error case is presented here and is also known as the least squares approach. The least squares error is defined as the squared distance from the predicted point $\hat{y}$ to the target point y and is given as

$$\mathcal{L}_i(f(\mathbf{x_i},\mathbf{W}),y_i) = (y_i - \hat{y}_i)^2, \tag{2.4}$$

for the data point i in a given dataset. A function used for optimizing a model like this is also referred to as a loss or cost function in the literature. Optimal parameters

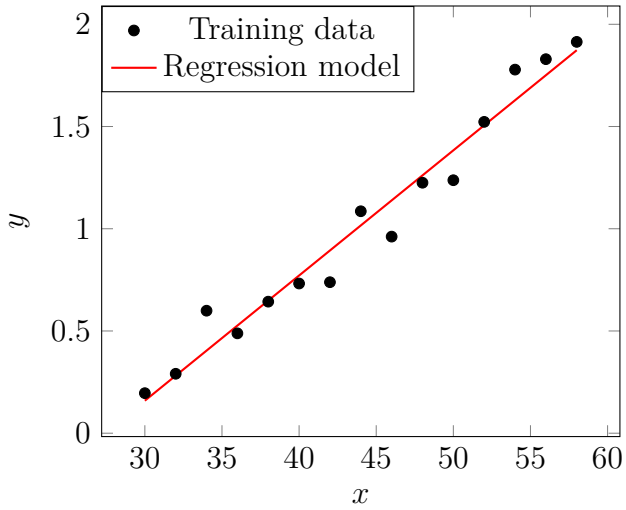


Figure 2.1: Regression example. A linear model fits the available training data.

for the entire dataset minimize the average least squares, also known as the Mean Squared Error loss function

$$\mathcal{L}(f(\mathbf{X},\mathbf{W}),\mathbf{y}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{2.5}$$

This function aims to find the optimal parameters minimizing the loss such that

$$\underset{\mathbf{W}}{\operatorname{argmin}}(f(\mathbf{X},\mathbf{W}),\mathbf{y}), \tag{2.6}$$

which has a closed form solution for $\mathbf{W}$. The solution is obtained by differentiating Eq. (2.5) with respect to the model parameters $\mathbf{W}$ for the available training data. The partial derivatives are then equated to zero for optimal parameters. It can be shown that this indeed corresponds to a minimum of Eq. (2.5).

The simple linear case corresponds to Eq. (2.2) with the weights w_0 and w_1, with the closed form solutions

$$\hat{w_1} = \frac{\overline{xy} - \bar{x}\bar{y}}{\overline{x^2} - \bar{x}^2} \tag{2.7}$$

$$\hat{w_0} = \bar{y} - w_1\bar{x}. \tag{2.8}$$

Here, the bar notation indicate the dataset average. For example, the different bar notation should be interpreted as

$$\overline{xy} = \frac{1}{N}\sum_{i=1}^{N} xy \tag{2.9}$$

$$\bar{x}\bar{y} = \frac{1}{N}\sum_{i=1}^{N} x \frac{1}{N}\sum_{i=1}^{N} y. \tag{2.10}$$

The same method can easily be extended for polynomials of a general degree n in matrix form. Each row in the input matrix $\mathbf{X}$ then represents one data point as in

Eq. (2.3)

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & ... & x_1^n \\ 1 & x_2 & x_2^2 & ... & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & ... & x_N^n \end{pmatrix}. \tag{2.11}$$

The closed form matrix solution can then be written as

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{2.12}$$

Note that $\mathbf{y}$ represents a column vector of all training targets. With the model in place, it may be used for predictions as

$$\hat{y} = \mathbf{W}^T\mathbf{x}, \tag{2.13}$$

where $\mathbf{x} = [1, x, x^2, ...x^n]^T$.

2.1.2 Bayesian Linear Regression

Another popular approach to linear regression is Bayesian Linear Regression (BLR). This approach differs from FLR in the sense that the model consists of probability distributions instead of a deterministic weight vector. The output of such a model is therefore also represented by a distribution known as the Bayesian predictive posterior. Another important difference is that any prior knowledge of the problem may be included in the model to help it perform better, especially when the available data is limited.

BLR has its core in the Bayes rule

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{2.14}$$

which gives a relationship between the conditional probabilities of two events. The left hand side denotes the probability of an event A occurring given that another event B occurs. This is equal to the probability of B given A times the probability of A divided by the probability of B.

In Bayesian linear regression, Eq. (2.14) is rewritten as

$$p(\mathbf{W}|\mathbf{y},\mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X},\mathbf{W})p(\mathbf{W})}{p(\mathbf{y}|\mathbf{X})}. \tag{2.15}$$

and consists of three key parts. These are the Bayesian prior distribution on the weights $p(\mathbf{W})$, likelihood of the data $p(\mathbf{y}|\mathbf{X},\mathbf{W})$ and posterior distribution $p(\mathbf{W}|\mathbf{y},\mathbf{X})$. The prior is a belief of how the data is distributed without having seen it. The likelihood indicates the probability of observing the targets given the inputs and model parameters. Finally, the posterior represents the weight distribution given the available training data and is used for making predictions. The denominator $p(\mathbf{y}|\mathbf{X})$,

represents the probability of the targets given the input data and is model independent. This quantity is a normalization constant, and is in general difficult to calculate [16].

Finding the posterior is the goal of Bayesian linear regression. The posterior does not have an analytical solution in general, but special cases exist. The likelihood and prior can form pairs said to be conjugate [16]. If this is the case, the posterior will have the same form as the prior and can be calculated analytically. For example, using a Gaussian prior and likelihood makes the posterior Gaussian as well. The denominator of Eq. (2.15) is in this case not required for finding the exact posterior. The proportionality of

$$p(\mathbf{W}|\mathbf{y},\mathbf{X}) \propto p(\mathbf{y}|\mathbf{X},\mathbf{W})p(\mathbf{W}), \tag{2.16}$$

is enough for finding the parameters of the Gaussian [16]. The procedure for finding the parameters is presented below with an assumption that the data noise, $\mathcal{N}(\varepsilon; 0,\sigma^2)$ is known. This assumption is not always true and σ^2 may be treated as a random variable itself, but assuming it is known slightly simplifies the procedure. This yields

$$p(\mathbf{W}|\mathbf{y},\mathbf{X},\sigma^2) \propto p(\mathbf{y}|\mathbf{X},\mathbf{W},\sigma^2)p(\mathbf{W}|\boldsymbol{\mu_0},\boldsymbol{\Sigma_0}), \tag{2.17}$$

where $\mu_\mathbf{0}$ and $\mathbf{\Sigma_0}$ denote the choice of parameters for the Gaussian prior.

The posterior $p(\mathbf{W}|\mathbf{y},\mathbf{X},\sigma^2)$ only depends on its mean and covariance parameters which here will be denoted $\boldsymbol{\mu_W}$ and $\mathbf{\Sigma_W}$. The posterior is Gaussian by conjugacy, and its parameters can be extracted from the product obtained by multiplying Gaussian prior and likelihood in Eq. (2.17)

$$\begin{aligned} p(\mathbf{W}|\mathbf{y},\mathbf{X},\sigma^2) &\propto \frac{1}{(2\pi)^{N/2}|\sigma^2\mathbf{I}|^{1/2}} \exp\Big(-\frac{1}{2}(\mathbf{y}-\mathbf{X}\mathbf{W})^T(\sigma^2\mathbf{I})^{-1}(\mathbf{y}-\mathbf{X}\mathbf{W})\Big) \\ &\times \frac{1}{(2\pi)^{N/2}|\sigma^2\mathbf{\Sigma}_0|^{1/2}} \exp\Big(-\frac{1}{2}(\mathbf{W}-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}^{-1}(\mathbf{W}-\boldsymbol{\mu_0})\Big) \\ &\propto \exp\left\{-\frac{1}{2}\left(\frac{1}{\sigma^2}(\mathbf{t}-\mathbf{X}\mathbf{W})^T(\mathbf{t}-\mathbf{X}\mathbf{W}) + (\mathbf{W}-\boldsymbol{\mu}_0)^T\mathbf{\Sigma}_0^{-1}(\mathbf{W}-\boldsymbol{\mu}_0)\right)\right\}. \end{aligned}$$

As mentioned, the posterior is Gaussian by conjugacy. Therefore, constants that don't include $\mathbf{W}$ may be omitted, which results in

$$p(\mathbf{W}|\mathbf{y},\mathbf{X},\sigma^2) = \mathcal{N}(\boldsymbol{\mu_W},\mathbf{\Sigma_W}) \propto \exp\left\{-\frac{1}{2}(\mathbf{W}^T\mathbf{\Sigma}_\mathbf{W}^{-1}\mathbf{W} - 2\boldsymbol{\mu}_\mathbf{W}^T\mathbf{\Sigma}_\mathbf{W}^{-1}\mathbf{W})\right\}. \tag{2.18}$$

Rewriting this further gives expressions for the posterior parameters

$$\begin{aligned} \mathbf{\Sigma_W} &= \left(\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X} + \mathbf{\Sigma}_0^{-1}\right)^{-1} \\ \boldsymbol{\mu_W} &= \mathbf{\Sigma_W}(\frac{1}{\sigma^2}\mathbf{X}^T\mathbf{y} + \mathbf{\Sigma}_0^{-1}\boldsymbol{\mu}_0). \end{aligned}$$

This motivates why the constant parameters may be omitted. Note also how the choice of prior parameters affect the posterior.

Table 2.1: Examples of conjugate prior pairs. More examples can be found in [16].

Prior	Likelihood	Posterior
Gaussian	Gaussian	Gaussian
Gamma	Gaussian	Gamma
Beta	Binomial	Beta

Table 2.1 summarizes some additional conjugate prior pairs that gives an analytical posterior following the same procedure as above. For the cases where an analytical solution cannot be found, it is possible to draw samples from the the known prior and likelihood to get an approximate posterior. This is a common approach, and is utilized in other Bayesian methods such as Bayes by Backprop and MCD. These methods are introduced in Section 2.5.

2.2 Artificial Neural Networks

In this section some of the underlying theory for Artificial Neural Networks is presented. ANNs are common machine learning models to use, and lots of variations exist for different applications. This section is limited to the function and training of a basic ANN. For a more thorough introduction see [22].

2.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are general mathematical function approximators that can be used for tasks like regression and classification. In short, an ANN is trained to map a set of inputs $\mathbf{x}$ to a desired target $\mathbf{y}$ using a dataset of known mappings to infer the parameters $\mathbf{W}$ of the model such that:

$$\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{W}), \tag{2.19}$$

where $\hat{\mathbf{y}}$ indicates a model prediction. The ANN is trained to generalize from the training data and can then be used to make predictions for new, previously unseen data.

An ANN consists of layers of neurons (nodes) connected by weight links (edges) as illustrated in Fig. 2.2. A vanilla feed forward ANN is fully connected, in the sense that every neuron has an individual link to each neuron in the next layer. This kind of vanilla ANN is also referred to as a Multilayer Perceptron (MLP). A MLP is schematically propagated from left to right, with inputs to the left and outputs to the right.

More formally, given an input $\mathbf{x} \in \mathbb{R}^n$ the output or activation a of a single neuron is given by

$$a(\mathbf{x}) = g(z) = g(\mathbf{w}^T\mathbf{x} + b), \tag{2.20}$$

where $\mathbf{w} \in \mathbb{R}^n$ are weights specific for the neuron, g is a differentiable *activation function* and b is a bias term. By including the bias term it is possible to shift the

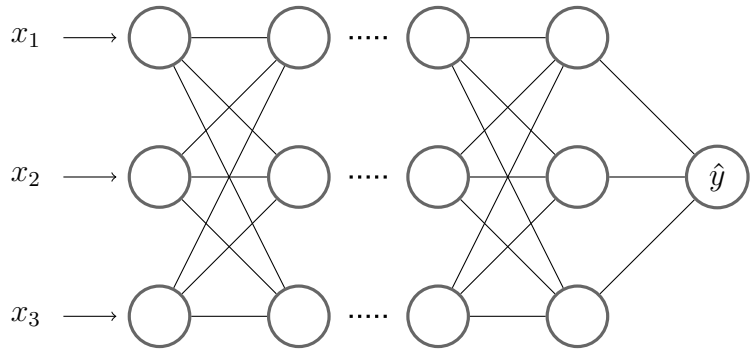


Figure 2.2: A schematic representation of a neural network with three input neurons and one output neuron.

activation function, which is not possible using only the term $\mathbf{w}^T\mathbf{x}$. The bias b can be considered as an additional weight associated to the neuron and can be rewritten as such e.g. $\tilde{\mathbf{w}} = [\mathbf{w}^T, b]^T$ and $\tilde{\mathbf{x}} = [\mathbf{x}^T, 1]^T$.

Single neurons can be grouped into a layer l containing m neurons and a MLP can then be constructed by assembling L layers, with possibly different numbers of neurons in each layer. The output $\mathbf{a}^l \in \mathbb{R}_l^m$ of layer $l > 1$ can then be computed using the output of the preceding layer $l - 1$ as

$$\mathbf{a}^l = g(\mathbf{z}^l) \tag{2.21}$$

$$\mathbf{z}^l = \mathbf{W}^l\mathbf{a}^{l-1} + \mathbf{b}^l, \tag{2.22}$$

where $\mathbf{W}^l = w_{ij}^l$ connects the j:th neuron in layer $l - 1$ to the i:th in layer l and $\mathbf{b}^l = b_i^l$ is the bias vector containing bias of each neuron i in layer l. Moreover, the activation function $g(\cdot)$ is applied element-wise to the intermediate $\mathbf{z}^l$. In the case of the final layer $l = L$ it is common to use a different activation function appropriate for the current problem setting. For the case of regression, a linear mapping $g(x) = x$ is preferred. For the specific case $l = 1$, often referred to as the input layer, the activation $\mathbf{a}^1$ is simply given by a linear mapping of the input $\mathbf{x} \in \mathbb{R}^n$

$$\mathbf{a}^1 = \mathbf{x}. \tag{2.23}$$

By using a linear activation function g the MLP may only learn a linear mapping from the training data relating the input state to the output state. However, if the activation function is non-linear the MLP may learn to approximate more complex non-linear mappings. Common choices for activation functions include the *Rectified-Linear-Unit* given as $\text{ReLU}(x) = \max(0, x)$, the sigmoid function, given as $\text{Sigmoid}(x) = \frac{1}{1+\exp(-x)}$ or the tanh function, which are all shown in Fig. 2.3. Note that the ReLU is the most commonly used activation function [23] today.

2.2.2 Training

Since a MLP is a composition of differentiable parts the full MLP is differentiable as well. By defining a loss function

$$\mathcal{L}(f(\mathbf{x}, \mathbf{W}), \mathbf{y}) = \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}), \tag{2.24}$$

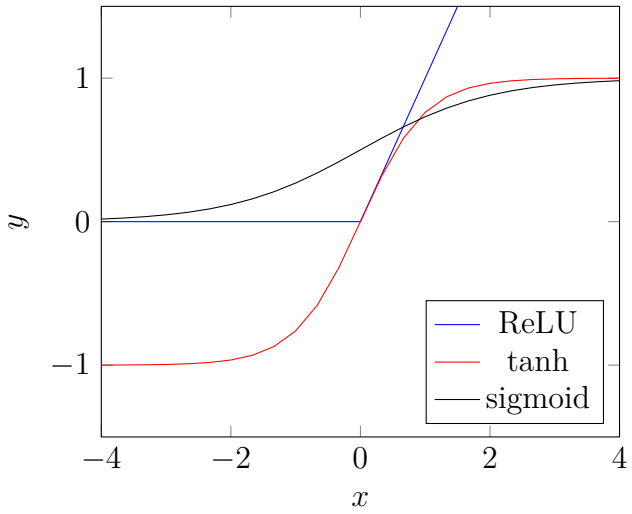


Figure 2.3: Common activation functions used for ANNs.

it is possible to train the MLP. As an example the Mean Squared Error between the predicted output and the true targets is often used as a loss function for regression.

By differentiating $\mathcal{L}$ with respect to the weights and biases of the model, it is possible to find how these contribute to the loss and adjust them in order to minimize the loss. This can be done effectively using the chain rule and memoization to compute the gradients of the loss function with respect to the weights and biases. This process is commonly referred to as *backpropagation* and is the standard way of training ANNs.

Starting from the output layer the error δ^l can be computed recursively as follows

$$\delta^{l-1} = g'(\mathbf{z}^l) \odot (\mathbf{W}^l)^T \delta^l \tag{2.25}$$

$$\delta^L = g'(\mathbf{z}^L) \odot \nabla_{\mathbf{a}^L} \mathcal{L}, \tag{2.26}$$

where g' is the derivative of the activation function and $\odot$ indicates element-wise multiplication. The gradients of the loss function with respect to the parameters of a layer l is then given by

$$\nabla_{\mathbf{W}^l} \mathcal{L} = \delta^l (\mathbf{a}^{l-1})^T \tag{2.27}$$

$$\nabla_{\mathbf{b}^l} \mathcal{L} = \delta^l. \tag{2.28}$$

Given the gradients of the loss function with respect to the weights and biases it is possible to use gradient descent methods to minimize the loss

$$\mathbf{W}^l \leftarrow \mathbf{W}^l - \eta \nabla_{\mathbf{W}^l} \mathcal{L} \tag{2.29}$$

$$\mathbf{b}^l \leftarrow \mathbf{b}^l - \eta \nabla_{\mathbf{b}^l} \mathcal{L}, \tag{2.30}$$

where η is the step size, or more commonly referred to as the *learning rate*. Using the full dataset to update the parameters of the ANN is referred to as one *epoch*. It is common to train the ANN for multiple epochs using the training data multiple times to minimize the loss.

Although it is in principle possible to compute the gradients using the full dataset, this is rarely done. Instead, it is more common to compute the gradient using small subsets of the full dataset, so called *mini-batches*, and update the weights and biases. Thus, one epoch consists of multiple weight and bias update steps. The main benefit of using mini-batch gradient descent, also known as stochastic gradient descent, is that it renders the training process tractable for large datasets and it may improve the ANNs ability to generalize [24].

Adam [25] is a popular optimizer algorithm based on stochastic gradient descent. It uses an adaptive learning rate involving the current and previous gradients. This is a handy feature, since the selecting a suitable learning rate is often non-trivial. Adam is usually a good first choice, even though other optimizers may be more efficient in certain cases.

From the general formula of the gradient for an arbitrary layer, presented in Eqs. (2.27) and (2.28), a problem with using the the sigmoid and tanh as activation functions becomes apparent. Namely, they saturate for large positive and negative inputs z, that is $g' \to 0$, causing the gradient to vanish. This happens since

$$|\nabla_l \mathcal{L}| \sim (g'(z))^{L-l}, \tag{2.31}$$

where $|\nabla_l \mathcal{L}|$ is the magnitude of a gradient vector for the layer l. This slows down learning and is called the vanishing gradient problem that can be especially problematic for deeper networks with many layers. A solution to the vanishing gradient problem is to use the ReLU activation function which has a constant derivative of one for $z > 0$.

2.2.3 Regularization

ANN models may vary infinitely by adjusting the number of hidden layers as well as the number of hidden neurons in them. Smaller models have in general less capacity, meaning that they cannot learn the most complex behaviors. Too large models may on the other hand be prone to over-fitting the data instead of generalizing. Over-fitting an ANN is basically the same concept as for FLR, where a complex high order model would fit all training points. By over-fitting the model to the training data it does not generalize well to new unseen data resulting in poor performance. There are however methods for reducing over-fitting, often referred to as *regularization*.

While training an ANN a validation set is often used. This is an often small subset of the dataset, and is different to the training set. After each epoch the validation set is used to evaluate the loss without updating the weights. This is done in order to determine whether the model is still learning or if the model has begun over-fitting the training data. The validation loss usually decreases for the initial epochs until, at some point, it starts increasing again. The increase of validation loss is a sign of over-fitting and the training may be stopped at that point. This regularization method is called *early stopping*.

Another common method of regularization is L_1 or L_2 regularization, where a penalty is imposed on the norm of the weights. This penalty is added to the loss

function in order for the training process to favour smaller weights, resulting in a simpler model. Specifically, the penalty is given by

$$\lambda \sum_k |w_k| \quad \text{or} \quad \lambda \sum_k w_k^2 \tag{2.32}$$

for the L_1 and L_2 case respectively, where w_k are the weights of the model and $\lambda > 0$.

Other regularization methods include *weight decay* [26] where in addition to updating the weights according to the gradients in Eqs. (2.27) and (2.28), the update step also subtracts a constant times the weight. This procedure reduces the norm of the weights, resulting in a simpler model.

Another regularization method is *dropout* [27] where the activation of a neuron is set to zero with some probability $p \in (0, 1)$. Dropout can be used for all neurons or just be applied to single layers and is often only used during the training process. Using dropout forces neurons to learn different things and use the available information more efficiently, thereby reducing over-fitting and improving generalization.

2.2.4 Ensembles

While a single ANN may be performing well for a given task, it is also possible to use a collection of multiple ANNs trained for the same task. This is commonly known as an ensemble of ANNs and has some advantages over single models.

One approach is to use an ensemble with models of the same architecture with randomly initialized weights and biases. The random weight initialization and stochasticity in the training should bring the models into different local minima of the loss function and by that give them slightly different predictive properties [14]. It is also possible to train the models on different subsets of the data if a sufficient amount of data exists.

A trained ensemble may then be used for predictions in multiple ways. One way to do this is weighting ensemble members differently based on loss function scores. However, since the models are trained on the same data, similar losses are likely obtained and a simple average prediction may be used instead

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} \hat{y_m}. \tag{2.33}$$

Here, $\hat{y}$ refers to the ensemble prediction and $\hat{y}_i$ to the individual model predictions. The averaging properties of an ensemble tend to give more robust and accurate predictions than a single model.

Another useful property of using an ensemble is the sample variance of the ensemble members, which could be used as an uncertainty estimate. The sample variance is given by

$$\sigma^2 = \frac{1}{M} \sum_{m=1}^{M} (\hat{y_m} - \hat{y})^2, \tag{2.34}$$

and is an indicator of the assembly disagreement. Models vastly disagreeing for some given sample may be an indication that the result should be looked at with caution.

2.3 Dynamic and Kinematic Models

In order to describe the time evolution of a system, a dynamic model can be used. Such a dynamic model describes the movement of an object over time as a function of the forces applied to the object. A common dynamic model used to describe vehicles is the Bicycle model.

In the context of lane departure detection it can be reduced to an in-lane form considering only the lateral and yaw dynamics. In this setting the dynamics of the bicycle model takes the form [28]

$$m\dot{v}_y = -mv_x\dot{\psi} + 2(F_f + F_r) \tag{2.35}$$

$$J_z\ddot{\psi} = 2(l_f F_f - l_r F_r) \tag{2.36}$$

where m is the mass of the vehicle, J_z the yaw inertia, l_f and l_r are the distance from the front and rear axles to the center of gravity, v_x and v_y the longitudinal and lateral velocity, F_f and F_r are the lateral force applied to the front and rear axles, and $\dot{\psi}$ is the turning rate with the orientation ψ defined with regards to a global frame. While the vehicle frame is defined such that the y-axis points to the left side of the vehicle and the x-axis points forward.

Performance comparisons in TA are often presented relative to some reference model. It is therefore important that the reference model is simple to implement, such that the reported performance is reproducible. Although it is possible to use a dynamical model as a reference model, it has some drawbacks. Namely the fact that a dynamical model requires multiple parameters, specific to the vehicle and which may vary over time, to be estimated in order to produce accurate predictions. Therefore, recreating an equivalent dynamical model as a baseline can be difficult for others who are using a different dataset. Especially since datasets in the automotive industry are sometimes proprietary, as is the case for the dataset used in this thesis.

A solution to these problems is to use a simple kinematic model as a reference model instead, where the movement of the vehicle is described using Newton's laws of motion, removing the need to estimate a large number of parameters. In this sense the kinematic model is agnostic to the properties of the specific vehicle and dataset and is therefore easy to reproduce.

In the automotive industry it is common to use kinematic models to compute SBTMs. By only considering a single future behaviour, it is possible to use simple model based state propagation to derive threat level metrics. These metrics can be divided into time, distance or acceleration domains. One such threat metric is the Time-to-Lane Crossing (TLC).

Although SBTM often produce simple threat metrics with a low computational cost, they represent the threat level in an idealized setting. Namely, they assume that in

any given situation the intention of every participant is known, or at least does not influence the trajectory mandated by physical state propagation. This assumption is reasonable for sufficiently short prediction horizons where every traffic participants trajectory can be considered deterministic.

The estimated TLC can be derived using a Constant Velocity Model (CVM). By assuming that the instantaneous lateral velocity will remain constant until a lane crossing event occurs. This assumption will produce accurate TLC predictions given a sufficiently straight road, which is a valid assumption for highway and country road driving conditions.

In order to compute the estimated TLC using a CVM, lateral velocity and lateral distance to either lane marker are needed. The lateral velocity $u^\diamond$ can be found from longitudinal velocity v as well as the heading $\alpha^\diamond$ relative to either lane marker. The lateral velocity is given by

$$u^\diamond = \sin(\alpha^\diamond)v\,, \tag{2.37}$$

where $\diamond$ is a wildcard symbol used to represent either the left or right side of the vehicle. Dividing the lateral distance to either lane marker with the associated lateral velocity

$$\mathrm{TLC} = \frac{d^\diamond}{u^\diamond}, \tag{2.38}$$

yields the estimated TLC. The TLC can then be used for a decision of whether an autonomous intervention is necessary.

2.4 Sources of Uncertainty

Uncertainty estimation is a valuable asset in data science and machine learning where it can provide additional insights when using model predictions. For example, a decision based on an erroneous prediction could be avoided if the model prediction is accompanied by a high uncertainty estimate. Furthermore, it is possible and useful to decompose uncertainty into two different types. Namely, by decomposing the uncertainty into aleatoric and epistemic uncertainty it is possible to better understand and use uncertainty measures.

Collected data generally contains inherent noise, affecting the output of a predictive model. Sensor inaccuracies or the outcome of a random process, such as a roulette wheel, are examples of such uncertainties. In Bayesian modelling, uncertainties inherent to the data are referred to as aleatoric uncertainty [29]. The aleatoric uncertainty can not be reduced by collecting more data, which perhaps is most clear in the roulette wheel example. While being irreducible, the aleatoric uncertainty can still be estimated.

Epistemic uncertainty, or model uncertainty, corresponds to facts that could be known to the model but are not. Unlike aleatoric uncertainty, it is possible to reduce the epistemic uncertainty by adding more informative data. Hence, an infinitely large dataset, exploiting the entire sample space, would tend the epistemic uncertainty to zero.

Inspired by [30], one can use the law of total variance in an attempt to formalize the notion of aleatoric and epistemic uncertainty. Using the variance of a random variable $y \in \mathbb{R}$ as a measure of uncertainty it may be decomposed as

$$\text{Var}[y] = \underbrace{\mathbb{E}_{q(\mathbf{W})}[\text{Var}[y|\mathbf{W}]]}_{\text{Aleatoric}} + \underbrace{\text{Var}_{q(\mathbf{W})}[\mathbb{E}[y|\mathbf{W}]]}_{\text{Epistemic}}, \tag{2.39}$$

where $\mathbf{W}$ is the parameters of the model and the subscript $q(\mathbf{W})$ indicates that the quantity is taken over the posterior of $\mathbf{W}$. The first term is the average of $\text{Var}(y|\mathbf{W})$ over $q(\mathbf{W})$, as such this term ignores any contribution to the uncertainty of y coming from $\mathbf{W}$, and can be interpreted as aleatoric uncertainty. While the second term is the variance of $\mathbb{E}[y|\mathbf{W}]$ when $\mathbf{W} \sim q(\mathbf{W})$, and therefore only considers the contribution of $\mathbf{W}$ to the uncertainty.

Moreover, from this decomposition it is also possible to see that the epistemic uncertainty can be removed provided that sufficient training data exist. Specifically, as more information is accumulated the posterior $q(\mathbf{W})$ will concentrate and the second term will vanish. At the same time $\mathbb{E}_{q(\mathbf{W})}[\text{Var}[y|\mathbf{W}]] \to \text{Var}[y]$, leaving only the intrinsic uncertainty as expected.

Furthermore, it is useful to distinguish two different types of noise or variance, which in turn gives rise to aleatoric uncertainties. The two different assumptions on the variance can be divided into homoscedastic and heteroscedastic variance. In the former case, the variance is constant and independent of the input state. While in the latter case the variance varies with the input state. Although, the homoscedastic case is easier to model, it is not a valid assumption in some scenarios.

2.5 Uncertainty Aware Neural Networks

ANNs may be used to capture the different types of uncertainty. As previously mentioned, it is possible to use an ensemble of standard MLPs in order to get a sample variance corresponding to an epistemic uncertainty estimate. This section presents different ANN approaches capable of estimating uncertainty, namely Gaussian MLPs, Bayes By Backprop and Monte Carlo Dropout. Note that only Gaussian MLPs and ensembles of such models are considered in this work, and that the remaining methods are provided for additional information on how uncertainty may be estimated using ANNs.

2.5.1 Gaussian MLPs

One way for an ANN to estimate uncertainty is to make it approximate a Gaussian distribution. This can be done by training the model to output the Gaussian distribution parameters μ and σ^2 through a special Gaussian output layer. So, instead of just giving a point estimate $\hat{y}(x)$ for each input x, the Gaussian output layer produces the parameters of a Gaussian distribution. Each prediction then consists of the Gaussian mean $\hat{\mu}$ with variance $\hat{\sigma}^2$ as an aleatoric uncertainty estimate. Such a layer cannot be trained by the MSE loss function, since an explicit target for the

output variance does not exist in general. However, by using a training criterion defined by the maximization of the likelihood, the uncertainty may be inferred without an explicit target. In the rest of this section an example will be presented, in order to provide some intuition of how the uncertainty may be inferred.

Although it is possible to approximate $y = \hat{f}(x)$ directly, it is sometimes insufficient in real-world situations, since in most real-world situations the measurements $y(x)$ are corrupted by noise. Therefore, it is reasonable to assume that the measurements $y(x)$ can be modeled as

$$y(x) = f(x) + \varepsilon(x), \tag{2.40}$$

where $f(x)$ is the true value corrupted by additive heteroscedastic noise $\varepsilon(x) \sim \mathcal{N}(0, \sigma^2(x))$. The observations $y(x)$ are then given by $y(x) \sim \mathcal{N}(\mu(x), \sigma^2(x))$, where $\mu(x) = f(x)$.

Given a dataset containing targets $y_1, \ldots, y_N$ corrupted by noise and corresponding inputs $x_1, \ldots, x_N$, it is possible for the model to also infer the predictive uncertainty associated with some input x. Assuming the measurements y_i to be a sample from a Gaussian distribution with the predicted mean $\hat{\mu}(x_i)$ and variance $\hat{\sigma}^2(x_i)$ parameterized by the Gaussian output layer, the model may learn to estimate the uncertainty. In practice this is realized by maximizing the likelihood

$$p(y_1, \ldots, y_N | x_1, \ldots, x_N) = \prod_{i=1}^{N} p(y_i; \hat{\mu}(x_i), \hat{\sigma}^2(x_i)), \tag{2.41}$$

with the assumption that $y_1, \ldots, y_n$ are independent. It is equivalent and numerically preferable instead to maximize the logarithm of the likelihood function

$$\sum_{i=1}^{N} \log p(\hat{y}_i; \hat{\mu}(x_i), \hat{\sigma}^2(x_i)), \tag{2.42}$$

where each term of the sum takes the form

$$-\frac{1}{2}\log \hat{\sigma}^2(x_i) - \frac{1}{2\hat{\sigma}^2(x_i)}(y_i - \hat{\mu}(x_i))^2 - \frac{1}{2}\log(2\pi). \tag{2.43}$$

By convention, training a neural network is a minimization task and the final loss function is instead the Negative Log Likelihood function (NLL) [14]

$$\text{NLL} = \frac{1}{2}\log \hat{\sigma}^2(\mathbf{x}) + \frac{1}{2\hat{\sigma}^2(\mathbf{x})}(y - \hat{\mu}(\mathbf{x}))^2 + C, \tag{2.44}$$

where C is a constant. With the NLL loss function it is therefore possible to capture the inherent noise of the data without explicitly having a target for it. An MLP with a Gaussian output layer is here referred to as a Gaussian MLP (GMLP), and an example of such a network is illustrated in Fig. 2.4.

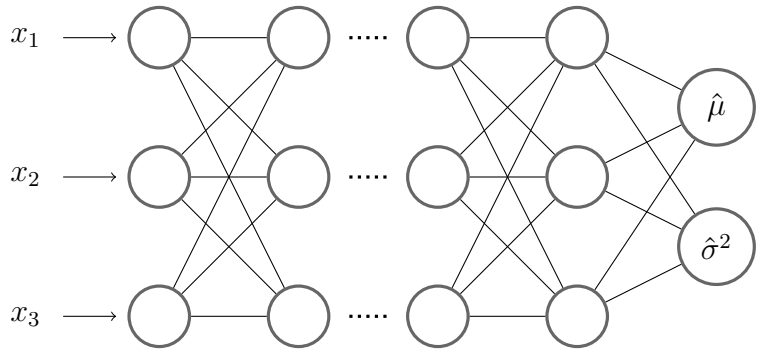


Figure 2.4: A schematic representation of a neural network with three input nodes and a Gaussian output layer parameterizing a Gaussian distribution.

2.5.2 Gaussian Ensembles

Ensembles of standard MLPs (MLPE) are, as mentioned earlier, useful for producing more robust outputs and for providing an epistemic uncertainty estimate using the sample variance. A Gaussian MLP ensemble (GMLPE) has the same advantages, but is also able to estimate aleatoric uncertainty. Such an ensemble may be seen as a Gaussian mixture model

$$p(y; \mu(\mathbf{x}), \sigma^2(\mathbf{x})) = \frac{1}{M} \sum_{m=1}^{M} \mathcal{N}(y; \mu_m(\mathbf{x}), \sigma_m^2(\mathbf{x})). \tag{2.45}$$

A Gaussian mixture model is a sum of multiple Gaussian distributions and is often used in clustering applications[16], but may also be used for regression. The mixture prediction of the GMLPE then consists of the mixture mean $\mu(\mathbf{x})$ and variance $\sigma^2(\mathbf{x})$, given by

$$\mu(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \mu_m(\mathbf{x}) \tag{2.46}$$

$$\sigma^2(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{M} \left(\sigma_m^2(\mathbf{x}) + \mu_m^2(\mathbf{x})\right) - \mu^2(\mathbf{x}). \tag{2.47}$$

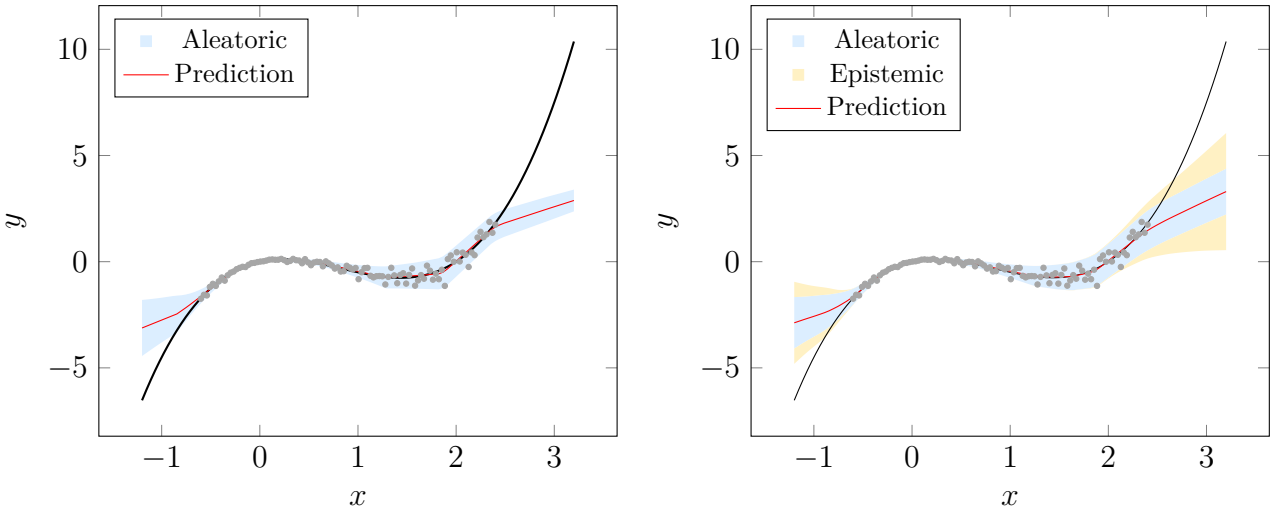
The mixture variance may be seen as a total uncertainty measure, and can therefore be divided into an aleatoric and an epistemic uncertainty estimate. This is done by rewriting the total uncertainty given in Eq. (2.47) and dividing it into an aleatoric and epistemic uncertainty:

$$\sigma^2(\mathbf{x}) = \frac{1}{M} \sum_{m=1}^{N} \sigma_m^2(\mathbf{x}) + \frac{1}{M} \sum_{m=1}^{N} \mu_m^2(\mathbf{x}) - \left(\frac{1}{M} \sum_{m=1}^{N} \mu_m(\mathbf{x})\right)^2 = \underbrace{\mathbb{E}[\sigma_m^2]}_{\text{Aleatoric}} + \underbrace{\text{Var}[\mu_m]}_{\text{Epistemic}} \tag{2.48}$$

A regression toy example in Fig. 2.5 illustrates the benefits of the different uncertainty estimates obtained using a GMLP and a GMLPE. The black line indicates ground truth with training samples denoted as gray dots. The samples are drawn with additive heteroscedastic Gaussian noise $\varepsilon \sim \mathcal{N}(0, \sigma^2(x))$ with σ^2 varying as

$$\sigma^2(x) = (|x| + 0.2)^2. \tag{2.49}$$

The GMLP manages to follow the ground truth well and gives a good aleatoric uncertainty estimate that captures the varying noise level. The GMLPE is also able to capture the aleatoric uncertainty, but additionally produces a more reliable total uncertainty estimate outside the training interval of $x \in [-0.6, 2.4]$.



(a) Gaussian MLP **(b)** Gaussian MLP Ensemble

Figure 2.5: A GMLP and a GMLP ensemble evaluated on a toy regression task with heteroscedastic noise. The uncertainty estimates are illustrated as two standard deviations ($\mu \pm 2\sigma$). Note that the aleatoric and epistemic uncertainty estimates for the GMLP ensemble are stacked so that the outer bound represents the total uncertainty.

2.5.3 Bayes By Backprop

Bayesian Neural Networks (BNN) are an extension of standard MLPs that are capable of providing an uncertainty estimate to the output. This is done by using Bayesian posterior inference [31], much like what is done in BLR.

Blundell *et al.* [19] propose a BNN method referred to as *Bayes by backprop*. This approach uses a posterior network where each weight in the network is represented by some probability distribution. At test time, weights are drawn from their respective distribution and the inputs are propagated through the fixed weight network. The output is then averaged over multiple such network samples to obtain a predictive posterior distribution.

The true Bayesian posterior can be expressed using the Bayes rule,

$$p(\mathbf{W}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{W})p(\mathbf{W}), \tag{2.50}$$

where $p(\mathcal{D}|\mathbf{W})$ being the likelihood and $p(\mathbf{W})$ the prior. Given the prior and likelihood, it is possible to measure how the posterior approximation differs from the true posterior. A measure that does this is the Kullback-Leibler (KL) divergence and is defined as

$$\mathrm{KL}(p||q) = \int_{-\infty}^{\infty} p(\mathbf{x}) \log\left(\frac{p(\mathbf{x})}{q(\mathbf{x})}\right) \mathrm{d}x\,, \tag{2.51}$$

where p and q are probability distributions.

The authors of [19] suggest training the models by using a variational approximation $q(\mathbf{W}|\theta)$ of the Bayesian posterior on the weights. This approach leads to finding the network parameters θ that minimise the KL divergence between the predictive posterior distribution, being the BNN, and the true Bayesian posterior

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \operatorname{KL}[q(\mathbf{W}|\theta)||p(\mathbf{W}|\mathcal{D})]. \tag{2.52}$$

Here θ^* represents the optimal model parameters and $p(\mathbf{W}|\mathcal{D})$ is the sought true Bayesian posterior. The posterior can be rewritten in terms of a known prior and a likelihood in order to define a loss function minimizing the KL divergence

$$\mathcal{L}(\mathcal{D},\theta) = \operatorname{KL}[q(\mathbf{W}|\theta)||p(\mathbf{W})] - \mathbb{E}_{q(\mathbf{W}|\theta)}[\log p(\mathcal{D}|\mathbf{W})]. \tag{2.53}$$

The second term acts as a regularizer and is needed to prevent the weight distributions from narrowing down too quickly during training.

This approach of BNNs uses backpropagation to update the model parameters. However, normal gradients are not used due to the weights being represented by probability distributions and thereby random variables. So called unbiased Monte Carlo gradients can then be used instead. The derivatives are taken with respect to the expectation of the random variable, which in this case may be expressed as the expectation of a derivative. This enables an approximation of the loss function as

$$\mathcal{L}(\mathcal{D},\theta) \approx \sum_{t=1}^{T} \log q(\mathbf{W}_t|\theta) - \log p(\mathbf{W}_t) - \log p(\mathcal{D}|\mathbf{W}_t), \tag{2.54}$$

where the loss is averaged over T that sampled weight configurations $\mathbf{W}_t$.

As for BLR, the choice of prior and likelihood determines the posterior distribution shape. For a Gaussian prior and likelihood the posterior will also be Gaussian, even though an analytical solution is intractable. At test time, samples are then drawn from the posterior network in order to get an approximate predictive posterior for a certain input. The characteristics of the predictive posterior is then used as a predictive mean with an associated uncertainty measure.

2.5.4 Monte Carlo Dropout

Dropout [27] is, as previously mentioned, a method commonly used to prevent overfitting the data in an ANN. During training, the activation of any neuron is set to zero with a certain probability $1 - p$. Note that also input neurons may be set to zero, although with a lower probability. The authors of [27] suggest using $p_h = 0.5$ for hidden neurons and $p_i = 0.8$ for input neurons. The procedure of dropping out neurons can be done for every training sample or batch-wise for higher computational efficiency. At test time, all neurons are used, but their outputs need to be scaled by p_h and p_i to compensate for the higher number of neurons.

It is also possible to utilize dropout during test time. Then, the same data is passed forward through the network multiple times, but with a new dropout configuration

each time. Doing this results in different outputs and a predictive mean and variance measure may be calculated.

The method of dropping out neurons at test time is called Monte Carlo Dropout (MCD) and according to the authors, the technique approximates Bayesian inference. They show, in their appendix paper [32], that MCD minimizes the KL-divergence between the approximate distribution and the posterior $p(\mathbf{W}|\mathbf{y},\mathbf{X})$ of a deep Gaussian process, with the minimization objective being

$$-\int q(\mathbf{W}) \log p(\mathbf{y}|\mathbf{X},\mathbf{W}) \, \mathrm{d}\mathbf{W} + \mathrm{KL}(q(\mathbf{W})||p(\mathbf{W})). \tag{2.55}$$

Here $q(\mathbf{W})$ represents the approximate distribution obtained by MCD, $p(\mathbf{y}|\mathbf{X},\mathbf{W})$ the deep Gaussian process likelihood and its prior $p(\mathbf{W})$. The exact motivation for MCD approximating the posterior of a deep Gaussian process is quite extensive and as this method is not used in the thesis, we recommend reading Section 3 in [32]. Weight decay as described in Section 2.2.3 is also utilized in the example implementation of MCD.

Formalized, the predictive mean using MCD is obtained as:

$$\mathbb{E}_{q(y|x)}[\mathbf{y}] = \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}(\mathbf{x},\mathbf{W}_t), \tag{2.56}$$

where $\mathbf{W}_t$ are the sampled weight configuration for a forward pass t. This can be recognized as model averaging for any probabilistic model. The uncertainty measure then basically consists of the sample variance over the T forward passes plus the inverse model precision as

$$\mathrm{Var}_{q(y|x)}[\mathbf{y}] = \tau^{-1}\mathbf{I}_D + \frac{1}{T} \sum_{t=1}^{T} \hat{\mathbf{y}}(\mathbf{x},\mathbf{W}_t)^T \hat{\mathbf{y}}(\mathbf{x},\mathbf{W}_t) - \mathbb{E}_{q(y|x)}[\mathbf{y}]^T \mathbb{E}_{q(y|x)}[\mathbf{y}], \tag{2.57}$$

where $\mathbf{y}$ is represented as a row vector and τ^{-1} denotes the inverse model precision. τ^{-1} is dependent on the design choices for the model and is defined as

$$\tau^{-1} = \frac{2N\lambda}{pl^2}, \tag{2.58}$$

where N is the number of training samples, λ the weight decay rate, p the dropout rate and l is the prior length scale used during training.

2.6 Uncertainty Calibration

Calibration is, in the field of uncertainty aware machine learning methods, a quality measure of the predicted uncertainty. The underlying idea of this is that a well calibrated model should output uncertainty estimates that match the true uncertainty. Multiple metrics exist for evaluating the calibration, some of which are presented here. Note that the GMLP based models proved to be well-calibrated for the problem studied in in thesis, and that additional calibration was therefore not necessary. A brief explanation of calibration methods is still included for reference.

2.6.1 Proper Scoring Rules

Scoring rules are used to quantify how well a model predicts its own uncertainty and are usable when comparing models [14]. Scoring rules are defined to follow

$$S(p_\theta,q) = \int q(y,\mathbf{x})S(p_\theta,(y,\mathbf{x}))\ \mathrm{d}y\,\mathrm{d}\mathbf{x}\,, \tag{2.59}$$

where $p_\theta(y|\mathbf{x})$ is the predictive distribution and $q(y|\mathbf{x})$ the true distribution.

Scoring rules that fulfil $S(p_\theta,q) \geq S(q,q)$ are said to be proper scoring rules, with equality only if the predictive distribution and the true distribution are the same. This means that a proper scoring rule may be minimized as a loss function, which actually is the case for the Gaussian MLP models used in this thesis.

There exist many proper scoring rules such as Negative Log Likelihood (NLL), Brier score [33] and Expected Calibration Error (ECE) [34]. NLL is a common loss function for ANN training, and is what is used to train the Gaussian MLPs. Brier score is used to measure accuracy in a classification setting, where the model outputs a per class probability.

ECE is a measure for finding how well calibrated model uncertainty estimates are, and can be used for both classification and regression [35]. ECE is in the classification case given by

$$\mathrm{ECE}_{class} = \sum_{j=1}^{J} \frac{|B_j|}{N}(|\mathrm{Acc}(j) - \mathrm{Conf}(j)|). \tag{2.60}$$

This formula is perhaps best explained by an example.

In the case of classification, the output probabilities are binned into J equally sized intervals. A resolution choice of 1 % results in a bin width of 1 %. For each bin B_j, the difference between the average true accuracy, $\mathrm{Acc}(j)$, and the predicted confidence, $\mathrm{Conf}(j)$, is calculated. View the confidence bin $B_j = [79\ \%, 80\ \%)$ specifically. First, all predictions with a confidence level in B_j are collected. Denote the fraction of predictions in B_j being correct as $\mathrm{Acc}(j)$ and the confidence level, $\mathrm{Conf}(j) = 79.5\,\%$, as the mean of the bin boundaries. Then, the absolute difference between $\mathrm{Acc}(j)$ and $\mathrm{Conf}(j)$ is calculated. For a well calibrated model, $\mathrm{Acc}(j)$ should be close to $\mathrm{Conf}(j) = 79.5\,\%$. Each confidence bin may contain different numbers of samples $|B_j|$. Each interval is therefore also weighted by the number of samples in it, divided by the total number of samples, N.

In the regression case there are no probabilities of a given output. Instead, confidence intervals are binned up in the same manner as the previously mentioned class probabilities. Now, $\mathrm{Acc}(j)$ denotes the fraction of targets being inside a confidence interval $\mathrm{Conf}(j)$. All predicted points are used for every confidence interval. Therefore, no weighting has to be done and the ECE is given by the average error of all intervals as

$$\mathrm{ECE}_{regr} = \frac{1}{J}\sum_{j=1}^{J}(|\mathrm{Acc}(i) - \mathrm{Conf}(i)|). \tag{2.61}$$

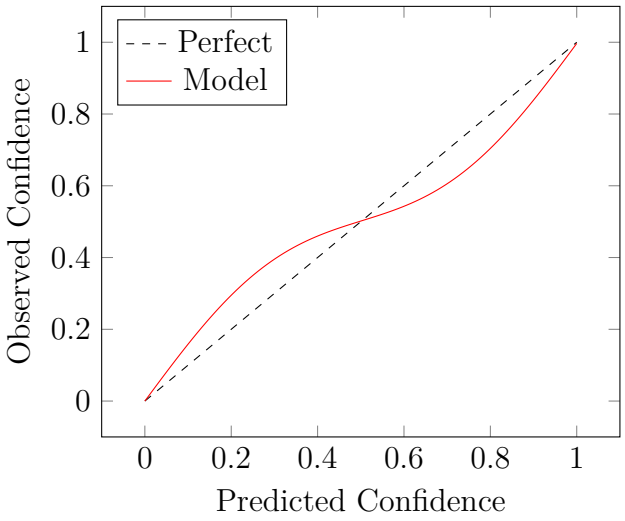


Figure 2.6: Example of a reliability plot. In this case model is underconfident for the smaller confidence intervals and overconfident for the larger.

A natural extension to the ECE is reliability plots. These are used to see how well calibrated the models are for the different class probabilities (classification) and confidence intervals (regression). Here, the unweighted quantity $\text{Acc}(j)$ is plotted as a function of $\text{Conf}(j)$. The desired look of the plot is a unit mapping between $\text{Acc}(j)$ and $\text{Conf}(j)$ for all confidence levels. Higher accuracy than confidence means that the model is underconfident whereas lower accuracy corresponds to an overconfident model. This is illustrated in Fig. 2.6.

2.6.2 Calibration of Uncertainty

If the proper scoring rules indicate an uncalibrated model, it is possible to calibrate the models to compensate for this. Many such methods can be applied after training the models, allowing for calibration of already existing models. Methods for this include temperature scaling [13], which recalibrates the uncertainty estimates without altering the accuracy of the model. Temperature scaling is commonly applied to the softmax outputs of a classification model as

$$p_c = \frac{\exp(z_c/T)}{\sum_{c=1}^{C} \exp(z_c/T)} \tag{2.62}$$

and adjusts the shape of the probability density function. During training the temperature, $T = 1$ is used [36].

Much calibration work has been conducted towards classification, which is a seemingly easier task than calibrating regression models. Classification has a fixed number of discrete classes while regression is continuous.

Some calibration methods for regression are inspired by classification, where the regression outputs are binned into smaller discrete intervals. Keren *et al.* [36] propose using a softmax output layer for regression value intervals. This turns the regression

problem into a classification task that may be calibrated using temperature scaling for example.

Calibration methods tailored for regression do however exist. Quantile calibration [34] and distribution calibration [37] are examples of such methods. Quantile calibration adds a calibration mapping function $c(\tau)$ to the regression outputs. Much like ECE and the reliability plots, the calibration is computed for a given quantile τ of the cumulative density function (CDF). The key idea is to use isotonic regression for finding $c(\tau)$ such that the model fulfills

$$\mathrm{P}(Y \leq y|X = x) = \tau \tag{2.63}$$

for all quantiles $\tau \in [0,1]$. Isotonic regression is a parameter free model that may learn the true distribution given enough i.i.d data [34].

Song *et al.* [37] argue that quantile calibration is a global calibration method that does not take the individual predictions into account. The model is calibrated on average, but not in individual cases. Their method, distribution calibration, resembles quantile calibration but instead uses a β-calibration map [38] to get a local calibration. The β-calibration map has three parameters that are found using Gaussian process regression.

2.7 Ensemble Distillation

Ensembles of standard or Gaussian MLPs are used to increase the predictive performance by averaging over multiple models to yield a more robust prediction. While the increased performance of an ensemble is desirable, it comes at the cost of an increased computational complexity. For some applications this may be acceptable and justified by the performance boost. However, some environments are very limited in terms of computational resources and using an ensemble model may not be feasible. Being able to transfer the predictive properties of an ensemble down to a single network would reduce the computational complexity and make the model suitable for a wider variety of applications. This process is called ensemble distillation, and in this section two such methods are presented.

Buciluǎ *et al.* [39] propose to train a distillation model using the same inputs as the ensemble, but to use the ensemble output as the training target. This should give the distilled model similar predictive properties as the ensemble for the training data. However, to further make the distilled model behave like the ensemble, additional unlabeled training data should be used as well using the ensemble to annotate the data. In cases where additional training data cannot be collected, synthetic data can be used instead. Synthetic data may be generated in different ways, such as distorting inputs or swapping input features of the existing training data. The authors do however note that real data is preferable to synthetic data if available.

The presented distillation method [39] is capable of reducing the ensemble down to a single network while keeping much of the ensemble performance.Lindqvist *et al.* [40] extend this work, by presenting a framework for distilling Gaussian MLP

ensembles that also retains the ensembles' ability to estimate aleatoric and epistemic uncertainty. The epistemic part is added to the model by outputting a higher order probability distribution than the original ensemble members. The specific case of Gaussian MLP ensemble is used as an example here, but the framework is general and may be applied to other uncertainty aware ensemble models.

In essence, a one-dimensional GMLP outputs a Gaussian distribution parameterized by a vector

$$\mathbf{z} = [\mu, \sigma^2], \tag{2.64}$$

with the predictive mean μ and aleatoric uncertainty σ^2. While the elements of $\mathbf{z}$ parameterize a probability distribution, they may also be seen as random variables. Hence, it is possible to model μ and σ^2 as Gaussian distributions themselves. This can be achieved by setting the individual ensemble member outputs as targets for the distilled model. In practice, the number of training samples is scaled by the ensemble size.

The reason for doing so is that the mean, aleatoric and epistemic uncertainty may then be predicted by a single network. This, such that μ and σ^2 are set as regression targets, resulting in an output vector of

$$\mathbf{z_{distilled}} = (\mathbb{E}[\mu], \mathrm{Var}[\mu]), (\mathbb{E}[\sigma^2], \mathrm{Var}[\sigma^2]). \tag{2.65}$$

Here, $\mathbb{E}[\mu]$ and $\mathbb{E}[\sigma^2]$ are the expected mean and aleatoric variance and are used as such. $\mathrm{Var}[\mu]$ now represents the spread of the ensemble, and is used as the epistemic uncertainty estimate. The remaining element, $\mathrm{Var}[\sigma^2]$ is the variance of the aleatoric variance, and is not used here. This, since only the mean, aleatoric and epistemic variances correspond to ensemble properties properties sought to adapt. A total uncertainty estimate can be obtained by summing the aleatoric and epistemic parts.

3 Method

This chapter is divided into four parts. The first part focuses on the dataset with its signals, data extraction and annotation process. The second part focuses on the model implementation and training procedure. While the third part describes the Threat Assessment system. Finally, the fourth part details the evaluation process in terms of regression and threat assessment.

3.1 Data

The following section describes the dataset used in this thesis, starting with an explanation of the sensor readings. The data extraction and annotation process is then presented.

3.1.1 Signal Configuration

The dataset used in this thesis is proprietary and managed by Zenuity. It contains time series data of in-car sensor readings, sampled at $f_s = 1/T_s = 40\,\text{Hz}$. The data was collected by different vehicles in multiple countries under various weather conditions. The dataset mainly represents the state of the car as well as the properties of the road approximately 100 m ahead. The dataset consists of multiple signals, among which:

- lane-marker polynomials $p_t^\diamond$ relative to the ego vehicle estimated from a forward facing camera,
- range of view $r_t^\diamond$ for which the polynomials are valid,
- longitudinal velocity v_t,
- yaw rate ω_t,
- front wheel angle δ_t,
- the longitudinal acceleration a_t.

An illustration of the system and variables is provided in Fig. 3.1.

The left and right approximate lane-marker polynomial p_t^l and p_t^r, respectively, at time instance t are defined as follows:

$$p_t^\diamond(x_t) = a_{0,t}^\diamond + a_{1,t}^\diamond x_t + a_{2,t}^\diamond x_t^2 + a_{3,t}^\diamond x_t^3, \tag{3.1}$$

where x_t is the longitudinal distance and $\diamond$ can be replaced to obtain the polynomial for the respective side. Using this definition, the distance to either lane marker at the current time instance t is given by $d_t^\diamond = p_t^\diamond(0) = a_{0,t}^\diamond$.

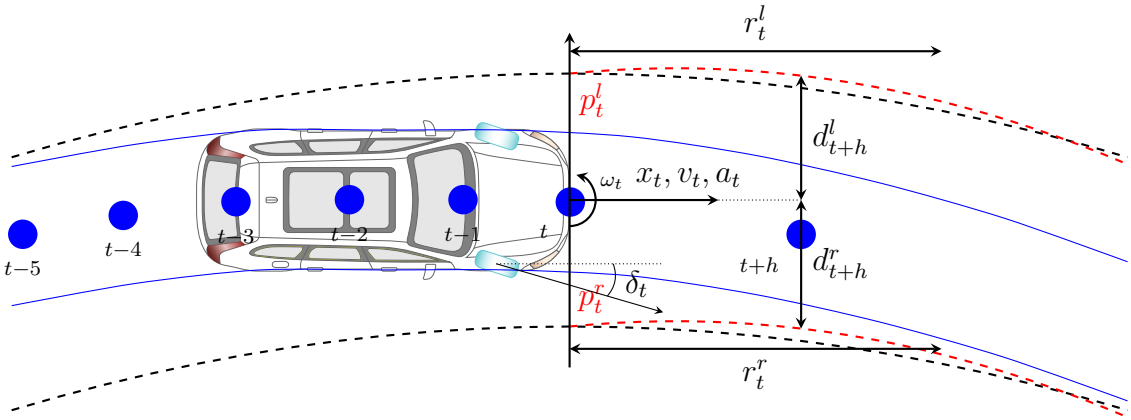


Figure 3.1: Illustration of the system and variables. The dashed black lines are the lane markers, while the red dashed lines are the estimated lane markers using a forward facing camera system. The car trajectory is represented by the solid blue lines.

3.1.2 Data Extraction

The dataset itself is large and collision free, and was pre-processed to only contain data from highway and country road driving. To achieve this only data satisfying the following criteria was extracted:

- Velocity must be higher than 60 km/h.
- The road must have a curve radius that is higher than 250 m.
- The lane width must not surpass 4 m.
- Sensor readings must exist for both lane markers.

The extracted data was then split into a negative set $\mathbf{N}$ and a positive set $\mathbf{P}$. The negative set $\mathbf{N}$ contains only normal driving sequences with no lane departures. The positive set $\mathbf{P}$ contains driving sequences ending in an unintended lane departure. The length of the driving sequences in $\mathbf{P}$ varies as a function of the prediction horizon H, where each sequence contains $h = Hf_s$ samples. Thus, a driving sequence in $\mathbf{P}$ corresponds to the last H seconds before a lane departure event.

In order to ensure that $\mathbf{P}$ is free from intended lane departures any departure events accompanied by a turn signal were discarded. Moreover, the vehicle should return to its original lane within 4 s, indicating an accidental lane departure. An example of a normal and a departure driving sequence is shown in Fig. 3.2.

In order to train the models and evaluate their positive performance an *event set* $\mathcal{C}$ was defined. A driving sequence $c \in \mathcal{C}$ was constructed starting from a driving sequence in $\mathbf{P}$ and extending it chronologically with the samples preceding it in $\mathbf{N}$. By varying the length of c, the relative abundance of normal and departure driving which the models are trained and evaluated on can be adjusted. By selecting an appropriate length of c the models can learn both driving scenarios. Setting $|c| = 4h$,

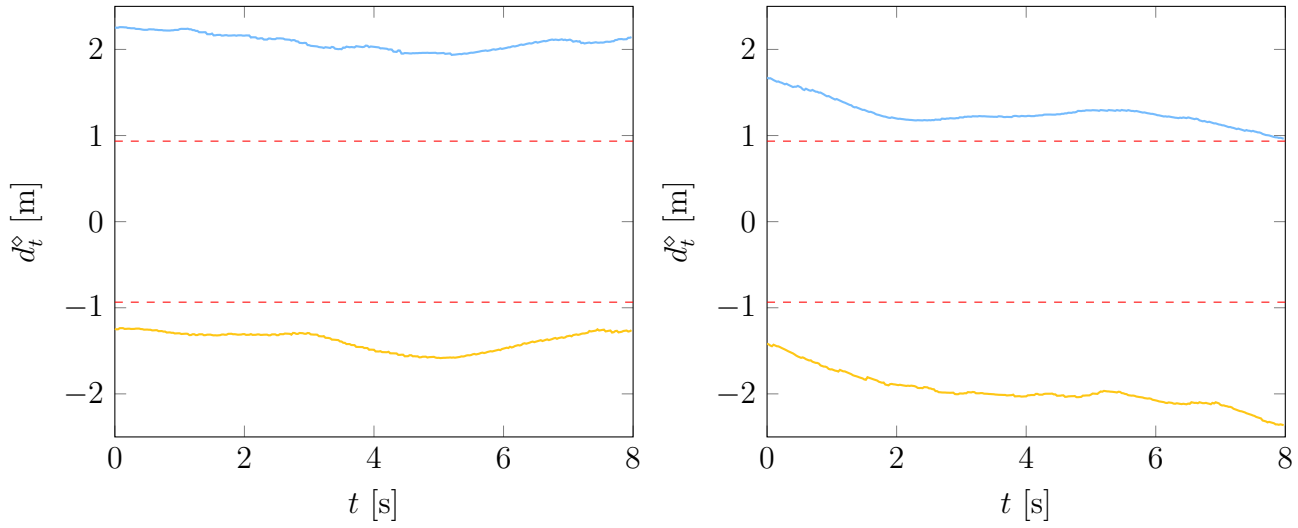


Figure 3.2: An example of normal (left) and a departure (right) driving sequence. The solid lines show the distance to the respective lane marker while the dashed red lines indicate the edge of the car.

that is $3h$ samples from $\mathbf{N}$ in addition to h samples from $\mathbf{P}$ empirically produced good results.

In addition to evaluating the models' positive performance, negative performance was evaluated by defining a *non-event set* $\mathcal{B}$ consisting of only normal driving. Thus an element $b \in \mathcal{B}$ is a driving sequence with no lane departures taken only from the negative set $\mathbf{N}$. Since the set $\mathcal{B}$ only contains normal driving there is no need to balance the data in the same fashion as for $\mathcal{C}$. The length of $b \in \mathcal{B}$ can therefore be chosen freely, and in this work $|b| = 10\,\text{s}$ was chosen. An illustration of the data extraction process is shown in Fig. 3.3.

The event set $\mathcal{C}$ contains 14703 unique driving sequences equivalent to roughly 16 h to 33 h depending on the chosen prediction horizon H. While the non-event set $\mathcal{B}$ contains 3000 driving sequences totaling approximately 8 h. Moreover, there is no overlap between the two datasets. That is $\mathcal{B} \cap \mathcal{C} = \emptyset$.

In order to train and evaluate the models implemented in this work the event set $\mathcal{C}$ was split into a training, validation and test set. The event set was split as follows: $|\mathcal{C}^{\text{trn}}| = 12645$, $|\mathcal{C}^{\text{val}}| = 1029$ and $|\mathcal{C}^{\text{tst}}| = 1029$.

3.1.3 Data Annotation

Since the data is comprised of time series, annotation for regression is straight forward. For any time instance t the model should predict the distance $d_{t+h}^\diamond$ to both lane markers at some prediction horizon H. The target $y_t^\diamond$ for a prediction horizon H is simply given by looking $h = f_s H$ steps forward.

$$y_t^\diamond = p_{t+h}^\diamond(0) = a_{0,t+h}^\diamond \,. \tag{3.2}$$

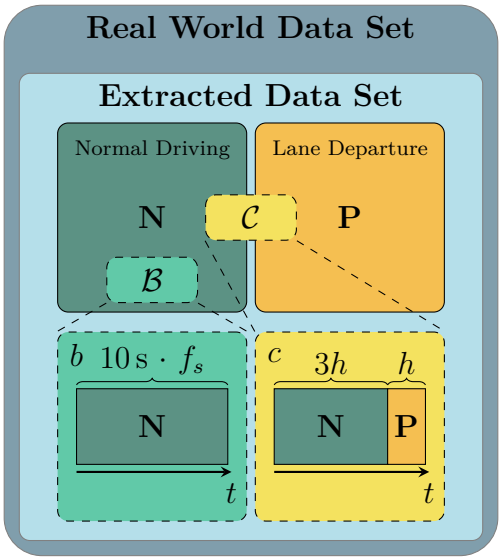


Figure 3.3: A schematic representation of the data extraction and selection process. The Extracted dataset consists of highway or country road driving sequences extracted from the full real-world dataset, using the criteria described in Section 3.1.2. The subsets **N** and **P** contain normal driving sequences and driving sequences ending in a lane departure, respectively. A member $c \in \mathcal{C}$ is constructed by taking, in chronological order, the first part from **N** and the last part from **P**. A member $b \in \mathcal{B}$ consist of a time series fully from **N**. Here h represents the prediction horizon in time steps, while f_s is the sampling frequency.

3.2 Threat Assessment

The following chapter describes the implementation details of the tested models. It includes the standard ML model as well as the uncertainty aware ML model, in addition to the kinematic baseline model.

3.2.1 Time Series Windowing

Time series regression using MLPs requires feeding an input vector with historical data to the model. Including such historical data consist of retrieving samples at previous time steps. In this thesis, a time series windowing inspired by [8] was used.

By using a fixed number of previous consecutive samples as input, trends in the data can be captured. However, the dataset used was collected at $f_s = 40\,\mathrm{Hz}$, meaning that the ego vehicle travels less than a meter between the samples at highway speeds. The state of the vehicle and the road therefore stay relatively constant between consecutive samples. Using a fixed number of consecutive previous samples may therefore add to the computational complexity without giving much new information.

Consequently, it may be beneficial to selectively assemble a subset of the previous

samples to constitute the model input vector. This can be done by applying a filter

$$\Gamma = \{\gamma_0, \gamma_1, \ldots, \gamma_n\}, \tag{3.3}$$

where $\gamma_i \in \mathbb{N}$ and γ_i is unique, that retrieves the desired samples. For every time step t in a driving sequence, an input vector $\mathbf{X}_t^{(\Gamma)}$ is given by

$$\mathbf{X}_t^{(\Gamma)} = \{\mathbf{x}_{t-\gamma_0}, \mathbf{x}_{t-\gamma_1}, \ldots, \mathbf{x}_{t-\gamma_n}\}, \tag{3.4}$$

where $\mathbf{x}_t$ is the vehicle and road state at time step t. A single training example was then defined as $\{\mathbf{X}_t^{(\Gamma)}, y_t^{\diamond}\}$.

3.2.2 Training

All models in this thesis were trained on driving sequences from the event set $\mathcal{C}^{trn}$ with various filters applied. In [8], the authors tackle a similar TA task on the same data as in this work, but with a linear vector auto-regressive model approach lacking uncertainty estimates. They show that a filter depth of 1 s e.g. considering only information from the last second, gives sufficient historic data for capturing a trend in the driving sequence, so to make accurate predictions. With this in mind filters with depth 0.5 s and 1 s were implemented. Specifically, four different filters shown in Table 3.1 were used.

Table 3.1: Filters of depth 0.5 s and 1 s used for time series windowing.

Filter	**Lags**
Γ_1	{0, 5, 19}
Γ_2	{0, 5, 39}
Γ_3	{0, 6, 13, 19}
Γ_4	{0, 7, 15, 23, 31, 39}

The models were designed to predict the lateral distance to the lane markers for a specific time horizon H. A perfect prediction would then return the exact position of the ego vehicle at a time H seconds into the future. Naturally it is harder to get accurate predictions for a longer prediction horizons. To investigate this matter the models were trained for different time horizons $H = \{1, 1.25, 1.5, 1.75, 2\}$ seconds.

Small models with three hidden layers of 10 neurons each were implemented. In our setting this results in a number of model parameters between 300 and 1800. The reason for using small models is that a linear auto-regressive model [8], with a similar number of model parameters, has proven to perform well on this specific regression task.

The standard ML as well as uncertainty aware ML models were then trained using the MSE and NLL loss function, respectively. Minimizing the loss was done using the Adam [25] optimizer with an initial learning rate of 0.001. The models were trained for 30 epochs and over-fitting was avoided by utilizing early stopping.

All model implementations were written in Python 3 using Tensorflow [41] as deep learning framework. Packages such as Numpy [42], Scipy [43], Scikit-Learn [44] and Pandas [45] were also frequently used in this work.

3.2.3 Gaussian Ensembles

The Gaussian ensemble models were trained as randomly initialized individual Gaussian MLPs using the training scheme above. This work leverages an ensemble of size 10 optimized by the negative log likelihood of the Gaussian probability density function (2.44). Note that the ensemble members stay independent until test time.

As to see whether training all ensemble members on the same data influenced the uncertainty estimate models were also trained on smaller subsets of the data. In this case, the ensemble members were trained on a random 2 % fraction of the training set.

3.2.4 Ensemble Distillation

Ensemble distillation was performed on the best performing Gaussian ensemble models, and was optimized using the NLL loss function 3.17. The same architecture of 3 hidden layers of 10 neurons each was used. As for the Gaussian ensembles, the entire training set was used. In practice this results in ten times the training samples, as the output from each ensemble member is used as a training target.

3.2.5 Kinematic Model

In Section 2.3 the CVM was described using the relative heading to each of the lane markers. However, this heading is not explicitly included in the dataset. Instead, it was computed using the instantaneous rate of change in $p_t^\diamond$ evaluated at $x_t = 0$.

Moreover, the CVM was previously shown computing the estimated TLC as the threat metric. Since the ML models used in this thesis were trained for time series regression of the lateral distance to the lane marker, it was more suitable to use a distance based threat metric. Therefore, the estimated distance to lane marker derived using a CVM was used as the underlying threat metric for the baseline kinematic model.

Starting from the road polynomial $p_t^\diamond$, as defined in Section 3.1.1, the instantaneous rate of change in the distance to the lane marker given by

$$\psi_t^\diamond = \left.\frac{dp_t^\diamond}{dx_t}\right|_{x_t=0} = a_{1,t}^\diamond, \tag{3.5}$$

can be used to find the heading $\alpha_t^\diamond$ from

$$\alpha_t^\diamond = \arctan\Big(a_{1,t}^\diamond\Big). \tag{3.6}$$

Given that $a_{1,t}^\diamond$ is small, which is the case for the used data, the lateral velocity can be found as

$$u_t^\diamond = v_t \sin \psi_t^\diamond, \tag{3.7}$$

where v_t is the longitudinal velocity at time step t. Assuming the lateral velocity to be constant over the prediction horizon H, the future distance to either lane marker is given by

$$\hat{d}^{\diamond}_{t+h} = p^{\diamond}_t(0) + u^{\diamond}_t H. \tag{3.8}$$

3.3 Decision Making

In this section, two intervention criteria for converting model predictions into a decision of whether an autonomous intervention is necessary, are introduced. Namely, an uncertainty aware criterion suitable for a GMLP based TA model, as well as a deterministic criterion using only the predicted future lateral position of the ego vehicle.

3.3.1 Deterministic Decision Making Method

A deterministic DM method uses the model's predicted lateral distance $\hat{d}^{\diamond}_{t+h}$ to the respective lane marker at time step $t+h$ to compute the associated overlap $\hat{o}^{\diamond}_{t+h} = \hat{d}^{\diamond}_{t+h} - \frac{w}{2}$ where w is the width of the car. An intervention criterion can then be defined as

$$I_{\text{Deterministic}}(\hat{o}^{\diamond}_{t+h}) = \begin{cases} 1, & \hat{o}^{\diamond}_{t+h} \leq \tau \\ 0, & \text{otherwise}\,, \end{cases} \tag{3.9}$$

where τ is a tunable parameter to determine the required overlap of the car and lane marker in order to trigger an autonomous intervention. The introduction of a tunable parameter τ is useful to adjust the properties of the DM process. Setting $\tau = 0$ would trigger an intervention as soon as the model predicts a future lane departure by a lane marker crossing. This intervention criterion is a natural choice for prediction models which only predict the future lateral position of the ego vehicle, such as the CVM and MLP based models. In the following discussion we refer to this intervention criterion as the *Deterministic* intervention criterion.

In the case of a GMLP the same intervention criterion can be utilized by using the expected value of the predicted distribution. That is

$$\hat{d}^{\diamond}_{t+h} = \mathbb{E}[\mathcal{N}(\hat{d}^{\diamond}_{t+h};\ \hat{\mu}^{\diamond}_{t+h}, (\hat{\sigma}^{\diamond}_{t+h})^2)] = \hat{\mu}^{\diamond}_{t+h}\,, \tag{3.10}$$

with $\hat{o}^{\diamond}_{t+h} = \hat{\mu}^{\diamond}_{t+h} - w/2$. An example using the deterministic intervention criterion with a GMLPE model is shown in Fig. 3.4.

3.3.2 Uncertainty Aware Decision Making Method

One way to utilize the uncertainty estimates of the model predictions is to calculate the probability of a lane departure according to the model. The probability of departure can then be used as a threat level metric used in the decision making process of whether an autonomous intervention is necessary. In this section the probability of departure is derived from the uncertainty aware ML model prediction and an uncertainty aware intervention criterion is defined.

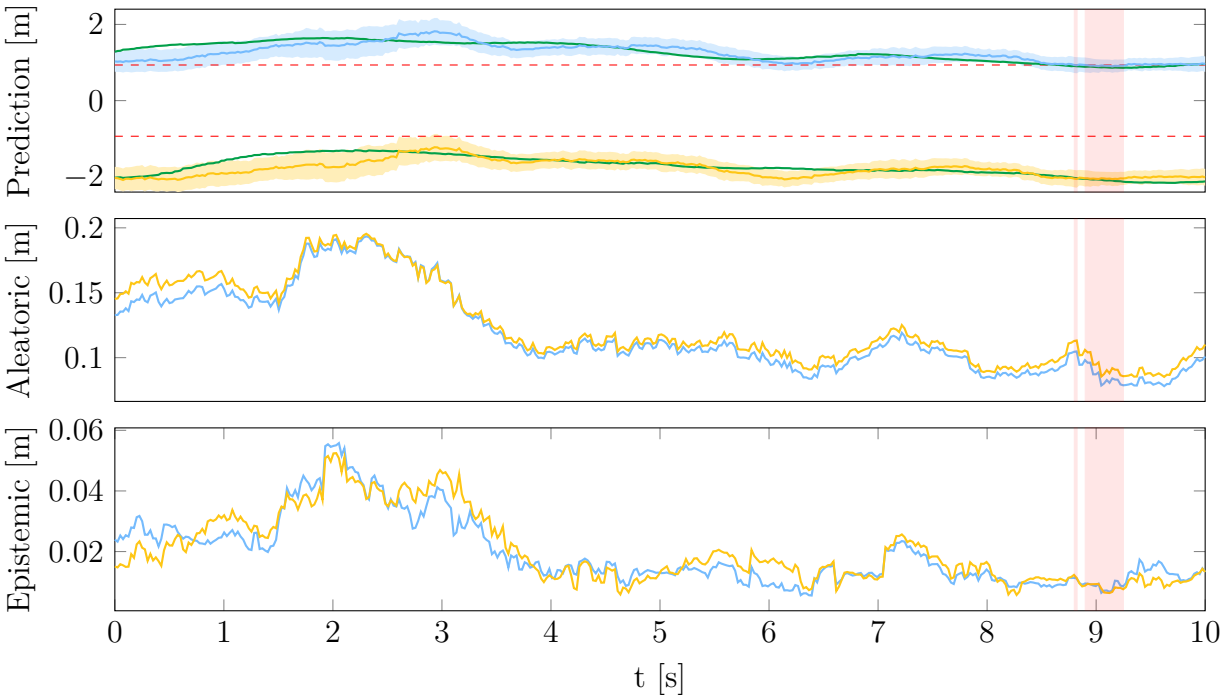


Figure 3.4: Model prediction for the left (—) and right(—) lane markers with ground truth (—) with a prediction horizon of 1.5 s. Along with the predicted lane markers a two sigma confidence band derived from the total uncertainty is shown. While the aleatoric and epistemic components of the uncertainty is shown in their respective plot. The edge of the car is marked by the dashed red lines while the red region indicates that an intervention was triggered by the model.

At each time instance t the ensemble prediction consists of a normal distribution

$$\hat{d}^{\diamond}_{t+h} \sim \mathcal{N}(\hat{\mu}^{\diamond}_{t+h}, (\hat{\sigma}^{\diamond}_{t+h})^2) \tag{3.11}$$

over the predicted lateral position of the lane marker at the time $t+h$. For ease of notation in the following discussion the subscript $t+h$ as well as the hat associated with model predictions are dropped.

The probability $q^{\diamond}_{t+h}$ of a lane departure can then be found by integrating

$$q^{l}_{t+h} = \int_{-\infty}^{\frac{w}{2}} \frac{1}{\sigma^l\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\mu^l}{\sigma^l}\right)^2} \,\mathrm{d}y \tag{3.12}$$

and

$$q^{r}_{t+h} = \int_{\frac{w}{2}}^{\infty} \frac{1}{\sigma^r\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\mu^r}{\sigma^r}\right)^2} \,\mathrm{d}y\,, \tag{3.13}$$

for the left and right side, respectively. Here w is the width of the car. Solving these integrals is equivalent to finding the predicted probability that the edge of the car crossed the lane marker, which is illustrated in Fig. 3.5.

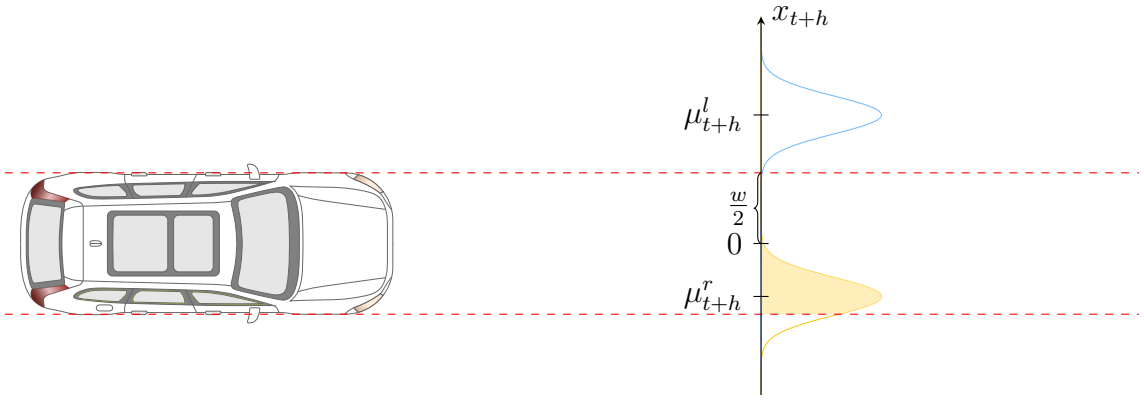


Figure 3.5: An example situation where the PD method shows a high probability of a lane departure to the right at $t+h$. The dashed red lines mark the edges of the car and the normal distributions shown in — and — are the predicted left and right lane marker distributions respectively.

Using the predicted probability of departure $q^{\diamond}_{t+h}$ as a basis for decision making is straight forward and the intervention criterion is defined as

$$I_{\text{PD}}(q^{\diamond}_{t+h}) \begin{cases} 1, & q^{\diamond}_{t+h} \geq \rho \\ 0, & \text{otherwise} \end{cases}, \tag{3.14}$$

where $\rho \in [0, 1]$ is probability threshold for when an intervention should be triggered. Note that although the full range of ρ is an acceptable parameter choice, restrictions are useful. Specifically, $\rho \in [0.5, 1)$ in order to produce sensible and achievable trigger criterion. Moreover, by setting $\rho = \frac{1}{2}$ this method reduces to the deterministic DM method discussed in Section 3.3.1. In the following discussion this method will be referred to as the *Probability of Departure (PD)* method.

The main benefit of using the PD method is its simplicity, requiring only a single parameter ρ to capture both the mean prediction as well as the associated predictive uncertainty into a single intervention criterion. Specifically, by using this criterion predicted small lane departures with high certainty will cause an autonomous intervention. Likewise, a high uncertainty prediction can trigger an intervention if the predicted lane departure is large enough.

In a similar fashion as for the deterministic intervention criterion described in Section 3.3.1, it is possible to introduce a tuneable parameter τ, in addition to ρ. Analogue to τ in the deterministic DM method, the properties of the PD method can be adjusted by changing the effective width of the car in Eq. (3.12) and Eq. (3.13) as follows

$$q^l_{t+h}(\tau) = \int_{-\infty}^{\frac{w}{2}+\tau} \frac{1}{\sigma^l\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\mu^l}{\sigma^l}\right)^2} \, \mathrm{d}y \,, \tag{3.15}$$

and similarly for the right side.

Changing the effective width of the car is equivalent to introducing new virtual lane markers by shifting the predicted location of the actual lane markers, which can

easily be seen through the variable substitution $y' = y - \tau$ in the previous integrals. By varying τ the trig time of an autonomous intervention can be controlled by changing the effective size of the car. A desired trig time can then be achieved by tuning τ on a calibration dataset. This process will be discussed more in depth in Section 4.5. Moreover, note that by using this modified PD method an intervention is triggered if the distance to the lane marker is τ or closer with a probability of at least ρ.

The tunable parameter τ introduced in the PD method is equivalent to the previously introduced thresholds for the kinematic CVM and the deterministic DM method introduced in Section 3.3.1. Larger values of τ corresponds to stricter intervention criterion which will result in a later effective trig time. Smaller values of τ instead results in earlier trig times.

3.4 Performance Evaluation

Trained models may be evaluated using different metrics. This section presents methods for evaluating the models in terms of statistical performance as well as performance within a TA system.

3.4.1 Statistical Performance

The statistical performance was evaluated primarily in terms of MSE

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3.16}$$

which is a common way to evaluate regression models. In this thesis, since the models predict the lateral distance to both lane markers, the reported MSE performance was the average of the MSE of the two outputs. Moreover, note that a low MSE does not always guarantee good performance in a TA system. This since the MSE represents the average quality of predictions, which may deviate in critical situations.

Moreover, the statistical performance for uncertainty aware GMLP models may also be evaluated using the Gaussian NLL function

$$\text{NLL} = \frac{1}{N}\sum_{i=1}^{N}(\frac{1}{2}\log\sigma^2(\mathbf{x}_i) + \frac{1}{2\sigma^2(\mathbf{x}_i)}(y_i - \mu(\mathbf{x}_i))^2 + \frac{1}{2}\log(2\pi)). \tag{3.17}$$

Also here the final loss is presented as the average error with respect to both lane markers. The last term in the sum is the numerical value of the constant in Eq. (2.44). This metric is natural to ensembles and single models of Gaussian MLPs as it was used as their training criterion. Standard MLP ensembles may also be evaluated with respect to this metric using their sample variance.

3.4.2 Threat Assessment Performance Evaluation

The performance of a TA system can be evaluated in terms of a confusion matrix. Specifically, the True-Positive-Rate (TPR) and False-Positve-Rate (FPR) was used

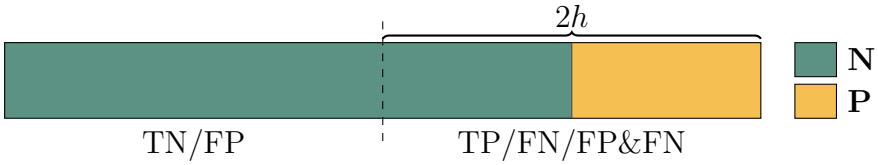


Figure 3.6: Schematic of departure evaluation. The last $2h$ samples of the driving sequence are considered as the accepted trigger window for an autonomous intervention.

to evaluate the model performance on both normal driving sequences as well as lane departure sequences. The models were evaluated on a test dataset $\mathcal{B}$ consisting of 3000 normal driving sequences each 10 s long and 1029 lane departure sequences in $\mathcal{C}^{\text{tst}}$.

Evaluation on a normal driving sequence $b \in \mathcal{B}$ is straight forward. One True Negative (TN) is determined if no interventions are triggered while one False Positive (FP) is determined if any intervention is triggered. Only the first intervention in any driving sequence is considered for evaluation purposes. This, since in a real life scenario, an autonomous steering intervention would be triggered, taking control away from the driver and disabling the TA system. Thus, the rest of driving sequence b following an intervention would not be realized in a real-world scenario, since the car is no longer actuated by the driver. Therefore, considering additional interventions shortly after the first, for evaluation purposes, would lead to an unrealistic performance evaluation. Moreover, it is non-trivial to determine exactly when the TA system should be activated again in an evaluation setting, and it is therefore simpler to disregard the rest of the current driving sequence.

Lane departure sequences' evaluation is a bit more complex due to the addition of a timing aspect of a successful intervention. Therefore an accepted trigger window of $2H$ s before the departure event was defined. Thus lane departure sequences consists of two parts. Namely, the first $2h$ samples which are considered normal driving and are evaluated as previously described, and the last $2h$ samples associated with the associated trigger window which is evaluated in a different manner. In the accepted trigger window, where an intervention for the correct side counts as one True Positive (TP). If the TA system triggers an intervention in the accepted trigger window for the incorrect side it is instead counted as a False Negative (FN) and a FP, since the true lane departure was not detected and an erroneous intervention was triggered. Once again, only the first intervention was considered for evaluation purposes, and as a result a FP in the first half of the driving sequence will also count as a FN since the true departure event was not caught. Evaluation on a departure sequence $c \in \mathcal{C}$ is shown schematically in Fig. 3.6.

4 Results

The results presented in this chapter are divided into two parts. The first part handles regression metrics as well as uncertainty estimates and calibration. The second part, starting in Section 3.3, treats the TA performance in terms of tuning the trig time, and evaluating and comparing TA performance. Finally the performance of a distilled model is presented.

4.1 Statistical Performance

Single models, ensembles of Gaussian models and standard MLPs were evaluated primarily in terms of MSE for the regression performance. The results are visualised in Tables 4.1 and 4.2 with the best result for each prediction horizon written in bold.

The standard MLPs were optimized using MSE as the loss function while the GMLPs used the NLL in (2.44). This does not seem to considerably affect the predictive error for this task, indicating uncertainty estimating capabilities may be added without sacrificing much predictive performance.

The difference between ensembles and single members is also small in this case, almost only visible for longer prediction horizons. Predicting on longer horizons is naturally a harder task, where the averaging properties of an ensemble come out ahead of the single models.

Table 4.1: Mean Squared Error of single GMLPs as well as GMLP ensembles.

Filter	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE
Γ_1	**0.004**	**0.004**	0.007	0.007	0.011	0.011	0.017	0.017	0.025	0.025
Γ_2	**0.004**	**0.004**	0.007	**0.006**	0.011	0.011	0.017	0.017	0.025	**0.024**
Γ_3	**0.004**	**0.004**	0.007	0.007	0.011	0.011	0.019	0.017	0.026	0.025
Γ_4	**0.004**	**0.004**	**0.006**	**0.006**	0.011	**0.010**	0.017	**0.016**	0.025	**0.024**

The Gaussian NLL evaluates the models' ability to make good predictions while also estimating an appropriate level of uncertainty. This is illustrated in Table 4.3 for single models as well as ensembles of Gaussian MLPs. Filter Γ_4 performs slightly better than the other configurations in this metric.

Out of curiosity, a similar test was conducted towards standard MLP ensembles using sample variance as uncertainty estimate. This resulted in very bad NLL values of around 30. A comparison like this is obviously unfair but shows that sample variance does not estimate well the actual predictive uncertainty.

Table 4.2: Mean Squared Error of single MLPs as well as MLP ensembles.

Filter	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	MLP	MLPE	MLP	MLPE	MLP	MLPE	MLP	MLPE	MLP	MLPE
Γ_1	0.004	0.004	**0.006**	**0.006**	0.011	0.011	0.017	0.017	0.026	0.024
Γ_2	0.004	0.004	0.007	**0.006**	0.011	**0.010**	0.017	**0.016**	0.024	0.024
Γ_3	0.004	0.004	**0.006**	**0.006**	0.011	0.011	0.018	0.017	0.026	0.024
Γ_4	0.004	**0.003**	**0.006**	**0.006**	0.011	**0.010**	**0.016**	**0.016**	0.024	**0.023**

Table 4.3: Gaussian Negative Log Likelihood of single Gaussian MLPs as well as Gaussian MLP ensembles. Lower values are better.

Filter	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE	GMLP	GMLPE
Γ_1	-2.406	-2.418	-2.112	-2.132	-1.879	-1.882	-1.605	-1.641	-1.421	-1.442
Γ_2	-2.405	-2.418	-2.103	-2.157	-1.855	-1.893	-1.625	-1.654	-1.431	-1.459
Γ_3	-2.412	-2.424	-2.126	-2.146	-1.877	-1.881	-1.615	-1.650	-1.370	-1.444
Γ_4	-2.448	**-2.473**	-2.175	**-2.187**	-1.878	**-1.910**	-1.654	**-1.677**	-1.435	**-1.460**

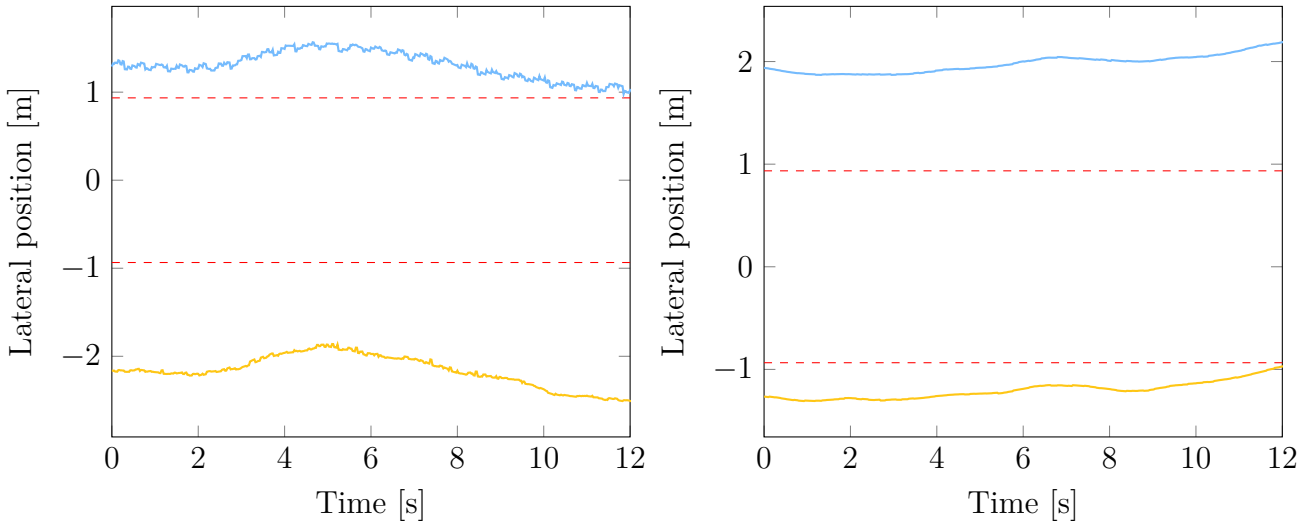
4.2 Aleatoric Uncertainty

The dataset has variations in noise among the different driving sequences. This could be due to different driving conditions affecting the data collecting sensors. Where worn lane markings, downpour or fog are specific examples of such conditions.

Uncertainty aware models should supposedly be able to distinguish between low and high noise data. Therefore models should, on average be more uncertain and output a higher aleatoric uncertainty for high noise data. Examples of high and low noise sequences are illustrated in Fig. 4.1.

A zero-phase digital second order low pass filter with 40 Hz sampling frequency and a cutoff frequency of 5 Hz was used to identify the noise level of the data. Filtering was done on a per driving sequence basis, assuming a fairly constant noise level within a driving sequence. The noise level for a driving sequence was calculated as the mean square distance between the filtered signal and the original signal. The driving sequences were then ranked and split by the noise median into high and low noise sets. The filter was only applied to the lateral position signals. Since these signals are most influential for regression as well as being straight forward to filter, the lateral position was used solely in identifying the noise level. Note that the models were trained on mixed noise data and that these sets were only used for evaluation.

Figures 4.2 and 4.3 illustrate how the uncertainty estimates in terms of standard deviation vary between the low high noise driving sequences. Uncertainty estimates



(a) High noise sequence

(b) Low noise sequence

Figure 4.1: Examples of driving sequences where the input signals are corrupted with high and low noise. Note that the inputs consist of multiple signals, although noise distinguishing was made solely on the lateral position signals as these are most important for regression.

for the full, undivided, set $\mathcal{C}^{tst}$ is also included for comparison.

According to Figs. 4.2 and 4.3 there is a difference between the different noise levels, indicating that models are able to detect noisy inputs and output a higher uncertainty estimate. The differences in uncertainty are relatively small for short prediction horizons. Noise is naturally more influential on longer prediction horizons as the error then propagates further. This explains why the models estimate a higher uncertainty level for a longer prediction horizon.

4.3 Epistemic Uncertainty

Theoretically, an ensemble of models trained on subsets of all training data should exhibit a higher epistemic uncertainty compared to an ensemble trained on the full dataset. This, since a subset of the data doesn't necessarily explore the sample space as the full training set does. Ensemble members trained mainly on different data should also learn different things. If so, there will be a spread among the ensemble members resulting in a higher epistemic uncertainty compared to models trained on the same data .

Figure 4.4 illustrates a comparison of uncertainties between a full set trained ensemble and a subset trained ensemble evaluated on $\mathcal{C}^{tst}$ using filter Γ_4. As expected, the subset ensemble feature a higher epistemic uncertainty than the full set ensemble. Also note an increase in aleatoric uncertainty, even if the relative difference to the full set aleatoric uncertainty is quite small. The epistemic uncertainty then constitutes a larger fraction of the total uncertainty for the subset model. This indicates

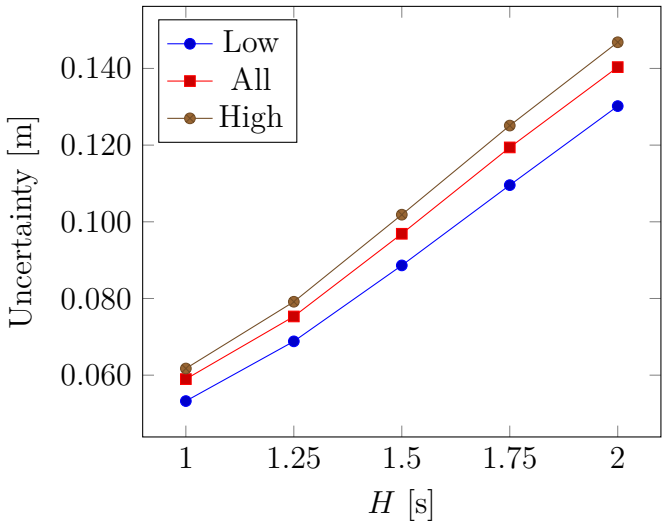
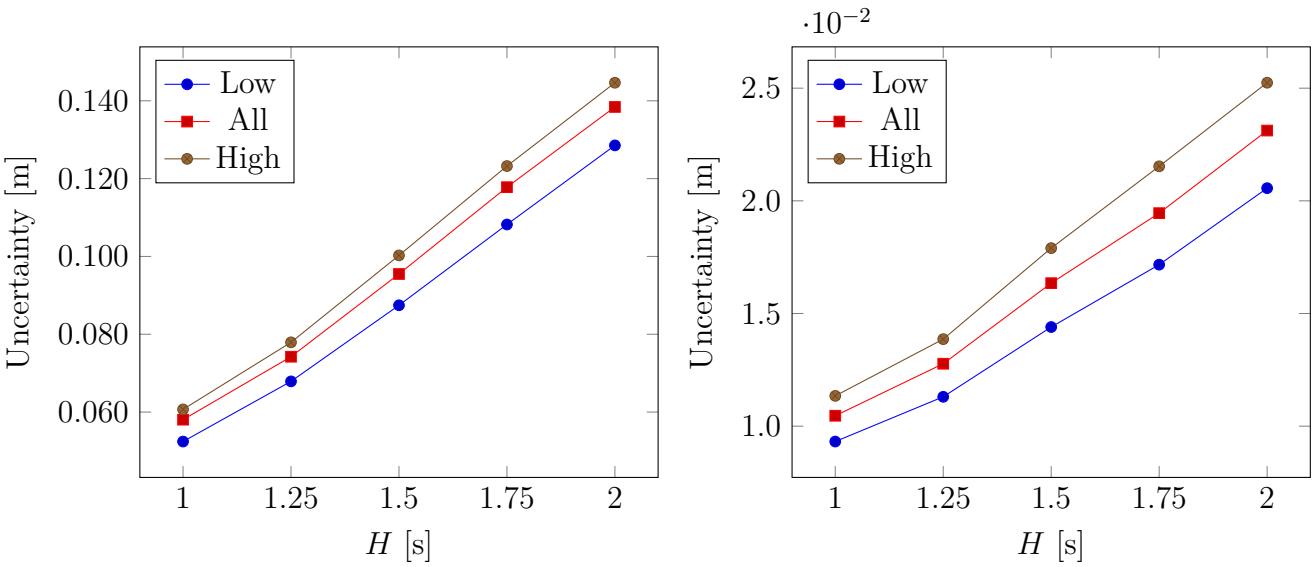


Figure 4.2: Total uncertainty estimates for low, high and all noise data for a Gaussian MLP ensemble using filter Γ_4. The all noise data is the undivided $\mathcal{C}^{tst}$



(a) Aleatoric

(b) Epistemic

Figure 4.3: Aleatoric and epistemic uncertainty estimates for low, high and all noise data for a Gaussian MLP ensemble using filter Γ_4. The all noise data is the undivided $\mathcal{C}^{tst}$.

that the epistemic uncertainty estimates indeed estimate model uncertainty from the absence of sufficient training examples.

4.4 Model Uncertainty Calibration

While NLL is indeed a proper scoring rule, it does not show for which confidence intervals the models are calibrated. Uncertainty estimates may for example be

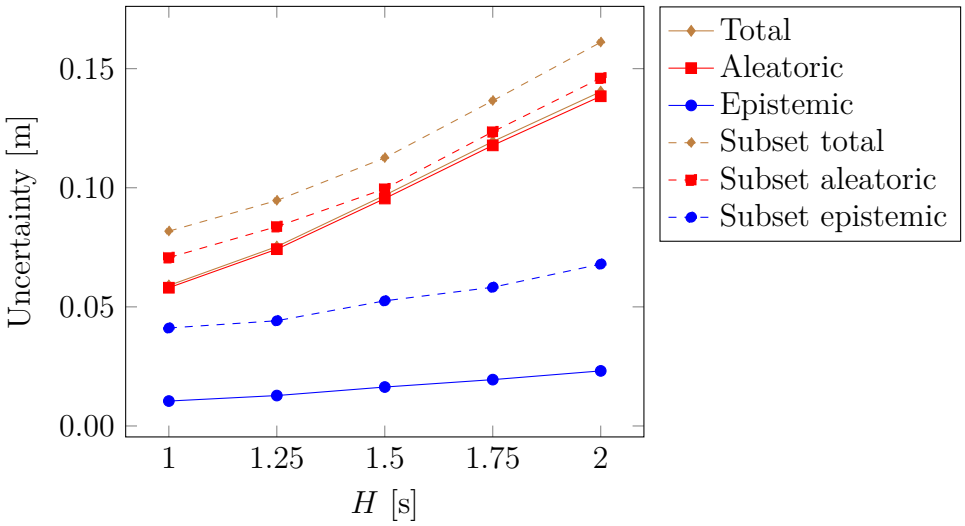


Figure 4.4: Uncertainty estimates for full dataset trained models and subset trained models. The epistemic uncertainty is higher for the subset models and makes up greater part of the total uncertainty compared to the full set models.

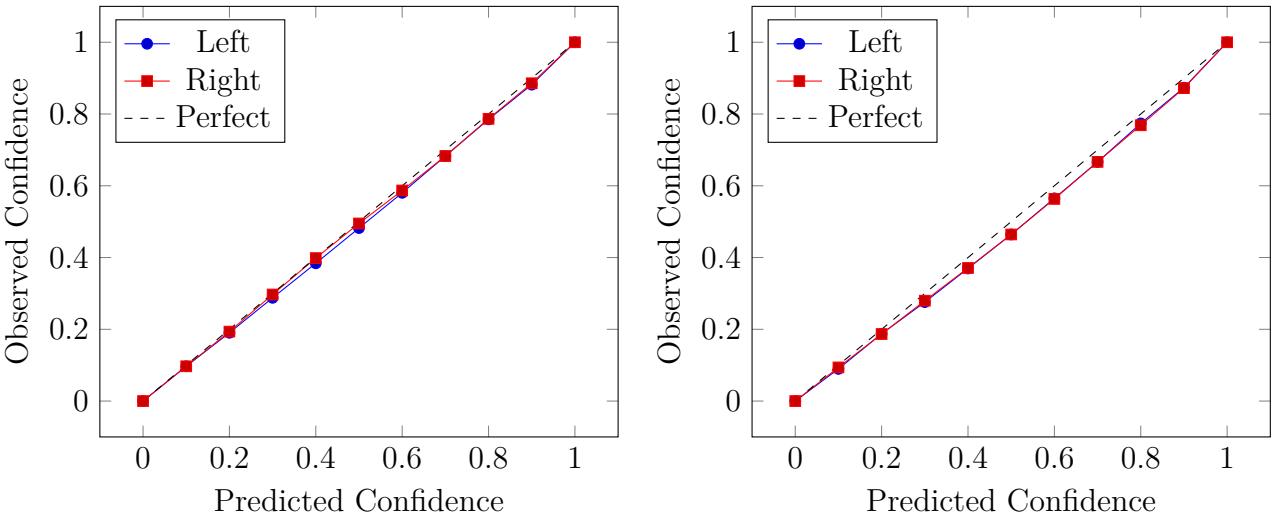
overconfident for a 50 % confidence interval, while being well calibrated for others. By calculating the fraction of predictions being inside a confidence interval of a given percentage as in Section 2.6.1, a reliability plot may be created. This tool makes it easy to check if the models are calibrated or not.

In Fig. 4.5 a reliability plot is shown for a GMLP ensemble as well as a single GMLP for comparison. They were evaluated on $H = 2$ seconds using filter Γ_4. The ensemble seems to be almost perfectly calibrated, while the single model is slightly overconfident. A standard MLP ensemble using sample variance was also evaluated for the same configuration. This results in overconfident predictions as shown in Fig. 4.6. It is unfair to compare GMLPs to MLPs in this matter, but it is clear that the sample variance alone is not a calibrated uncertainty estimate.

4.5 Trig Time Tuning

In order to make a fair comparison when evaluating different TA systems, all systems should have the same average trig time $\bar{t}_h$. Ensuring that any model does not gain an unfair advantage in performance as a result of a later trig time. As a result of the problem becoming inherently easier closer to the lane departure. Although a TA system is designed for a specific prediction horizon H, there is no guarantee that the mean trig time $\bar{t}_h$ matches the one specified by the design prediction horizon. The trig time of a TA system can be tuned by adjusting the parameter τ, such that the average trig time on a calibration dataset matches the desired trig time.

The appropriate threshold τ_* for the desired trig time H was found by systematically testing different values of τ and record $\bar{t}_h$. This process is repeated until the set of $\bar{t}_h$ on the calibration data encompasses the desired trig time. The optimal threshold τ_* such that $\bar{t}_h = h$ was then found by linear interpolation. This process is described



(a) Gaussian MLP Ensemble

(b) Single Gaussian MLP

Figure 4.5: Reliability plot for a single GMLP and a GMLP Ensemble at $H = 2$ using filter Γ_4. For this configuration the GMLP ensemble proves to be well calibrated, while the single GMLP is slightly overconfident overall.

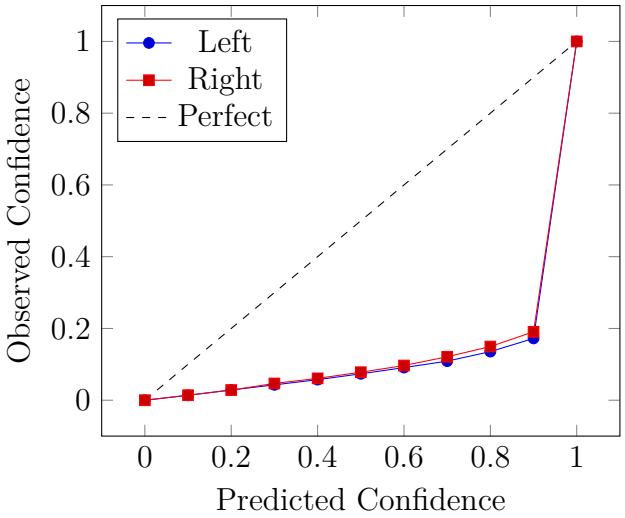


Figure 4.6: Reliability plot for an MLP ensemble at $H = 2$ using filter Γ_4, showing overconfident predictions.

in Algorithm 1.

The trig time of all evaluated TA systems were tuned using Algorithm 1 on the validation dataset $\mathcal{C}^{\text{val}}$ using a threshold step size of $\Delta\tau = 0.01$. An example of the trig times evaluated on the test set $\mathcal{C}^{\text{tst}}$ before and after tuning is shown in Fig. 4.7.

From Fig. 4.7 it is obvious that the distribution of trig times is skewed and that the mean trig time $\bar{t}_h$ does not match the mode of the distribution. To account for this in the trig time tuning it would be possible to tune using the mode of the distribution instead of the mean. However, the mode is dependent on the bin size

Algorithm 1: Tune trig time of TA-system.

Result: Tuned threshold τ_*

$\tau \leftarrow 0$

$t \leftarrow$ `MeanTrigTime(`τ`)`

$\boldsymbol{\tau} \leftarrow \tau$, $\mathbf{t} \leftarrow t$ // save to list

if $h \leq t$ **then**

  while $h \leq \max(\mathbf{t})$ **do**

    $\tau \leftarrow \tau - \Delta\tau$

    $t \leftarrow$ `MeanTrigTime(`τ`)`

    $\boldsymbol{\tau} \leftarrow \tau$, $\mathbf{t} \leftarrow t$ // save to list

  end

else

  while $h \geq \min(\mathbf{t})$ **do**

    $\tau \leftarrow \tau + \Delta\tau$

    $t \leftarrow$ `MeanTrigTime(`τ`)`

    $\boldsymbol{\tau} \leftarrow \tau$, $\mathbf{t} \leftarrow t$ // save to list;

  end

end

$\tau_* \leftarrow$ `Interp1D` $(\mathbf{t}, \boldsymbol{\tau}, h)$

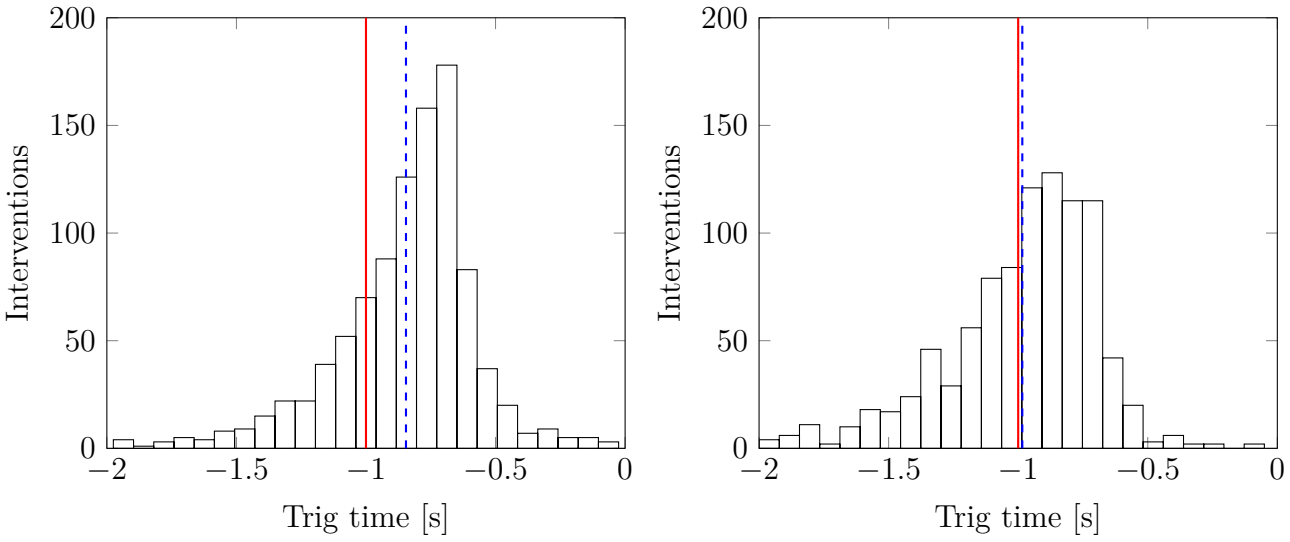


Figure 4.7: Histogram of trig times for a model with a desired prediction horizon (——) of 1 s with actual mean trig horizon (- - -) before and after tuning.

used and it is not obvious how one could automatically select an appropriate bin size. Moreover, the discrepancy between mode and mean will not impact the relative TA performance evaluation between models since they are all tuned in the same way. Thus, trig time calibration using the mean remains suitable.

4.6 Threat Assessment Performance

In this section the properties of the implemented TA methods are investigated and their performance evaluated. In order to facilitate fair TA performance comparisons the trig time of all evaluated models were tuned using the process described in Section 4.5. Moreover, the proprieties of the proposed uncertainty aware TA model is investigated and compared to a deterministic TA model. The best performing ML based TA models are then compared against each other and a baseline kinematic CVM.

4.6.1 Deterministic Threat Assessment Method Performance

To better facilitate TA performance comparisons between different models it is useful to reduce the number of models to compare, by only comparing the best candidates. In order to determine the best candidate for the respective model using the deterministic intervention criterion one can compare the TA performance for the different filters. This is shown in Figs. 4.8 and 4.9 for MLPEs and GMLPEs, respectively.

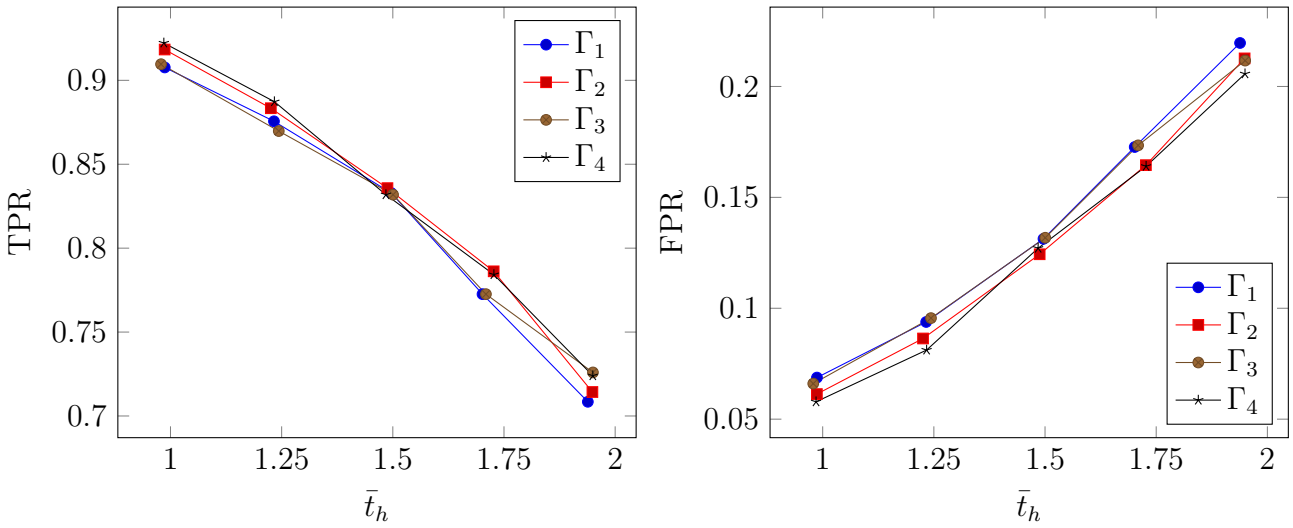


Figure 4.8: TA performance comparisons for MLPE models using the deterministic intervention criterion method for different filters.

According to Figs. 4.8 and 4.9, Γ_4 show the best TA performance, both in terms of TPR and FPR. However, in the case of the GMLPE, Γ_4 were late in its mean trig time on $\mathcal{C}^{\text{tst}}$ for a prediction horizon of 2 s, which might make for an unfair performance gain. Therefore, in the case of deterministic GMLPE, Γ_2 was used for future performance comparisons while for a MLPE using the deterministic intervention criterion Γ_4 was the reference.

Among the tested filters there is no significant difference in TA performance using the deterministic TA method for filters of the same depth. Indicating that the number of samples in a filter does not impact performance. There is however, a possible difference in performance for filters of different depth. Lending support to

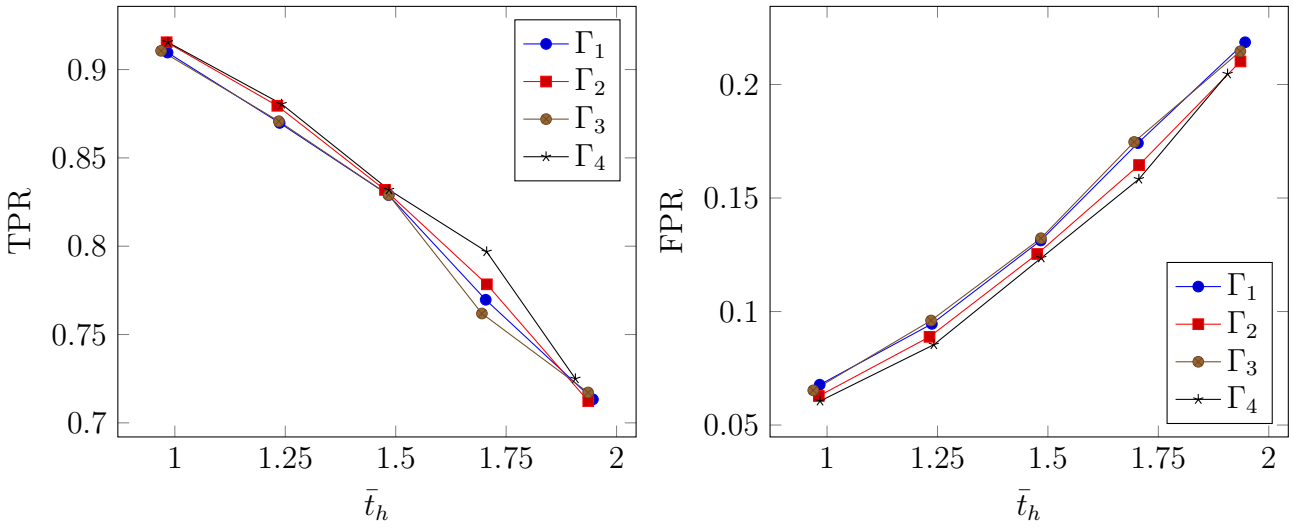


Figure 4.9: TA performance comparisons for GMLPE models using the deterministic intervention criterion for different filters.

that filters that take into account information further back in time may perform better.

4.6.2 Properties of Uncertainty Aware Threat Assessment Method

The introduction of the probability threshold ρ in the PD method mandates additional tuning in addition to the trig time tuning of τ, as described in Section 4.5. To investigate the effects of ρ on TA performance, multiple values of ρ were sampled and the models positive and negative performance was evaluated on the test set $\mathcal{C}^{\text{tst}}$. Specifically, the TA performance was evaluated for $\rho = \{0.6, 0.7, 0.8, 0.9\}$, and the results are shown in Tables 4.4 and 4.5.

Table 4.4: Positive model performance using the trig time tuned PD method for the best probability threshold ρ value for respective filter and prediction horizon. The best TPR is emphasized in bold for each prediction horizon. Additionally, ρ values which optimized both positive and negative performance for the given filter and prediction horizon are highlighted.

Filter	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	TPR	ρ	TPR	ρ	TPR	ρ	TPR	ρ	TPR	ρ
Γ_1	0.911	0.7	0.869	0.6	0.830	**0.6**	0.767	0.6	**0.718**	**0.7**
Γ_2	0.916	**0.6**	0.879	**0.7**	**0.834**	0.7	0.784	**0.6**	0.713	**0.6**
Γ_3	0.914	0.7	0.872	0.6	0.829	**0.7**	0.765	0.6	0.715	**0.6**
Γ_4	**0.920**	0.7	**0.885**	0.7	0.838	0.7	**0.791**	**0.6**	0.716	**0.6**

From the TPR performance of the PD model shown in Table 4.4, Γ_4 performs best

on most prediction horizons. Although the differences between using different filters are relatively small. In terms of FPR performance, Table 4.5, Γ_4 shows the best performance on all prediction horizons.

Table 4.5: Negative model performance using the tuned trig time PD method for the best probability threshold ρ value for respective filter and prediction horizon. The best FPR is emphasized in bold for each prediction horizon. Additionally, ρ values which optimized both negative and positive performance for the given filter and prediction horizon are highlighted.

Filter	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	FPR	ρ	FPR	ρ	FPR	ρ	FPR	ρ	FPR	ρ
Γ_1	0.066	0.8	0.0936	0.8	0.133	**0.6**	0.176	0.7	0.216	**0.7**
Γ_2	0.062	**0.6**	0.0886	**0.7**	0.125	0.6	0.165	**0.6**	0.211	**0.6**
Γ_3	0.062	0.9	0.0956	0.8	0.133	**0.7**	0.177	0.7	0.214	**0.6**
Γ_4	**0.056**	0.9	**0.0817**	0.8	**0.123**	0.6	**0.160**	**0.6**	**0.210**	**0.6**

Although, the model shows agreement on the optimal value of ρ for $H = \{1.75, 2\}$ in the terms of TPR and FPR performance, there is a discrepancy for other prediction horizons. A disagreement in value of ρ indicates that the choice of ρ represents a trade-off between positive and negative performance. To better assess this trade-off for Γ_4, the TPR and FPR can be plotted against the mean trig time $\bar{t}_h$ for the different values of ρ, which is shown in Fig. 4.10.

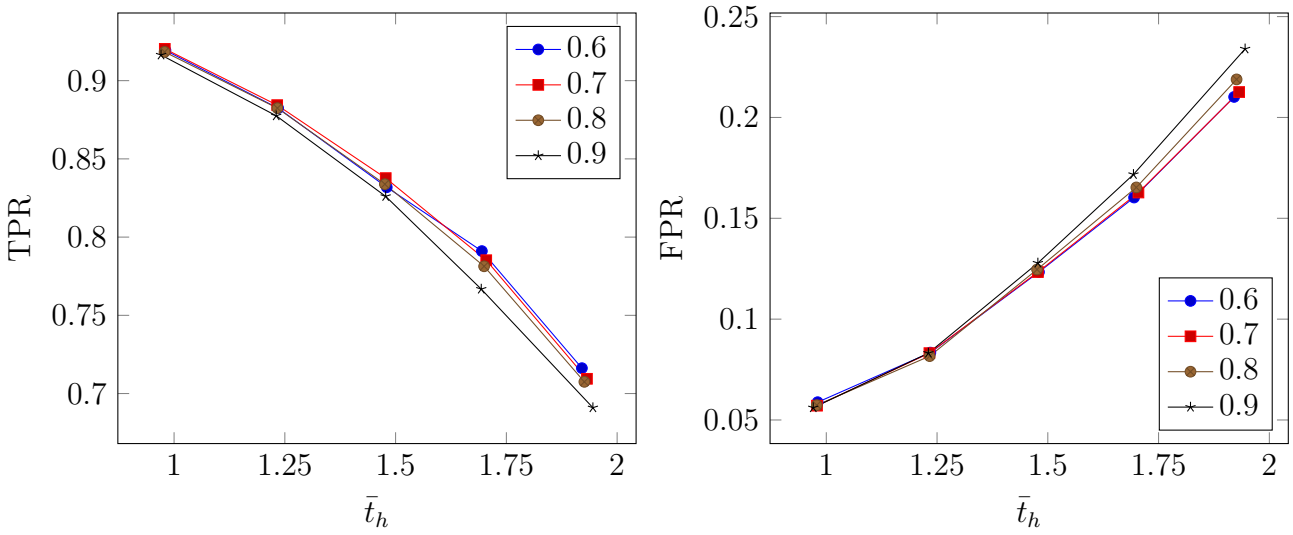


Figure 4.10: TA performance of GMLPE using Γ_4 filter for different values of the probability threshold ρ

From Fig. 4.10 the absolute value of ρ does not seem to significantly impact TA performance in the case of Γ_4. However, for $\rho = 0.9$ there is a performance decrease in terms of TPR and FPR for longer prediction horizons. For the longer prediction

horizons a smaller value of ρ seems preferable, in this case $\rho = 0.6$. While for the shorter prediction horizons, $H \leq 1.5$, $\rho = 0.7$ performs better. As previously shown in Fig. 4.2, the average total uncertainty is prediction horizon dependent, where longer prediction horizons show higher predictive uncertainty. The different values of ρ and the corresponding decision boundaries shown in Fig. 4.11 may be more suited for different average predictive uncertainties. This might explain the difference in the optimal value of ρ for different prediction horizons. These values of ρ will be used in future TA performance comparisons against other methods.

To better understand how the PD TA method works and how it differs from the deterministic TA method it is useful to plot the probability of departure against the predicted overlap and the predictive uncertainty. This is shown in Fig. 4.11.

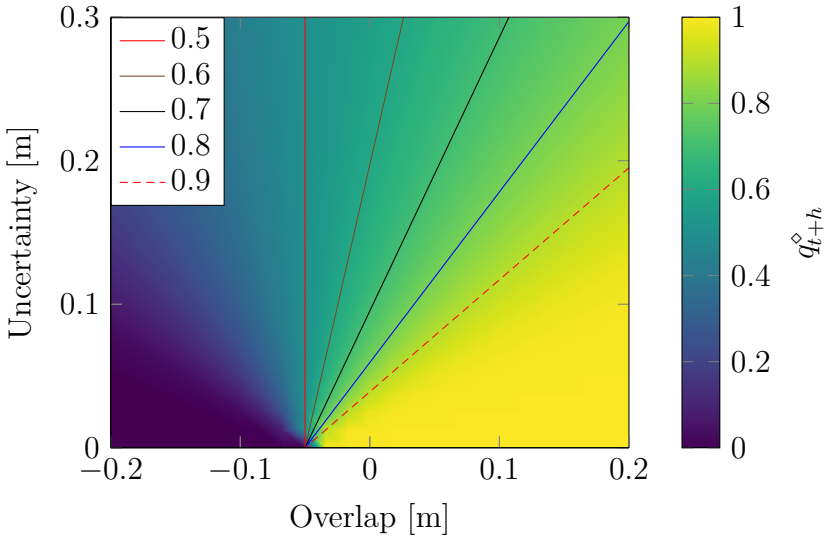


Figure 4.11: Probability of departure at time $t+h$ for a trig time tuned TA system with $\tau = -0.05$ based on the predicted overlap and estimated uncertainty. With intervention decision boundaries for different values of the probability threshold ρ. An intervention is triggered in the region to the right of the respective decision boundaries.

From Fig. 4.11 the decision boundaries for a triggered autonomous intervention form what appears to be straight lines. Where an intervention is triggered to the right of their respective line. Moreover, by adjusting the value of ρ the slope of the decision boundary can be altered. Specifically, by increasing ρ the decision boundary rotates clock-wise, and vice versa for decreasing values of ρ. With a slanted decision boundary, even small overlaps with the lane marker will result in a triggered intervention provided that the uncertainty is sufficiently low. Likewise, a large overlap can trigger an intervention even if the uncertainty is higher. Finally, by adjusting τ the decision boundaries can be shifted left or right, where a smaller value of τ will move the boundaries to the left and vice versa.

From this figure it is also evident that the PD method is equivalent to the deterministic TA method for $\rho = 0.5$. Since in this scenario the decision boundary becomes

vertical and therefore does not consider the predictive uncertainty for a potential autonomous intervention.

4.6.3 Threat Assessment Performance Comparison

In Fig. 4.12 the performance of the best TA methods are compared to each other and the baseline CVM. In this performance comparison the ML based TA models outperform the baseline CVM in both TPR and FPR for all evaluated prediction horizon. Moreover, the ML based models show similar TPR and FPR performance regardless of TA method for every prediction horizon. A more detailed TA performance comparison between the ML based models are shown in Table 4.6.

As previously shown, the predictive uncertainty, although well-calibrated, is mostly aleatoric which means that it does not provide much actionable information. Instead most of the predictive uncertainty is a result of irreducible noise. Additionally, due to the symmetric nature of the Gaussian distribution over the predicted lane markers, the predicted mean will always be the most probable location of the real lane marker. Thus the predictive performance of the uncertainty aware TA model will be limited by its MSE performance. Which might explain the similarity in TA performance for uncertainty aware and standard ML models.

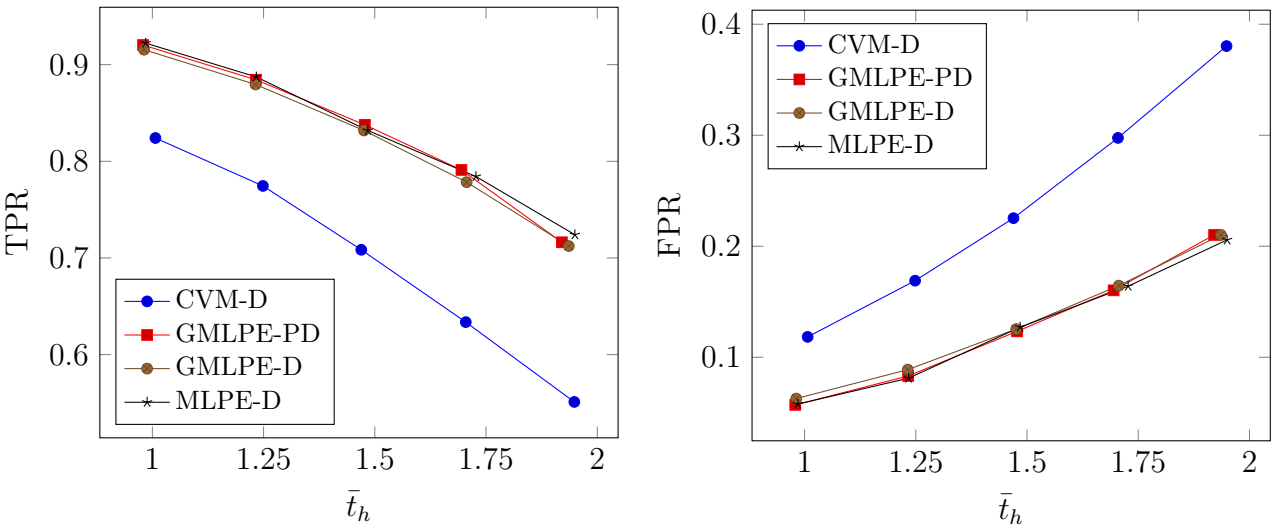


Figure 4.12: TA system performance comparison of the best ML TA models compared to the baseline CVM, where D indicates that the deterministic intervention criterion was used.

According to Table 4.6 the GMLPE-PD and MLPE-Deterministic showed the highest performance in general. With the best performing model varying with the tested prediction horizon. While the GMLPE-Deterministic model, which does not use an uncertainty aware intervention criterion although capable of producing predictive uncertainty estimates, showed worse performance.

In Fig. 4.13 two scatter plots of the predicted overlap and the prediction error, along with the total uncertainty are shown. The points shown in the scatter plots

Table 4.6: Comparison of TA performance of the best ML models, where D indicates that the deterministic intervention criterion was used. The best performance is highlighted in bold for each prediction horizon.

Method	$H = 1$		$H = 1.25$		$H = 1.5$		$H = 1.75$		$H = 2$	
	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR	TPR	FPR
GMLPE-PD	0.920	**0.057**	0.884	0.083	**0.838**	**0.123**	**0.791**	**0.160**	0.716	0.210
GMLPE-D	0.915	0.063	0.879	0.089	0.832	0.125	0.778	0.165	0.712	0.210
MLPE-D	**0.922**	0.058	**0.888**	**0.081**	0.832	0.127	0.784	0.164	**0.724**	**0.206**

are classified by TP or FP interventions from the GMLPE-Deterministic model with a prediction horizon of 1.5 s. According to the figure, TP and FP cases seem to share similar clustering characteristics. Indicating that the two cases can not be simply separated using the predictive uncertainty.

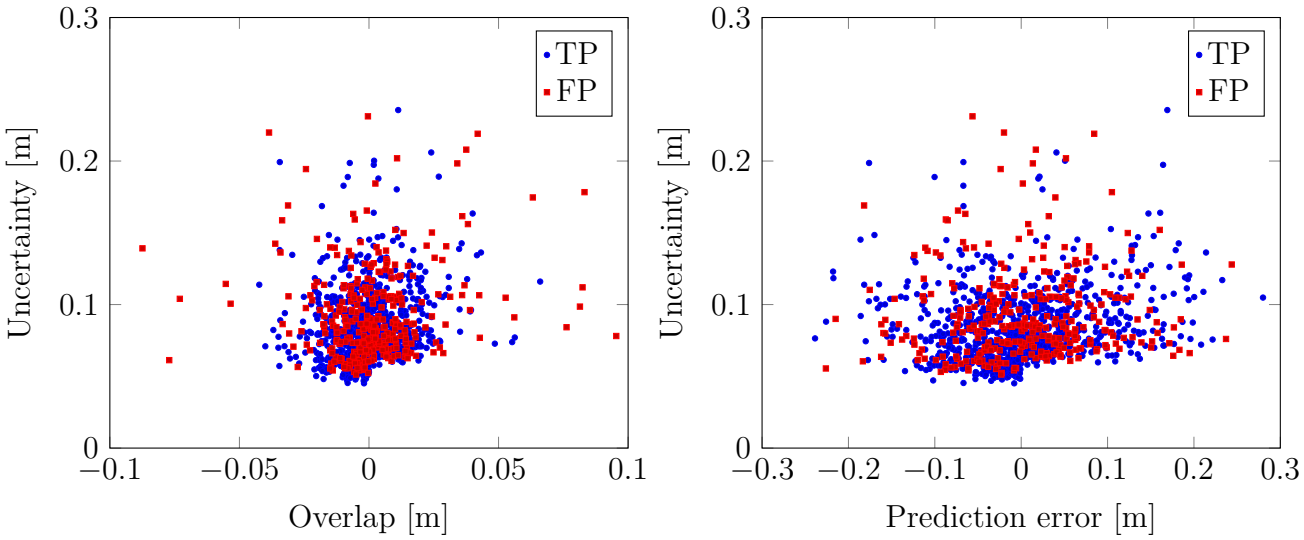


Figure 4.13: Scatter plot of TP/FP lane departure detections from the GMLPE-Deterministic model. The sign of the prediction error and overlap indicates for which side the prediction was made, where positive values are for the left side and negative for the right side.

4.7 Distilled Models

Ensemble distillation was made to investigate if it is possible to keep the ensemble performance using a single model. Being more efficient is desirable in an environment with limited computational power, as is the case for any vehicle today.

The distilled models was evaluated in terms of statistical performance (MSE), calibration and TA performance. For these evaluation metrics, the overall best performing filter configuration Γ_4 was used.

Table 4.7 shows the statistical performance of the distilled model, its ensemble correspondence and a single Gaussian MLP. The difference of these was expected to be small, as the single Gaussian MLPs and Gaussian MLP ensembles showed similar performance in Table 4.1. As expected, the differences between the three models were small, and the distilled model performed in line with the single model in this matter.

Table 4.7: Mean squared error for a distilled GMLP, its corresponding GMLP ensemble and a single GMLP using filter Γ_4.

Model	$H = 1$	$H = 1.25$	$H = 1.5$	$H = 1.75$	$H = 2$
GMLPD	0.004	0.006	0.011	0.017	0.025
GMLP	0.004	0.006	0.011	0.017	0.025
GMLPE	0.004	0.006	**0.010**	**0.016**	**0.024**

Figure 4.14 shows a reliability plot for the distilled model in the same setting (filter Γ_4 on $H = 2$ seconds) as the GMLP and GMLP ensemble in Fig. 4.5. The distilled model proved to be well calibrated as well.

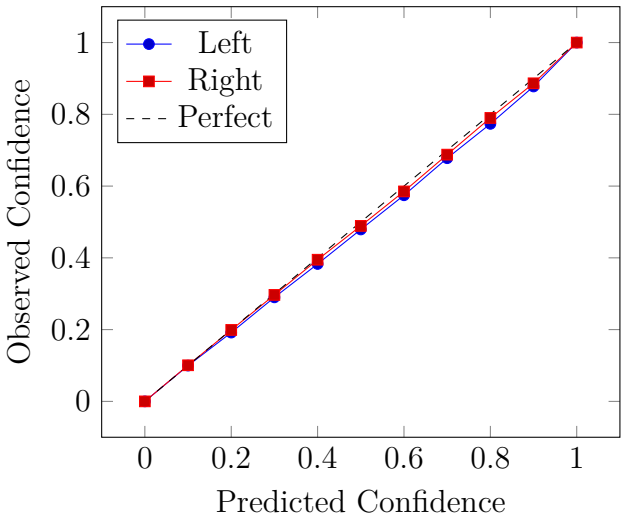


Figure 4.14: Reliability plot for a distilled Gaussian MLP at $H = 2$ seconds using filter Γ_4. The distillation process keeps the new model well calibrated.

Finally, the TA performance was evaluated in terms of positive and negative performance and is illustrated in Fig. 4.15. In general, the ensemble models seems to give slightly better performance in both TPR and FPR. This is most evident in the cases where the trig time tuning results in about the same trig time.

In conclusion, the distillation process [40] worked to successfully reduce a Gaussian MLP ensemble down to one single distilled Gaussian MLP that retains the uncertainty estimates. The distilled model proved to be well calibrated and give good statistical performance, even though a small performance loss can be seen for longer

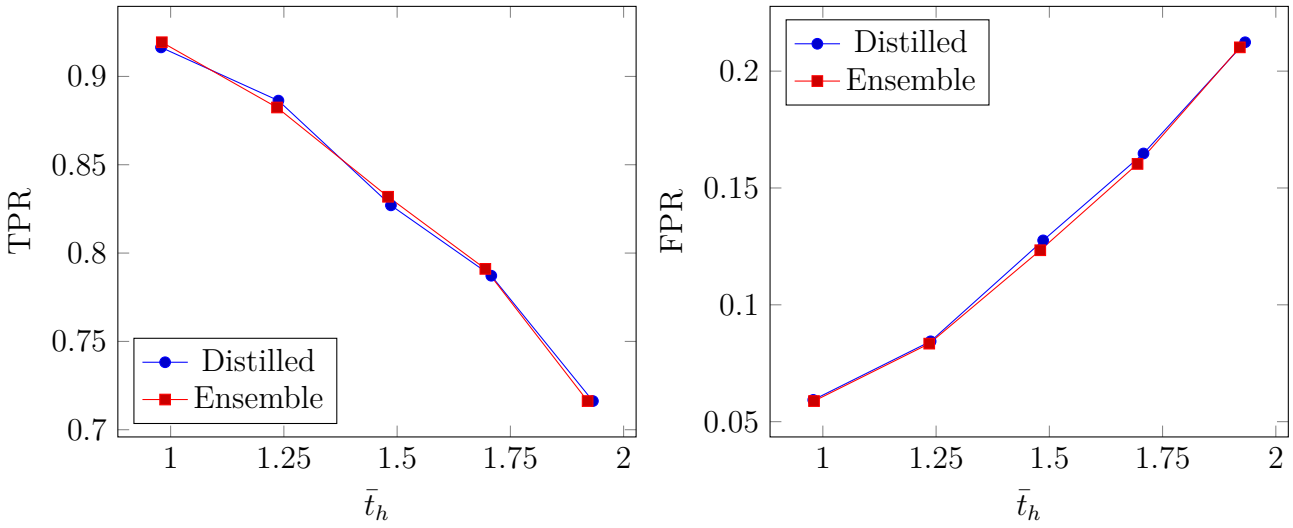


Figure 4.15: TA performance comparisons for distilled GMLP models and their GMLPE correspondences for Γ_4.

prediction horizons in terms of MSE. This is the case for TA performance as well, where the ensembles in general perform slightly better than the the distilled model.

5 Conclusions

The main contribution of this thesis is the implementation and evaluation of a GMLP and a GMLPE, in the context of predicting unintended lane departures. The predictive performance was benchmarked against a regular MLP and a kinematic prediction model, both in terms of statistical and application performance.

5.1 Statistical Performance

The statistical performance was evaluated using MSE, which corresponds to the expectation of the squared prediction errors. The results showed that a Gaussian MLP ensembles performed in line with standard MLPs ensembles. This, despite GMLPs being optimized using NLL loss function while the MLPs were optimized using the MSE loss function. GMLPEs also proved to give well-calibrated uncertainty estimates and were able to distinguish between epistemic and aleatoric uncertainty.

Using GMLPEs has advantages in statistical performance and uncertainty estimation, but may be computationally infeasible due to propagating through multiple networks. Ensemble distillation was performed to reduce the ensemble into one network capable of estimating both aleatoric and epistemic uncertainty. This proved to give well calibrated uncertainty estimates and performed in line with GMLPs in terms of MSE.

5.2 Threat Assessment Performance

In the application of a lane keeping assist system, the evaluated data-driven TA methods outperform a baseline kinematic CVM in terms of TPR and FPR performance. Moreover, the uncertainty aware GMLPE model using the PD criterion performed in line with a standard MLPE model. Consequently, this indicates that for the evaluated dataset, the predictive uncertainty estimate does not accurately separate TP and FP detections. This is likely caused by the fact that the uncertainty is mostly aleatoric and does not provide sufficient information for the decision making process. Instead, the system performance seems to be limited by its MSE performance.

5.3 Future Work

This work was limited to GMLPs as an uncertainty aware ML model. Other promising uncertainty aware ML models exist, some of which were highlighted in the related work section. Implementing some of these could perhaps provide new insights in using uncertainty estimates for TA.

Moreover, this thesis represents a proof of concept and more work would be required for real-world applications. Such considerations would be part of an industrialization

process. For example, the limitations imposed on the operational domain caused by the extraction process of the training data from the full dataset. As such, there is a possibility that some real world scenarios are not captured in the training data.

Bibliography

[1] J. Dahl, G. R. de Campos, C. Olsson, and J. Fredriksson, "Collision avoidance: A literature review on threat-assessment techniques", *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 1, pp. 101–113, 2019, ISSN: 2379-8858.

[2] R. van der Horst and J. Hogema, *Time-to-collision and collision avoidance systems*. na, 1994.

[3] J. Rawlings and D. Mayne, "Postface to model predictive control: Theory and design", *Nob Hill Pub*, vol. 5, pp. 155–158, 2012.

[4] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin, "Provably safe and robust learning-based model predictive control", *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[5] M. Althoff, "Reachability analysis and its application to the safety assessment of autonomous cars", PhD thesis, Technische Universität München, 2010.

[6] M. Althoff and A. Mergel, "Comparison of markov chain abstraction and monte carlo simulation for the safety assessment of autonomous cars", *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1237–1247, 2011.

[7] J. Jansson, "Collision avoidance theory: With application to automotive collision mitigation", PhD thesis, Linköping University Electronic Press, 2005, pp. 49–50.

[8] J. Dahl, G. R. de Campos, and J. Fredriksson, "A path prediction model based on multiple time series analysis tools used to detect unintended lane departures", 2020.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", in *IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[11] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need", in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[12] J. Dahl, R. Jonsson, A. Kollmats, G. R. de Campos, and J. Fredriksson, "Automotive safety: A neural network approach for lane departure detection using real world driving data", in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 3669–3674.

[13] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks", in *34th International Conference on Machine Learning - Volume 70*, JMLR.org, 2017, pp. 1321–1330.

[14] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles", in *Advances in Neural Information Processing Systems*, 2017, pp. 6402–6413.

[15] T. Bui, D. Hernández-Lobato, J. Hernandez-Lobato, Y. Li, and R. Turner, "Deep gaussian processes for regression using approximate expectation prop-

agation", in *International conference on machine learning*, 2016, pp. 1472–1481.

[16] S. Rogers and M. Girolami, *A first course in machine learning*. CRC Press, 2016, pp. 278–297.

[17] D. Hernández-Lobato, J. M. Hernández-Lobato, Y. Li, T. Bui, and R. E. Turner, "Stochastic expectation propagation for large scale gaussian process classification", *arXiv preprint arXiv:1511.03249*, 2015.

[18] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning", in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 1050–1059.

[19] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, "Weight uncertainty in neural network", in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1613–1622.

[20] T. Pearce, M. Zaki, A. Brintrup, and A. Neel, "Uncertainty in neural networks: Approximately bayesian ensembling", *arXiv preprint arXiv:1810.05546*, 2018.

[21] J. Moberg, L. Svensson, J. Pinto, and H. Wymeersch, "Bayesian Linear Regression on Deep Representations", no. NeurIPS, 2019. arXiv: `1912.06760`.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[23] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[24] N. S. Keskar, J. Nocedal, P. T. P. Tang, D. Mudigere, and M. Smelyanskiy, "On large-batch training for deep learning: Generalization gap and sharp minima", *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, pp. 1–16, 2019. arXiv: `arXiv:1609.04836v2`.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*, 2014.

[26] A. Krogh and J. A. Hertz, "A simple weight decay can improve generalization", in *Advances in neural information processing systems*, 1992, pp. 950–957.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[28] P. Falcone, M. Ali, and J. Sjoberg, "Predictive threat assessment via reachability analysis and set invariance theory", *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1352–1361, 2011.

[29] A. Der Kiureghian and O. Ditlevsen, "Aleatory or epistemic? does it matter?", *Structural safety*, vol. 31, no. 2, pp. 105–112, 2009.

[30] S. Depeweg, J.-M. Hernandez-Lobato, F. Doshi-Velez, and S. Udluft, "Decomposition of uncertainty in bayesian deep learning for efficient and risk-sensitive learning", in *Proceedings of the 35th International Conference on Machine Learning*, vol. 80, 2018, pp. 1184–1193.

[31] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[32] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation", *arXiv preprint arXiv:1506.02157*, 2015.

[33] G. W. Brier, "Verification of forecasts expressed in terms of probability", *Monthly weather review*, vol. 78, no. 1, pp. 1–3, 1950.

[34] V. Kuleshov, N. Fenner, and S. Ermon, "Accurate uncertainties for deep learning using calibrated regression", in *Proceedings of the 35th International Conference on Machine Learning*, 2018, pp. 2796–2804.

[35] J. Snoek, Y. Ovadia, E. Fertig, *et al.*, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift", in *Advances in Neural Information Processing Systems*, 2019, pp. 13 969–13 980.

[36] G. Keren, N. Cummins, and B. Schuller, "Calibrated prediction intervals for neural network regressors", *IEEE Access*, vol. 6, pp. 54 033–54 041, 2018.

[37] H. Song, T. Diethe, M. Kull, and P. Flach, "Distribution calibration for regression", *arXiv preprint arXiv:1905.06023*, 2019.

[38] M. Kull, T. Silva Filho, and P. Flach, "Beta calibration: A well-founded and easily implemented improvement on logistic calibration for binary classifiers", in *Artificial Intelligence and Statistics*, 2017, pp. 623–631.

[39] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression", in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

[40] J. Lindqvist, A. Olmin, F. Lindsten, and L. Svensson, "A general framework for ensemble distribution distillation", *arXiv preprint arXiv:2002.11531*, 2020.

[41] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. [Online]. Available: `https://www.tensorflow.org/`.

[42] T. Oliphant, *NumPy: A guide to NumPy*, USA: Trelgol Publishing, 2006. [Online]. Available: `http://www.numpy.org/`.

[43] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: `https://doi.org/10.1038/s41592-019-0686-2`.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: `10.5281/zenodo.3509134`. [Online]. Available: `https://doi.org/10.5281/zenodo.3509134`.