CHALMERS
UNIVERSITY OF TECHNOLOGY

Machine Learning for Lane Positioning in Autonomous Vehicles

Supervised Learning on High-Dimensional Data

Master's thesis in Complex Adaptive Systems

ANDERS HANSSON
RICHARD SUNDQVIST

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

This page intentionally left blank.

MASTER'S THESIS 2018

Machine Learning for Lane Positioning in Autonomous Vehicles

Supervised Learning on High-Dimensional Data

ANDERS HANSSON
RICHARD SUNDQVIST

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Machine Learning for Lane Positioning in Autonomous Vehicles
Supervised Learning on High-Dimensional Data
ANDERS HANSSON
RICHARD SUNDQVIST

© ANDERS HANSSON, 2018.
© RICHARD SUNDQVIST, 2018.

Supervisors: Tommy Tram, *Zenuity & Department of Electrical Engineering*
Hannes Marling, *Zenuity*
Examiner: Mats Granath, *Department of Physics*

Master's Thesis 2018
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: A Volvo XC90, Volvo Car Corporation (CC BY-NC 4.0)

Typeset in LaTeX
Compiled with pdfLaTeX via sharelatex.com
Gothenburg, Sweden 2018

Machine Learning for Lane Positioning in Autonomous Vehicles
Supervised Learning on High-Dimensional Data
ANDERS HANSSON
RICHARD SUNDQVIST
Department of Physics
Chalmers University of Technology

Abstract

The lateral position of a car in its current lane affects the perceived comfort and safety of passengers. The problem of selecting a comfortable lateral position for an autonomous car in traffic is not trivial. This thesis addresses this issue by breaking it down into two related problems to be solved, using Feed-Forward Neural Networks and Random Forests. The first problem, Barrier Detection (BD), is a perception problem. It involves identifying relevant variables for detecting barriers on the road side and distinguishing barriers from instances resulting in similar sensor stimuli such as traffic signs. The second problem, Lane Positioning (LP), is a Decision & Control (D&C) problem. The task is to select a suitable lateral position for the host vehicle within the current lane, given the immediate surroundings.

Furthermore, strict timing and memory requirements are in place. This necessitates pruning and preprocessing of high-dimensional raw data before the data is forwarded, and places constraints on the implementations. Both problems were approached with supervised learning methods, and the results for BD were promising with a test set accuracy of 80% using neural networks. The neural networks for LP failed, with the best neural network attaining only 60% accuracy.

By using a different classifier type, Random Forest (RF), an out-of-bag[1] error of 4% was achieved for the LP problem. However, sufficient performance was only reached for extreme lateral offsets and these classifiers grossly violated the memory constraints. Random lateral offsets in the data appeared to overshadow the offsets taken in response to significant scenarios. This could explain why the classifiers failed to identify trends for smaller offsets.

Keywords: autonomous cars, supervised learning, deep learning, neural networks, random forests, decision & control, perception

[1]**Out-of-bag error**: Estimate of predictive power for ensemble classifiers, see subsection 2.3.2.

This page intentionally left blank.

Acknowledgements

The authors of this Master thesis would like to thank Zenuity for being our hosts and for giving us the opportunity to solve a real industry problem, with all the frustrating little quirks that it entails. In particular we would like to thank Team Kalman and Team Turing at Zenuity for mentoring us and for enriching our experience at the company. Furthermore, we would like to thank our opponents Jonas Ericsson and Linus Arnö for their feedback.

We would like to thank our supervisors, Tommy Tram and Hannes Marling at Zenuity, for their guidance and for the many rewarding discussions (and the unrelated banter) we've enjoyed in the months that we've known them. Finally, we would like to thank our examiner Mats Granath at the Department of Physics at Chalmers, for his advice and for giving us freedom to steer the direction of the thesis ourselves. We would also like to thank him for the work he's done as the director of the Complex Adaptive Systems programme, which we have enjoyed immensely during our final years at Chalmers.

Anders Hansson, Gothenburg, June 2018
Richard Sundqvist, Gothenburg, June 2018

This page intentionally left blank.

Table of Contents

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network.

BD Barrier Detection.

CFS Correlation-based Feature Selection.

D&C Decision & Control.

EA Evolutionary Algorithm.

FFNN Feed-Forward Neural Network.

GD Gradient Descent.

LP Lane Positioning.

ReLU Rectified Linear Unit.
RF Random Forest.
RUS Random Undersampling.

SGD Stochastic Gradient Descent.

t-SNE t-Distributed Stochastic Neighbor Embedding.

Glossary

AdaGrad The *Adaptive gradient* algorithm, see [1].
Adam The *Adam* algorithm, see [2].

Ftrl The *Follow The (Proximally) Regularized Leader* algorithm, see [3].

RMSProp The *RMSProp* algorithm, see [4]. This algorithm is not accompanied by a published scientific paper, but was introduced as part of a COURSERA lecture.

1
Introduction

1.1 Levels of autonomy – incremental progress

The landscape of the automotive market is changing rapidly, with the development towards autonomous vehicles making steady progress. Most of the traditional car manufacturers are involved, as well as a large number of well-known tech companies [5], [6]. A majority of consumers, however, still state that they would be afraid to ride in a *fully* autonomous car today [7], [8]. Such vehicles, in which a steering wheel is essentially optional, are often referred to as having *level 4* [9] or *level 5* [10] automation, depending on the standard used.

There are no personal vehicles at this level of automation today, though several manufacturers offer systems with *partial* autonomy [11]–[14]. The car must have an alert and active driver, however in limited situations, the car can manage its own operation. These are commonly referred to as *Advanced Driver-Assistance Systems (ADAS)*. These systems can be seen as an intermediate step, and are being further developed to allow for even higher levels of automation in the future.

1.2 Background

Zenuity is a developer of ADAS software [15], and routinely performs test driving around Gothenburg as a part of the development effort. During several such sessions, developers noted that the prototype vehicle appeared to position itself closer to adjacent freighters than was intended. The on-board logs of the prototype vehicle later revealed that an appropriate distance had in fact been kept between the vehicles in question, indicating that the freighters simply made for more imposing figures than a personal vehicle at the same distance.

The discomfort of the prototype vehicle's passengers, however, was real. The effect is not limited to freighters; similar complaints were made about, for example, road barriers. A flat offset could be implemented, but the host vehicle may then end up crowding smaller vehicles such as motorcycles instead. This is an issue for manufacturers: An autonomous car must not only be physically safe, it must also *feel* safe, both to its passengers and to the surrounding traffic. The lateral position of the host vehicle within the current lane plays an important role in this endeavour.

Barriers also play a more subtle role in the positioning of the vehicle: Vision based perception algorithms usually rely on estimating the position of the road markers for determining the lateral position of the car [16]. To a black-and-white camera, the top of a barrier may appear as if it were a road marker. The barrier

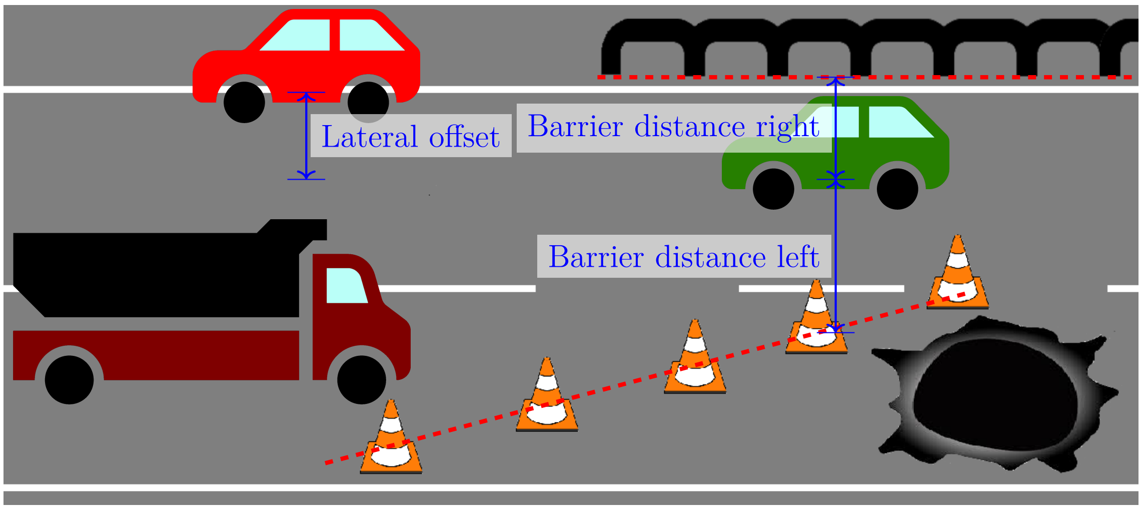

Figure 1.1: A possible scenario with three vehicles on a damaged road. The **red** car is avoiding a heavy freighter on its left side: The line shows how lateral offset is measured for the Lane Positioning (LP) problem. The **green** car has barriers on both sides: The lines show how lateral distance to barriers is measured for the Barrier Detection (BD) problem. *(Note that traffic cones are out of scope.)*

may also be placed inside the lane; this is very common during road work. Both of these cases may result in the vehicle positioning itself improperly in the lane.

The problem arises: How does one detect a scenario where a lateral offset from the centre of the lane is desirable? The vehicle must be able to detect that there is a relevant object, such as a barrier or a heavy vehicle, present in the first place. Once this is done, the vehicle can consider what it knows about its surroundings to make a decision.

Note that *choosing* a position for the car in the lane is a *Decision & Control* problem. This differs from the task of *estimating the current position* of the car in the lane, which is a *perception* problem, that has previously been solved using deep learning methods [17]. The detection of relevant objects such as barriers and freighters are perception problems as well.

1.3 Purpose

The purpose of this thesis is to develop a prototype system for detecting when a lateral offset is desired in an autonomous car. Solutions should, if possible, respect timing and memory constraints in place for an autonomous vehicle. The goal is not to create a finished product, but to explore possible solutions.

1.4 Problem formulations

In order to develop a lateral positioning system, perception algorithms for detecting objects and scenarios must be implemented. Zenuity can reliably detect road vehicles, but have had issues with reliably detecting barriers (see subsection 3.4.1

for details). The lateral positioning issue will thus be split into two separate problems: *Lane Positioning (LP)* and *Barrier Detection (BD)*. The motivation for this split is two-fold: a strong BD solution is needed regardless of the outcome for the LP problem, while perception algorithms for other known factors related to lateral positioning already exist. The formal definitions of the problems are as follows:

Lane Positioning *(Decision & Control problem)*: Given the local environment of the host vehicle, select a position in the current lane which minimizes the discomfort of the passengers and surrounding traffic. The key assumption is that mimicking human drivers will lead to comfortable behaviour.

Barrier Detection *(Perception problem)*: Given the local environment of the host vehicle, determine if there are any laterally adjacent barriers present. In this thesis barriers are defined as fences, guard rails, and concrete barriers[1].

Distances are measured using the ISO-8855[2] axis reference frame. Please refer to Figure 1.1 for an explanation of how they are defined for both problems.

1.5 Motivation for a machine learning approach

This study attempts to solve the problems defined in section 1.4 using machine learning methods, specifically supervised learning. The reasons are that both LP and BD are complex problems which depend on many interacting variables, and the best strategy in some scenarios may be unknown. Zenuity has provided access to recorded data of human driving in real-world traffic.

The complex nature of the problems and the fact that we have access to statistical data motivates a machine learning approach. Mimicking human drivers using supervised learning methods avoids the issue of having to mathematically define what it means for something to be "comfortable".

[1]Temporary barriers, such as traffic cones, are out of scope.

[2]**ISO-8855**: Coordinate system standard commonly used for road vehicles. The origin is fixed in the middle of the plane which spans the wheels.

2
Theory

This section aims to provide the reader with a brief theoretical background related to the upcoming methods and results in this thesis.

2.1 Classification and regression

In machine learning and statistics, two central problems are classification and regression. Classification is the problem of predicting the associated class or category of new observations [18].

Classification

Given: n observations $\left(\vec{X}^1, \dots, \vec{X}^n\right)$ with associated labels $(Y^1, \dots, Y^n)$.

$\forall i : \vec{X}^i \in \mathbb{R}^p, Y^i \in \mathbb{N}_k^* = \{1, 2, \dots, k\}$, where k is the number of classes.

Goal: Predict the label Y^i for observation $\vec{X}^i$.

A *classifier* is an algorithm that (approximately) solves the classification problem.

Regression

Given: n observations $\left(\vec{X}^1, \dots, \vec{X}^n\right)$ associated with target values $(Y^1, \dots, Y^n)$.

$\forall i : \vec{X}^i \in \mathbb{R}^p, Y^i \in \mathbb{R}$

Goal: Predict the values $\vec{Y}^i$ such that the distance from the true vales to the predicted values is minimized [19]. This is essentially function fitting.

A *regressor* is an algorithm that (approximately) solves the regression problem.

The main difference is that regression deals with ordinal target labels and classification deals with discrete and nominal target labels.

2.2 Artificial neural networks

The Artificial Neural Network (ANN) computational model is a class of mathematical models which can be used as classifiers or regressors. More specifically an ANN is a directed graph with weighted edges and the nodes are computational units called neurons. ANNs are inspired by biological neural networks found in brains, hence the name [20]. In this thesis neural networks will always refers to ANNs, not biological neural networks.

Each neuron in an ANN is associated with two scalar values called activation and local field. The local field is a weighted sum of all inputs to the corresponding neuron. The activation of a neuron is the neural output and it is computed by using the local field as an input variable to an activation function. The activation function is a parameter of the network model. See section A.1 for some examples of activation functions. For further reading about ANNs, see [21].

2.2.1 Feed-forward neural network

A Feed-Forward Neural Network (FFNN) is a directed acyclic graph of neurons; a structure of topologically ordered layers of neurons. The activation of neurons in one layer will serve as input to the next layer [20]. The activation of the last layer is the network output. Layers between the network inputs and the last layer (output layer) are referred to as hidden layers. The activation a_i^l of the neuron with index i in layer l are given by Equation 2.1. Note that the superscript is not an exponent, but the layer index.

$$a_i^l = \sigma^l \left(\sum_j w_{ij}^l a_j^{l-1} + b_i^l \right) = \sigma^l(z_i^l) \tag{2.1}$$

σ^l is the activation function for layer l (assuming all neurons in one layer has the same activation function). w is the weight tensor, an element w_{ij}^l is the weight between neuron i in layer l and neuron j in layer $l-1$. b is the bias matrix, an element b_i^l is the bias of neuron i in layer l. z_i^l is the local field (input) to neuron i in layer l [21].

The weights and biases are parameters that are tuned in a process referred to as training. In supervised learning one must define a cost function C (sometimes called a loss function), which measures the error of the network output based on the target values and the actual output. The cost function is optimized with gradient descent with respect to the weights and biases [21]. A visualization of the neural networks used in this thesis can be seen in Figure 3.4.

2.2.2 Optimization of neural networks

Neural networks are usually trained by optimizing a defined loss function and performing gradient descent, though other methods such as an Evolutionary Algorithm (EA) can be used (see section 2.6). The update rule with gradient descent is:

$$w^l \to w^l - \eta \nabla_w^l C \qquad b^l \to b^l - \eta \nabla_b^l C \tag{2.2}$$

η is a parameter called the learning rate. The derivatives are computed with an algorithm called backpropagation, see subsection 2.2.3.

Using Gradient Descent with respect to the entire training data set at each iteration is called *batch training*. Batch training takes a long time and is usually not practical when dealing with large data sets. An alternative is Stochastic Gradient Descent (SGD). In SGD, only one instance of training data is considered at each iteration with random sampling [21]. *Mini-batch training* is a combination of both methods, which randomly samples a subset of the training data at each iteration. The optimization methods used in this thesis are based on this idea [1]–[4].

2.2.3 Backpropagation

Backpropagation is an algorithm for computing the partial derivatives needed when training a FFNN with gradient descent [21], see Algorithm 1. Backpropagation for a FFNN can be summarized with the following four equations:

$$\boxed{\begin{aligned} \delta^L &= \nabla_a^L C \odot \partial\sigma^L(z^L) && \text{(2.3a)} \\ \delta^l &= \left((w^{l+1})^T \delta^{l+1}\right) \odot \partial\sigma^l(z^l), \forall l \neq L && \text{(2.3b)} \\ \nabla_b^l C &= \delta^l && \text{(2.3c)} \\ \nabla_w^l C &= a^{l-1}(\delta^l)^T && \text{(2.3d)} \end{aligned}}$$

C is the cost function, L is the index of the final layer, δ^l is the error vector for layer l, $\partial\sigma^l(z^l)$ is the derivative of activation functions in layer l, and the $\odot$ symbol denotes the Hadamard product (elementwise multiplication). See section A.3 for derivation of these equations. Different termination criteria can be used in Algorithm 1. A

Algorithm 1 Backpropagation with gradient descent for a FFNN

```
 1: while not Termination Criteria do
 2:     Feed FFNN with training input
 3:     for $l = 1, \dots L$ do
 4:         Compute $a^l = \sigma^l(z^l)$                          ▷ Equation 2.1
 5:     end for
 6:     Use output $a^L$ to compute cost $C$
 7:     Compute output error $\delta^L$                          ▷ Equation 2.3a
 8:     for $l = L-1, \dots, 2, 1$ do
 9:         Compute $\delta^l$                                   ▷ Equation 2.3b
10:     end for
11:     Compute derivatives                     ▷ Equation 2.3c & Equation 2.3d
12:     Update weights and biases                                ▷ Equation 2.2
13: end while
```

common method is *holdout validation*. The data is split into two sets, the training set and validation set. The neural network does not train on the validation set, but the validation error is measured during training. In order to prevent overfitting on the training set, training is halted when the validation error stops decreasing [21].

2.2.4 Softmax loss function

The Softmax function ζ can be used as the activation function for the final layer of a classifier neural network. The Softmax (Equation 2.4) squashes the local fields of the final layer into probabilities, similar to the Boltzmann distribution [22], [23]. The output y is the estimated probabilities of fed input belonging to the respective classes.

$$\zeta\left(z^L\right) = \frac{1}{\sum_{j=1}^K e^{z_j^L}} \begin{pmatrix} e^{z_1^L} \\ \vdots \\ e^{z_K^L} \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = y \tag{2.4}$$

K denotes the number of classes and z^L is a vector containing the activations of all neurons in output layer. $t = \left(t_1 \dots t_K\right)^T$ is a vector containing the true probabilities of the label. A data instance can only belong to one class, which means that t follows a categorical distribution.

$$\exists i \in \{1, \dots, K\} \text{ such } \forall j \neq i : \{t_j = 0\} \text{ and } t_i = 1$$

The derivative of the Softmax function (which is needed for backpropagation, see subsection 2.2.2), is given by Equation 2.5 [22], [23].

$$\frac{\partial y_i}{\partial z_j^L} = y_j \delta_{ij} - y_i y_j \tag{2.5}$$

where δ_{ij} is the Kronecker delta.

A loss function related to the Softmax can be derived with a maximum log-likelihood estimation. The loss function for one data instance is defined by Equation 2.6.

$$H(t, y) = -\sum_{i=1}^{K} t_i \log\left(y_i\right) \tag{2.6}$$

Since t_i are the true probabilities and y_i the estimated probabilities, H is a cross-entropy function. For this reason, it is often called the *Softmax cross-entropy function.*

2.2.5 Adaptive learning rate

An adaptive learning rate refers a dynamically adjustable η-value (see Equation 2.2) [24]. How exactly this is done varies, but some common strategies are to begin with a large η to quickly find a minimum, then reducing it to find the exact value. Two optimization methods which use strategies similar to this are RMSProp [4] and Adam [2]. The learning rate η may also be increased in order to escape local minima. Another optimation method, AdaGrad [1], uses different values for different features depending on properties discovered during training.

2.3 Random Forests

Random Forest (RF) *("random decision trees")* is an ensemble learning method, first proposed by Ho in 1995 [25]. The basic idea is to combine multiple *decision trees* to produce a more accurate estimator. Several decision trees are created, and the final value is then decided by a vote (for classification) or by taking the average (for regression). The available hyperparameters vary, but parameters such as the number of trees to create and the number of children per node are generally tunable. They are known for being both accurate and easy to use, as they construct themselves, but offer less control compared methods such as FFNNs.

A decision tree is a cascading sequence of decisions, which can perform both regression and classification. It works by answering a series of questions about various properties to reach a decision. The same property can occur more than once,

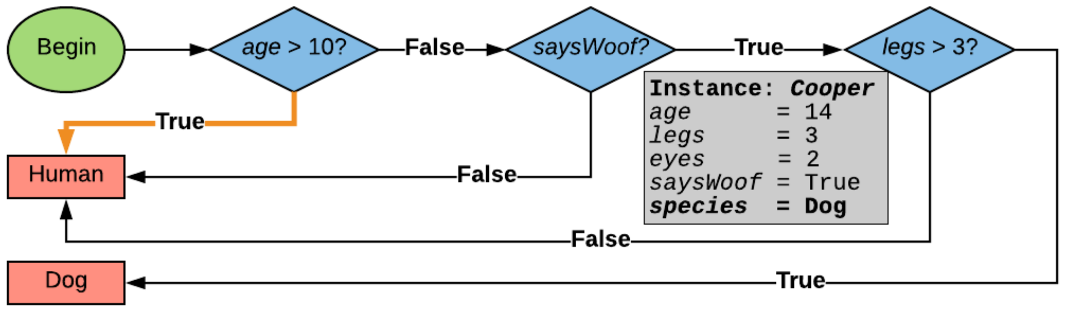

Figure 2.1: A binary decision tree which incorrectly classifies an old, three-legged dog as a human. Ensemble learning corrects for weaknesses of individual trees.

and the tree does not have to use all of the properties available to it. Figure 2.1 shows a decision tree which has incorrectly learned that a dog cannot be older than 10 years old, causing it to classify "Cooper" as a human. Individual trees tend to overfit to their training data [26], and one of the benefits of the RF method is that it avoids overfitting by ensemble learning [27].

2.3.1 Ensemble learning

Random Forests use bootstrap aggregating *("bagging")* [27], [28]. Bootstrap aggregating means that given a data set, new data sets are generated by sampling from the original set. Each new set is referred to as a bag which are true subsets of the original training set. When training the RF, the decision trees are trained on different bags. Each tree is trained on a single bag. For further reading about how individual decision trees are trained, see [29], [30].

2.3.2 Out-of-bag error

The *out-of-bag error* (sometimes *out-of-bag estimate*) is a method commonly used to measure the accuracy of ensemble methods such as Random Forests. The technique works by feeding decision trees with data from a bag which was not used to train them. This gives a measurement of the accuracy, but sometimes underestimates the actual performance [31], [32]. All data is used for training, but the test data and the training data is always different for any given decision tree.

2.4 t-Distributed Stochastic Neighbor Embedding

The t-Distributed Stochastic Neighbor Embedding (t-SNE) method was introduced by Maaten and Hinton in 2008 [33]. It is used to project high-dimensional data on 2d or 3d-space. It differs from some older techniques such as Principal Component Analysis (PCA), which attempts to minimize the sums of Euclidean distances [34]. Conversely, the t-SNE method uses pairwise *probabilistic* distances (Equation 2.8) and attempts to preserve the high-dimensional structure by keeping points which are close together in the high-dimensional structure close together in the low-dimensional projection. This is sometimes beneficial when regarding high-dimensional with complex local structures. In practice, t-SNE and PCA can be combined to speed up computations for high dimensional problems [33].

2.4.1 The mathematics of t-SNE

Given a set of N high dimensional points $\vec{x_i}, i = 1, \dots N$, distance metric d and a desired perplexity, t-SNE starts with computing the pairwise similarities using a gaussian kernel, Equation 2.7 and Equation 2.8. The kernel variances σ_i are parameters for each point that is set in order to attain a desired perplexity.

$$p_{i|j} = \exp\left(\frac{-d(\vec{x_i}, \vec{x_j})^2}{2\sigma_i^2}\right)\left(\sum_{k \neq i} \exp\left(\frac{-d(\vec{x_i}, \vec{x_k})^2}{2\sigma_i^2}\right)\right)^{-1} \tag{2.7}$$

$$p_{i|i} = 0$$

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2N} \tag{2.8}$$

The low dimensional similarities q_{ij} are computed with a t-Student kernel.

$$q_{ij} = \exp\left(1 + ||\vec{y_i} - \vec{y_j}||^2\right)^{-1}\left(\sum_{k \neq l}(1 + ||\vec{y_k} - \vec{y_l}||)^{-1}\right)^{-1} \tag{2.9}$$

The low-dimensional counterparts $\vec{y_i}$ of $\vec{x_i}$ are computed by minimizing the Kullback-Leibler divergence between the joints distributions P and Q, Equation 2.10 [35].

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right) \tag{2.10}$$

2.5 Feature selection

A *feature* is a variable to use as input to a machine learning algorithm. The feature itself can be of any type, such as an integer, Boolean or decimal value. *Feature selection* is the process of selecting a minimum subset of features in order to minimize computational load while maintaining high performance with the predictor algorithm. A standing hypothesis is that the best subset has a high class-dependency and low dependency among the selected features. Mathematically formulated, feature selection is selecting a feature subset S_{max} such that the sum in Equation 2.11 is maximized [36]. Note that there are alternative measures of the merit of a feature subset, such as Equation 2.13.

$$S_{\max} = \underset{S}{\operatorname{argmax}}\left(\frac{1}{|S|}\sum_{X^i \in S} I(X^i, Y) - \frac{1}{|S|^2}\sum_{X^i, X^j \in S} I(X^i, X^j)\right) \tag{2.11}$$

Where $|S|$ is the cardinality of subset $S = \{X^1, \dots, X^{|S|}\}$, and I is a dependence measure. Y denotes the response variable. In order to find relevant features for estimating the predictor variable, one must find a way to measure relations between variables. In other words, one must define a dependence measure.

2.5.1 Dependence measures

A dependence measure $I(X, Y)$ between two stochastic variables X, Y is a measure of statistical relationship. An ideal dependence measure should satisfy the following axioms [37].

(i) $I(X, Y)$ should be defined for both continuous and discrete variables.
(ii) I should be normalized, $I(X, Y) \in [0, 1]$
(iii) $I(X, Y) = 0$ iff X and Y are independent.
(iv) $I(X, Y) = 1$ iff there is a deterministic relation $f(X) = Y$.
(v) $I(X, Y)$ is invariant under any strictly increasing transformation.
(vi) $I(X, Y)$ is a metric, it measures the *"distance to a deterministic dependence"*. If X, Y are neither independent nor related by a deterministic function, the dependence measure lies in $(0, 1)$.

Pearson correlation coefficient

The Pearson correlation coefficient is a measure of linear dependence between two stochastic variables [38] X and Y, and is defined by Equation 2.12.

$$\rho(X, Y) = \frac{\text{Cov}[X, Y]}{\sqrt{\text{Var}[X]\text{Var}[Y]}} = \frac{\text{E}\left[(X - \mu_x)(Y - \mu_y)\right]}{\sigma_X \sigma_Y} \tag{2.12}$$

Pearson correlation is also a measure of negative linear correlation, $\rho(X, Y) \in [-1, 1]$. The Pearson correlation coefficient can be used a dependence measure by taking the absolute value $|\rho(X, Y)|$, since a negative correlation also indicates a relation.

2.5.2 Correlation-based Feature Selection

The Correlation-based Feature Selection (CFS) method was introduced by Mark A. Hall in a 1999 dissertation [39]. The working hypothesis of the paper is clearly stated by the author:

> *A good feature subset is one that contains features highly correlated with (predictive of) the class, yet uncorrelated with (not predictive of) each other.*

A heuristic measure of the *goodness* of a feature subset is used to accomplish this. Features are added to an initial empty set (or removed, if the initial set is all available features) such that the score, given by Equation 2.13, is maximized. The optimization can be carried out with heuristic search such as evolutionary algorithms, best first search and greedy hill climbing [39].

$$\text{score} = \frac{k\bar{r}_{cf}}{\sqrt{k + k(k-1)\bar{r}_{ff}}} \tag{2.13}$$

Where k is the number of chosen features in the subset, $\bar{r}_{cf}$ is the average class-feature correlation, and $\bar{r}_{ff}$ is the average feature-feature correlation. Different correlation metrics may be used.

The method may fail when the hypothesis is violated, though there is some tolerance and failure isn't immediate. Once again, in the words of the author:

CFS may fail to select relevant features, however, when data contains strongly interacting features or features with values predictive of a small area of the instance space.

For real-world data, the hypothesis often holds reasonably well. Feature interaction to the degree required to make CFS fail is primarily found in carefully constructed artificial data such as the MONK's problems, see citation [40].

2.5.3 ReliefF

Relief is a feature selection algorithm for binary classification, originally proposed by Kira and Rendell in 1992 [41]. Relief ranks each feature depending on how well it can distinguish between closely placed samples. ReliefF is an extension of Relief for multiclass classification [42]. ReliefF randomly selects a sample and utilizes its k-nearest neighbours. The quality of each feature depends on how many of the k-nearest neighbours belong to the same class, measuring the difference in the respective feature. Read more about Relief and ReliefF in [43]. The ranks computed by Relief or ReliefF can be used as ad-hoc dependence measures.

2.6 Evolutionary Algorithms

An Evolutionary Algorithm (EA) is a stochastic optimization method inspired by Darwinian evolution. It works by treating vectors of variables as biological *individuals* [21]. The *fitness* of an individual is measured by a *fitness function*, which must be designed for the specific problem being solved. Individuals with a high fitness are selected for *reproduction* using a *random* (this is what makes it a stochastic method) *selection* process, combining the variables of several individuals in a *reproduction* process to produce "children", which form the next *generation* (iteration). A few of the best individual are often preserved the next generation as well: this is called *elitism*. EAs have been used for a wide range of tasks, including the optimization of ANNs. For more information about EAs, see [21].

2.7 Random Undersampling

Random Undersampling (RUS) is a common technique used to prevent bias in estimators when the underlying data has an overwhelmingly *dominant* class, whose members greatly outnumbers other classes. The technique is simple: An appropriate amount of samples are selected at random and removed from the dominant class. This has been shown to increase accuracy [44], [45].

3
Methods

This thesis is split into two main problems: Barrier Detection (BD) and Lane Positioning (LP). Zenuity provided recorded data from cars driven in real world traffic. Given that data with recorded outcomes was available, the problems were approached as classification tasks using supervised learning. Data logs, in the form of *time series*, were data mined with relevant scenarios in order to create data sets.

In order to train a classifier on the data sets, the data had to be labelled with categories using various classification schemes. The data was labelled based on the available signals. This worked well for LP, giving that the position which was then discretized into intervals as a simplification. No strong ground truth labels for BD were available, so a proprietary algorithm provided by Zenuity (see subsection 3.4.1) was used instead. This algorithm was not designed to produce machine learning labels and sometimes fails to identify barriers. The output of this algorithm was used as labels after being converted to a boolean value and improved somewhat using a heuristic postprocessing algorithm.

After labelling the data, it was noted that the class distribution was highly skewed (which was expected). In order to remove learning biases towards dominant classes, Random Undersampling (RUS) was used to handle the imbalances.

The data available had over 6000 variables and thus required dimensionality reduction. The available variables were not raw sensor readings, but state estimations from various processing algorithms. These state estimations includes information about the local surroundings such as nearby cars, road signs, lane markers, road geometry etc. Relevant input variables were selected and identified using correlation-based methods. Data samples in the feature space were visualized using the t-SNE algorithm to visualize the selected features.

Feed-Forward Neural Network and Random Forest (RF) were used as the classifier models, with time steps treated independently. Various hyperparameters, such as activation functions and network structure, were tested to find the strong neural network candidates. For RF, the number of decision trees was the only hyperparameter considered.

3.1 Data

Zenuity provided access to a database containing measurements from car expeditions worldwide. The data was stored in log files, in the form of *time series*. Each step in a time series contained recorded signals from real-world driving in traffic. The data used was recorded in various conditions and locations, with a focus on right-hand European traffic. Most of the data was recorded during test driving in Gothenburg highway traffic, along a route commonly referred to as the *"Volvo Drive Me Gothenburg route"* (see [46]).

3.1.1 Data mining

An excessive amount of data was available, most of which was irrelevant for the given tasks. In order to create a suitable data set, time series that fulfilled a certain set of conditions were selected.

Here are the conditions that each time series needed to satisfy in order to be included in the data set. Note that slightly different conditions were used for finding data for LP and BD.

- **No outdated log files**
- **Only driving on a freeway or a carriageway**
- **Additional conditions**: Lane Positioning
  - *A log file must not contain any of the following:*
    - Lane changes
  - *A log file must contain any of the following:*
    - Horizontal curvature (road turns)
    - Motorcycles
    - Trucks
    - Road barriers
- **Additional conditions**: Barrier Detection
  - *A log file must contain the following:*
    - Road barriers

3.2 Classification schemes

In order to approach both BD and LP with supervised learning, we must formulate them as classification or regression problems. BD was split into two binary classification tasks, one for each side of the host car. The lateral position is a continuous signal but instead of using regression for LP, the lateral position was discretized in order to be solved as a classification problem. The reason is to minimize the noise, as an autonomous car which makes sudden and sharp adjustments with the steering wheel will discomfort and possibly scare its passengers.

3.2.1 Lane Positioning schemes

Classification schemes considered as labels for the LP problem:

- **Lane position with k intervals** After discretizing the lateral space in the lane into k intervals of equals size, classify which interval the host car should be positioned in.
- **Lane position switch with k intervals** After discretizing the lateral space in the lane into k intervals of equals size, classify when the host car should switch interval and when it should stay in the current interval.

- **Discretized lateral offset** This identical to *lane position with* 3 *intervals*, with the exception that the centre interval has a width $2d$ not dependent on the lane width[1].

See subsection 3.3.1 for a detailed description on how the LP labels were created.

3.2.2 Barrier Detection schemes

Classification schemes considered as labels for the BD problem:

- **Barrier detection (Left)** Classify if there is a barrier or not within 1.5 meters to the left of the host vehicle.
- **Barrier detection (Right)** Classify if there is a barrier or not within 1.5 meters to the right of the host vehicle.

See subsection 3.4.2 for a detailed description on how the BD labels were created.

3.3 Lane Positioning

Lane Positioning (LP) was approached as a multiclass classification problem. In order to reduce noise, the position in the lane was transformed from a continuous signal to a discrete one by dividing the lane into intervals. A few different approaches to creating labels were tested, described in section 3.2. This problem differs from Barrier Detection, in that there are additional restrictions on the signals that can be used. Signals such as blinker activation generally suggest a lane change or a turn, not a simple adjustment within the current lane. The classifiers should learn to make decisions to select where car *should be positioned* in the lane, not to estimate where it *is positioned*. The assumption was made that by mimicking human drivers, the classifiers will learn to select comfortable offsets in relevant scenarios.

3.3.1 Labels

The classification scheme *lane position with k intervals*, described in section 3.2, was computed by discretizing the lateral position relative to the lane width into k positions. See Figure 3.1 for an example with $k = 3$ intervals.

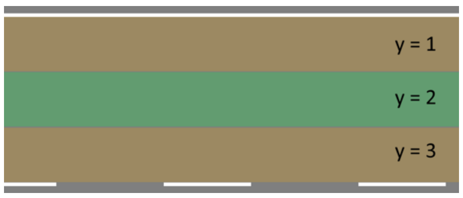

Figure 3.1: Discretization of the lateral position relative to the lane into $k = 3$ intervals. For instance a car with no lateral offset will have a lane position $y = 2$. The intervals have an equal size which it depends on the width of the lane.

[1]The width of the inner interval is fixed for any given value of the parameter d, but the width of the outer intervals varies depending in the width of the lane.

The *discretized lateral offset* scheme is identical to the *lane position with 3 intervals* scheme, with the exception that the width of the centre interval is a fixed parameter (typically 10-30 cm, as opposed to being dependent on lane width). The motivation behind this classification scheme was to detect smaller offsets without introducing too many classes.

3.3.1.1 Eager and delay labelling heuristics

The classification scheme *lane position switch with k intervals* was implemented using *eager labelling*. The reason for this is because otherwise, only the exact sample when the host car crosses the border between two intervals will count as a position switch. A relevant event, such as an overtake, which triggers an offset will cover many samples due to the sampling frequency. A *delayed* effect was also added, where samples after the switch were labelled as a position switch, see Figure 3.2. There is a risk of introducing more noise when choosing large delay or eager values, as the duration of the events are unknown.

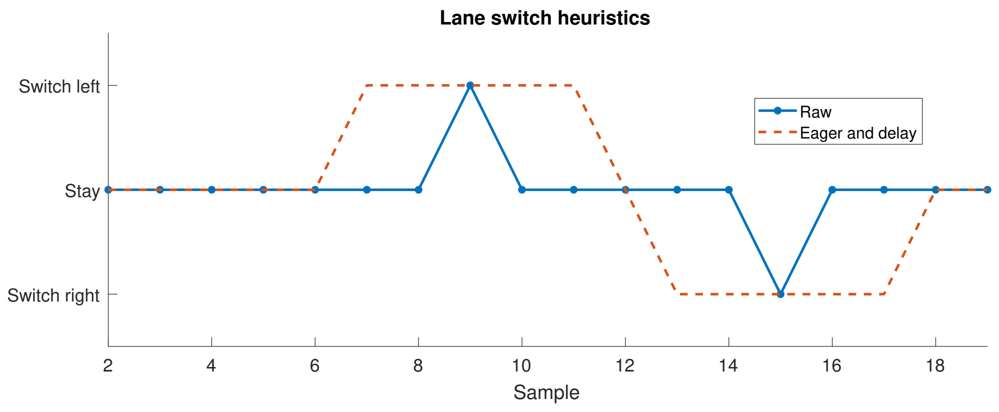

Figure 3.2: A hand-craft, graphical demonstration of the difference between the raw labels (lane position switch with k intervals) and with the eager and delayed labelling.

3.4 Barrier Detection

Barrier Detection (BD) was approached with *binary* classification, as described previously in section 3.2. The task was simplified into estimating if there was a barrier within 1.5 meters (lateral distance) of the car[2]. Two classifiers are created, one for the left side and one for the right. A major simplification was to treat each time step independently, and to detect barriers at each step instead of remembering previous states.

[2]Barriers are detected by the classifier when they are directly lateral to the host, see Figure 1.1.

3.4.1 Proprietary algorithm for Barrier Detection

Zenuity has a proprietary detection algorithm for detecting barriers, designed with following properties:

- Low rate of false positives
- Low memory consumption and fast execution
- Accurate estimate of the position of the barrier

The algorithm is based on a set of conditions and a quality counter with a minimum and a maximum value. The quality counter starts at its minimum value. In each sample, the quality counter increases or decreases depending on whether the conditions are fulfilled. Once the maximum value is reached, the sensor input used to decide whether to increment or decrement the quality counter is processed to calculate the position of the barrier.

Availability and accuracy had been issues with this approach, as well as the detection of barriers with poor radar reflection properties. The lack availability manifests as false negatives (failure to detect a barrier) when converting to boolean labels. Motivated by the lack of availability, it was decided that a machine learning solution would be attempted for the Barrier Detection problem instead. This algorithm will be referred to as the *proprietary algorithm.* These labels were improved as described in subsection 3.4.2.

3.4.2 Labels

The proprietary algorithm was used to create class labels for BD. A sample is labelled as *barrier detected* when the quality counter (recall the description in subsection 3.4.1) is at its maximum value. This happens when a barrier is less than 1.5m from the host vehicle. A sample is labelled as *no barrier detected* otherwise. Recall that this ground truth is not always accurate.

3.4.2.1 Eager labelling heuristic

The algorithm described in subsection 3.4.1 was designed to be used in real-time and not to immediately react to potential barriers, however we are using prerecorded data and thus *know the future.* With this in mind, we can improve the labels using this simple rule:

> *If the quality counter reaches the maximum value, assume there is a barrier present in all steps between where the maximum value was reached and the step where it started to increase.*

This will be referred to as *eager labelling.* Eager labelling is beneficial as it makes the labels more consistent: A barrier looks the same to the network whether the labelling algorithm has just spotted it, when the raw label still says there there is no barrier, and when it has been following the same barrier for a while. A second issue is that the proprietary algorithm may erroneously "drop" the barrier if the sensor input is disrupted by a spike in noise. Eager labelling handles both of these cases to some degree. Refer to Figure 3.3 for a graphical view of the difference between the quality counter, raw labels, and eager labels.

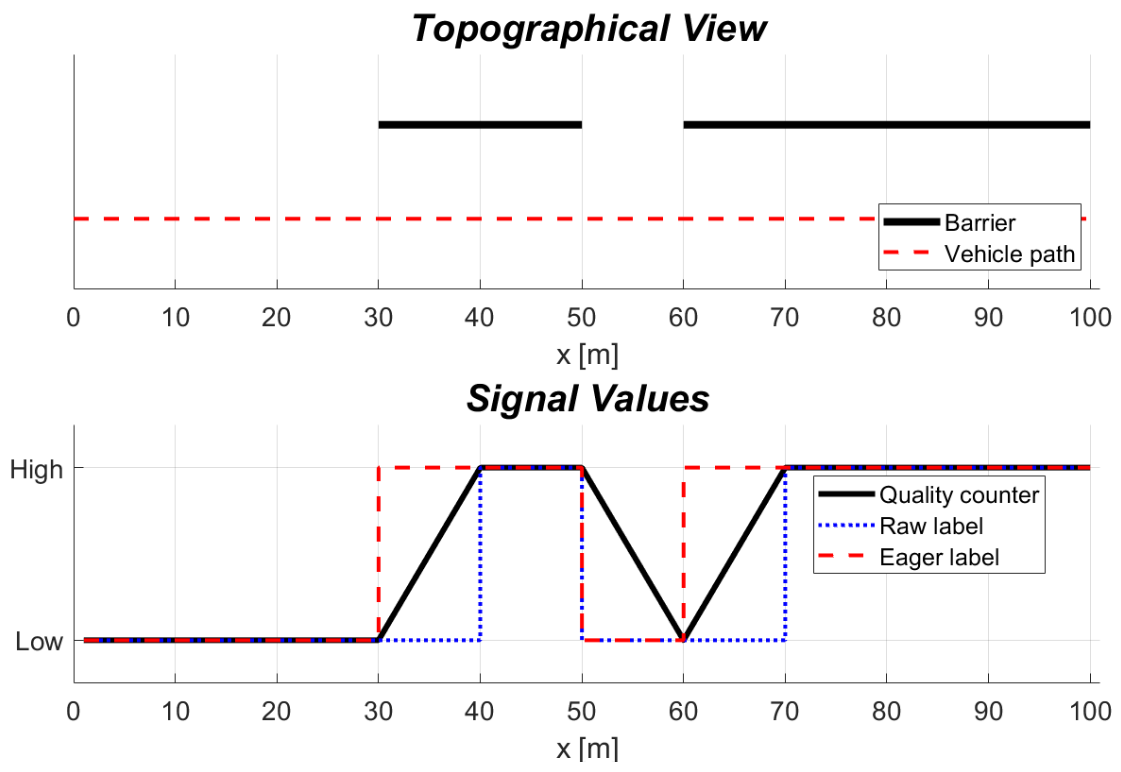

Figure 3.3: A hand-crafted, graphical demonstration of the difference between the quality counter, raw labels (live signal), and the eager labels. signal high means that there is a barrier present.

3.5 Neural networks

In order to solve the classification problems, Feed-Forward Neural Networks were used. The main reason why neural networks are suitable is because they have many adjustable hyperparameters. This allows them to be scaled in order to meet the memory constraints. Once a neural network is trained an output can quickly be computed by feeding an input, which allows it to be used as a real-time application (given that the network is not too large). This is a critical property as there are strict timing and memory requirements in place in an autonomous vehicle.

Google's machine learning library TensorFlow[3] was used to implement and train the neural networks. A motivation for using TensorFlow is that it has built-in functionality for utilizing GPUs when training neural networks [49]. Due to the size of our data set, GPUs were needed. The DNNClassifier class in the TensorFlow Python

[3]In machine learning, a *tensor* is an array with an arbitrary number of indices [47], hence the name TensorFlow. Tensors in mathematics and physics have a more restrictive definition, see [48].

API was used for implementation. The Softmax function was used for activation in the final layer in order to squash the raw outputs into probabilities. Different activation functions were tested for the hidden layers. During training, Softmax cross entropy was used as the cost function. The cost function was optimized with various optimization algorithms based on backprogagation such as AdaGrad, Adam, RMSProp and Ftrl.

The data set was split into 3 sets: 70% training data, 10% validation data, 20% test data. The training stops when the error on the validation data stops decreasing for a given number of iterations. After training, the performance was evaluated on the test data.

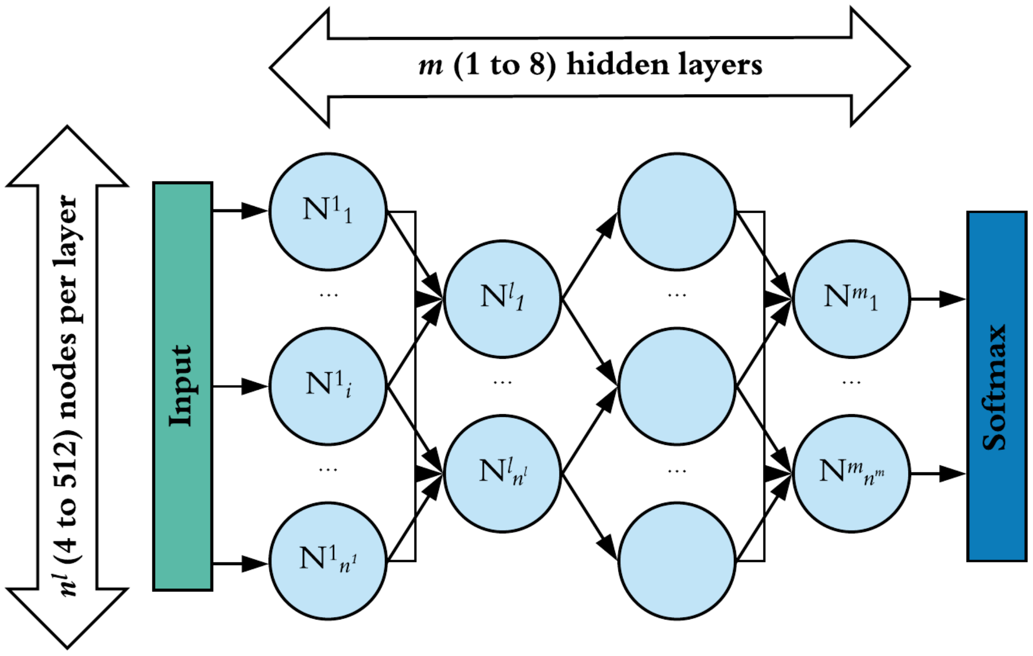

Figure 3.4: FFNNs with 1 to 8 hidden layers and 4 to 512 neurons per layer were used. The output layer was a Softmax layer, which squashed output into probabilities.

3.6 Class imbalance

The distribution of classes was highly skewed in the available data. When using randomly chosen time series the most dominant class was often several orders of magnitude larger (in terms of number of instances) than the smallest (or even the second largest) class. When training a classifier on imbalanced data, there will be learning biases toward majority classes [50]. Previous studies have shown that classification performance on imbalanced data can be improved by using Random Undersampling (RUS) on majority classes [51], [52]. Random Undersampling was therefore used to combat class imbalances.

3.7 Feature selection

Over 2000 signals are typically as inputs. Many of these are vector or matrix quantities, resulting in about 6000 scalar features. Radar and cameras were used as the physical sensors, but raw sensor data was not available. Instead the outputs of preprocessing algorithms, state estimations by Kalman filters, or other types of sensor fusion were used. Most of the signals were irrelevant to both BD and LP, and must be pruned using feature selection. Three different methods were chosen:

Pearson correlation (Equation 2.12) was used to measure the class dependency of each feature. Features were handpicked to form an input set among features with the highest correlated variables.

ReliefF was used as an alternative way of measuring the class dependencies. A feature subset was handpicked among the variables with highest ReliefF ranks. MathWorks' implementation of the algorithm in MATLAB [53] was used, with the number of nearest neighbours set to 5.

Correlation-based Feature Selection is a way to measure the merit of a feature subset. Person correlation was used to measure the dependence between features. The merit score in Equation 2.13 was optimized with an Evolutionary Algorithm, using binary arrays as individuals and a population size of 50 individuals, roulette wheel selection, elitism, and a mutation rate as the reciprocal of the individual length.

3.7.1 Exception for Lane Positioning

The feature selection methods mentioned above were used for BD. While feature selection was performed for the LP problem, it did not yield any useful results. The most correlated features included signals such as blinker activation or steering wheel angle. These signals should not be used. The reason is that because the classifiers should learn to *make a decision*, not just inform of something that is already happening (recall that LP is a D&C problem). Once the steering wheel or blinkers are used, the decision has already been made. Instead of using the feature selection methods described in the previous section, various features relating to road curvature, barriers and nearby vehicles were hand picked.

3.8 Signal Preprocessing

Some of the selected input features underwent additional preprocessing. All features related to objects surrounding the host car were presented in arrays, without order. These objects were sorted in ascending order in terms of distance to the host car. This ensures that the first element in such an arrays is always related to the object closest to the host car, the second element corresponds to the second closest object, and so on.

3.9 Hyperparameters

In order to optimize the prediction accuracy of the ANNs, the following hyperparameters were tested: activation function, network structure, and optimization method. A hyperparameter search is a time-consuming task since an ANN must be trained and evaluated for each hyperparameter configuration. In order to minimize long simulation runs, the hyperparameter search process was carried out on smaller data sets of a few hundred time series. There is a risk of the hyperparameters not being optimal after transitioning to larger data sets, but it can be used to get an idea of what works.

- Activation function:
  - ReLU (Equation A.2)
  - LeakyReLU (Equation A.3)
  - Sigmoid (Equation A.4)
  - Softsign (Equation A.5)
  - Tanh
- Network structure:
  - Number of hidden layers (between 1 and 8)
  - Number of neurons in each hidden layers (between 4 and 512)
- Optimizer:
  - AdaGrad
  - Ftrl
  - Adam
  - RMSProp

3.10 Random Forest

RFs were used as a complement to neural networks for the LP problem. MathWork's implementation in MATLAB, *TreeBagger* [54], was used. The same features were used for RF as for the FFNNs. Each class was sampled evenly by using RUS on the dominant classes. The performance of the RF classifiers were evaluated using the out-of-bag error, see subsection 2.3.2. The RF method is quicker to use since it does not need any hyperparameter tuning, but often performs at a level similar to neural networks [55]–[57].

4
Results

Data sets with a total of 16 to 17 million samples, which corresponds to over 100 hours of driving, were created. The Barrier Detection (BD) and Lane Positioning (LP) problems use their own appropriately selected subsets of this data set.

A set of highly correlated features were identified and classifiers with high performance were found for the BD problem. Features for LP were hand picked to avoid using features which reflect an already taken decision, since LP is a D&C problem. The neural networks failed to attain high accuracy for all LP classification schemes. RF classifiers were able to achieve a high performance for LP labels with 3 uniform intervals, but these classifiers violated the memory constraints.

4.1 Lane positioning

Feed-Forward Neural Networks were initially trained using the (**lane position with** k **intervals** labels, see section 3.2). The best network used labels with 5 intervals and resulted in a 93% classification accuracy on the test set. Due to heavy class imbalances, the neural networks had a huge bias towards the centre interval. The class distribution is plotted in Figure 4.1. When using RUS on the dominant class, the test set accuracy dropped below 50%.

In an attempt to improve performance, the output was converted into a recurrent feature by feeding the previous output as an input. This added memory to the classifier and resulted in high test set accuracy, but the classifiers just repeated last output. For every sample where the lane position changed, the classifiers failed. The lane switch labels (see subsection 3.3.1) were created with the hope that they would allow for better performance without repeating the previous output. By training on the lane position switch labels, the best network got a test set accuracy of 60%, with class balancing.

Finally the **lane position with** k **intervals** labels were attempted again, but with a Random Forest as the classifier. The best such classifiers achieved an out-of-bag accuracy of 96% with 3 intervals, which corresponds to very large offsets.

4.1.1 Class balance

The centre position is, as expected, the dominant class. Figure 4.1 shows the distribution of about 17 million samples. The number of instances drops by several orders of magnitude each step away from the centre. The most dominant class, when the car is the middle interval takes, up 93.89% of all samples. The smallest class, furthest to the left, makes up only 0.3‰ of all samples. Similar imbalances appeared in the class distributions for LP with different k-value labels.

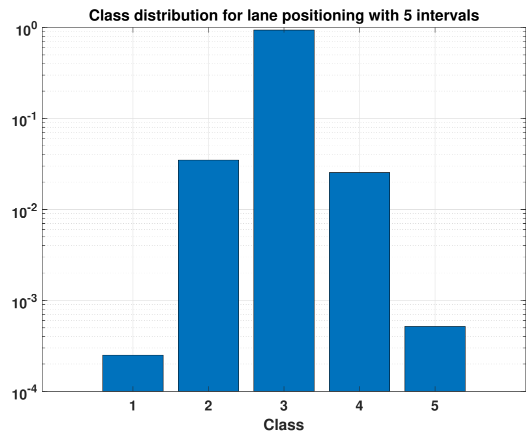

Figure 4.1: Class distribution of a data set when using *lane position with 5 intervals* as labels. The classes are the discrete lane position intervals sorted from left to right. Note that the vertical axis is logarithmic, the most dominant class (class 3) takes up 93.89% of all samples. In comparison the smallest class, furthest to the left (class 1), makes up only 0.3‰ of all samples.

4.1.2 Selected features

The problem with approaching LP with supervised learning is that the classifiers are to learn where the host car *should* be positioned, not where the host car *is* positioned. To do this, the chosen features should be used to detect scenarios in which a lateral offset is desired, not simply reflect an already taken decision. The best correlated signals were those related to blinker activation and steering wheel angle, but these indicate that a decision has already been made. The features selected for LP (see Table 4.1) had poor correlations with the class labels. No significant clusters were identified using t-SNE visualizations.

4.1.3 Random Forest

The best performer was the Random Forest algorithm. Error values are shown in Table 4.2. The method offers little control over the size of the forests (see section 2.3), which resulted in classifiers which were too large to use in our current environment. It is possible that the forests can be made smaller though (see Figure 4.2), since high accuracy is achieved fairly early, however each decision tree consisted of around 30000 nodes which is far too many.

4.2 Barrier Detection

All results in this section are for a network with a single hidden layer of 80 neurons, using Sigmoid as the activation function and trained for the *left* side using Adagrad

Signal	Description
Barrier present (Left)	Barrier within 1.5 m left of the host car, boolean.
Barrier present (Right)	Barrier within 1.5 m right of the host car, boolean.
Road curvature	A measurement of how fast the road turns.
Lane width	Width of the current lane.
Speed (self)	Absolute speed relative to the ground, of the host car.
Local objects	
- Lateral positions	
- Longitudinal positions	
- Lateral velocities	
- Longitudinal velocities	
- Lateral accelerations	
- Heading angles	
- Object curvatures	
- Object types	Truck, motorcycle, pedestrian, etc.
- Motion patterns	Oncoming, receding, stationary etc.
- Speeds	Absolute speed relative to the ground.

Table 4.1: Selected signals for the LP problem. All signals related to local objects (below the dividing line) are vector quantities and sorted as described in section 3.8. These signals explained further in Appendix C.

as the optimizer. Everything in the section applies to the right side as well, but for a different neural network and and with slightly different yet similar numbers. Since data consists of right-hand traffic there is a small asymmetry between the left and the right side. See tables and figures summarizing the neural networks trained for the BD problem in Appendix B.

A test set accuracy of about 80% was reached by the best neural networks on BD with eager labelling. Accuracy among the top performers was very close, so a minor sacrifice in accuracy was made by selecting one of the smaller neural networks to save some valuable memory (see section B.3). The performance of the network is shown in Table 4.3, but note that the numbers in this table are for a much smaller set which had been manually labelled. Manual labelling was done by watching forward-facing camera footage. Incidentally, we discovered that this network has an unintended quirk: it may classify a line of stationary cars as a barrier.

4.2.1 Confusion matrices

Confusion matrices for the BD problem are shown in Table 4.4. As the table shows, the neural network was proficient at correctly identifying barriers. The issue seemed to be false positives, which comes in at 29%. Something interesting happens when switching to manual labels, which includes barriers not detectable by the proprietary algorithm: the number of false positives is reduced by more than 50%. It appears that the classifiers are able to find barriers which are not labelled in the training data. It is not clear why this happens, but there is some discussion on the subject in subsection 5.2.1. The final confusion matrix, "Z/M", compares the training labels

Label	Total samples	Trees	OoB-error
3 intervals	276360	286	0.040549
5 intervals	19760	248	0.29061
7 intervals	9744	287	0.34266
9 intervals	6146	296	0.38926
11 intervals	3130	131	0.44688

Table 4.2: Out-of the bag errors (lower is better) found using random forest and sampling equally number of instances of each class. The higher number of intervals, the less instances exist of the smallest class, hence the decrease in total samples. The third column shows the number of decision trees used to obtain the best result. The features used as inputs are described in Table 4.1.

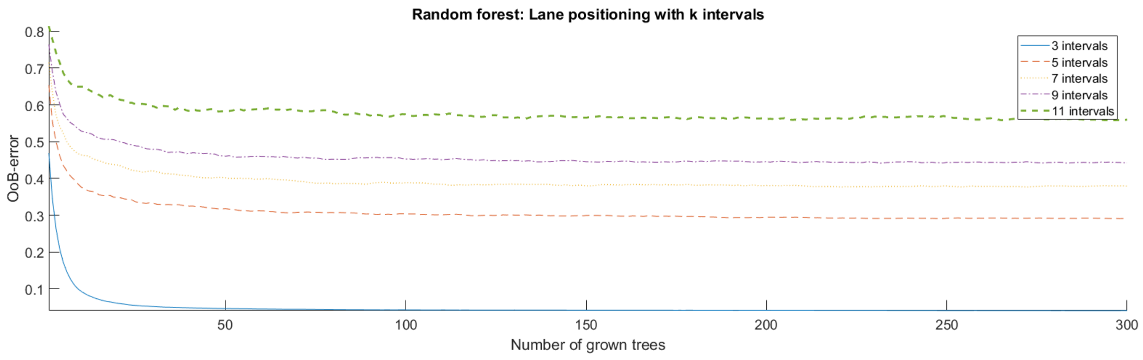

Figure 4.2: Random Forest perfomance on lane positioning with k uniform intervals. Out-of the bag errors plotted against the number of grown decision trees. The features used as inputs are described in Table 4.1.

to the manual labels. This shows that a large number of barriers (mostly concrete barriers) are missed by the proprietary algorithm.

4.2.2 Class balance

Barrier Detection class balance can be seen in Table 4.5. The majority of time series do not contain any barriers, necessitating data mining as well as RUS to aid in the training of the networks. Even if a fair number of series contain barriers on *either* side (39.2%), relatively few individual samples do (11.6%). This necessitated class balancing, even when only selecting time series with barriers present.

4.2.3 Selected features

The selected features for BD are shown in Table 4.6, and visualized with t-SNE in Figure 4.3. Using an Evolutionary Algorithm for optimizing a feature subset with a high metric score resulted in feature subsets which were too large. Instead, the features with highest class-correlation and best rank from ReliefF were selected. Not all highly correlated features were used to compute the final results in order to save memory, but could be included to produce successful classifiers for BD. Local

Predictor	Truth	Accuracy
FFNN	Labels	72.28%
FFNN	Manual	86.81%
Zenuity	Manual	75.97%

Table 4.3: Accuracy for our classifier and the proprietary algorithm (see subsection 3.4.1) on manual labels. The first row shows the performance on the labels (barrier detection with eager labelling) used in training. Note that these values were computed on a smaller data set, which had been manually labelled, of about $5 \cdot 10^4$ samples.

N/Z	T	F
T	0.96	0.29
F	0.04	0.71

N/M	T	F
T	0.88	0.141
F	0.12	0.859

Z/M	T	F
T	0.51	0.17
F	0.49	0.859

Table 4.4: Confusion matrices for the Barrier Detection-network ([80]-Sigmoid-Adagrad), with the true values on the columns. The different sets are: [Z]enuity (with the eager heuristic, i.e. the labels), [N]etwork output, and [M]anual labels. Note that these values were computed on a smaller data set, which had been manually labelled, of about $5 \cdot 10^4$ samples.

objects that were moving were filtered out by a preprocessing step before being fed as inputs to the classifiers, since they cannot be part of a barrier.

The road edge and lane marker signals may seem like strange candidates for detecting barriers, but make sense when considering the fact that a barrier is very often accompanied by one or both. A barrier will of course effectively end the lane in that direction, effectively making it a road edge.

[1]Relative to the ground.

	left only	right only	both	any	neither	total
# series	1181	901	77	1930	4925	4925
% series	24.0	18.3	1.6	39.2	100	
# samples (million)	1.27	0.49	0.03	1.79	14.4	15.7
% samples	8.1	3.1	0.2	11.6	91.7	

Table 4.5: Barrier counts in time series and samples. A *single* sample in a series having a property makes the series count as having that property, so *Left only + right only + both ≠ any* for the series (they do for samples).

Signal	**Description**
Lane marker presence	
Lane marker track status	
Road edge properties	Both sides.
Road edge quality	Reliability of road edge estimations
Road edge track status	
Local objects	*Stationary objects only*[1].
- Lateral positions	
- Longitudinal positions	
- Type	Truck, motorcycle, pedestrian, etc
Excluded significant signals	
Adjacent lane type	Lane type of neighbouring lanes
Current Lane	
Lane marker type	Solid, dashed, unknown etc
Road geometry	
Local objects	
- Motion pattern	Oncoming, receding, stationary etc

Table 4.6: Selected signals for the Barrier Detection problem. All signals related to local stationary objects (below the dividing line) are vector quantities and sorted as described in section 3.8. The signals in this table are further explained in Appendix C.

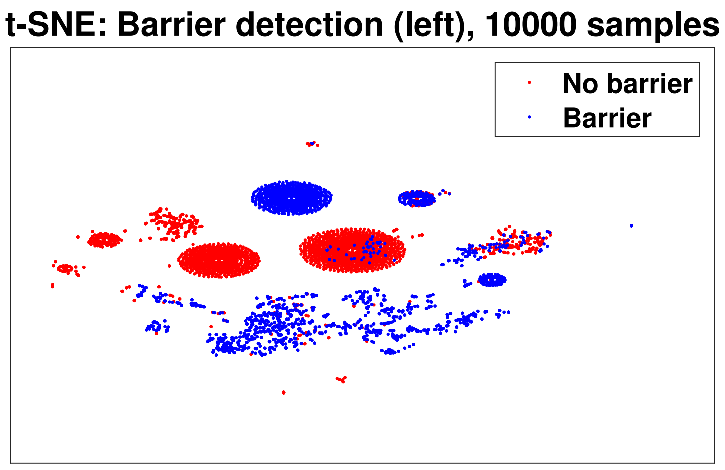

Figure 4.3: Projection of the selected features (see Table 4.6) for BD in 2D using t-SNE. The data has been coloured after manually made labels. Despite some overlap, the samples have been projected into class-wise clusters. This clustering indicates that there is a correlation between the selected features and the presence of barriers. Note that the plot contains only 5000 random samples of each class, a larger data set could potentially result in larger class overlap.

5
Discussion and Conclusions

Our conclusions and suggestions for future work for the Lane Positioning (LP) and Barrier Detection (BD) problems are presented below. In summation, we believe that better data is needed for a machine learning solution to LP. The results for BD are promising, but could be further improved using better ground truth labels.

5.1 Lane Positioning

We were not able to produce any useful classifiers which respected the memory constraints for the Lane Positioning (LP) problem, with accuracy topping out at about 60% for the best FFNNs when expunging any features, such as blinker activation, which indicate that a decision has already been made (recall that we want a decision-making algorithm here, not a position estimator). The best results were achieved with the Random Forest algorithm (4% out-of-bag error), though the structures are quite large. Figure 4.2 shows that only around 15 decision trees are needed to surpass 90% out-of-bag accuracy, but each tree consisted of approximately 30000 nodes. In comparison, the largest neural networks consisted of no more than 1024 neurons. The size of the decision trees might suggest that many more neurons are needed to achieve success with neural networks. However, such large models violate the memory constraints of the autonomous car.

One important remark is that RF could only attain successful results when discretizing the lane into 3 lateral positions. This means that if the lane width is 3.5 meters, then the implementation can only recognize offsets larger than approximately 58 centimeters. At such large offsets the car ends up almost next to the lane edge, which is only desired in rare occasions and during lane changes. It is also possible that the RF classifier learned to identify samples after a lane change, not a position change within the current lane as intended.

We have not found any indication that LP is solvable with smaller offsets on our data set with our classification schemes, and treating each sample independently was probably inappropriate for LP since the position depends on past events. By watching recorded front camera footage we noticed another problem for the supervised learning approach: In most cases when the car had an offset there was no special scenario such as an adjacent freighter, barrier, motorcycle, or road curvature. The driver seemed to have random offsets every now and then, overshadowing the actual relevant scenarios.

5.1.1 Future work

Our suggestions for future work on the Lane Positioning problem:

1. Create and use a data set with less random offsets. We believe this is the most important point, as the lateral offsets seemed to be taken at random more often than they were in response to significant scenarios.
2. Mimicking human drivers may not be the best approach to train a classifier for LP, as driver behaviour is individual and sometimes random.
3. Try alternative machine learning methods, such as:
   - Recurrent neural networks such as LSTM networks, which can remember previous states instead of treating the samples independently.
   - Reinforcement learning. This would require a realistic simulation of the host car in traffic and a suitable reward function. How does on define a reward function that results in *comfort*?
4. Use a classical algorithm such as a scoring system. This approach faces the same issue as reinforcement learning by trying to measure comfort.

5.2 Barrier Detection

For the Barrier Detection (BD) problem, an 80% classification accuracy on a large test set was achieved using FFNNs. The clustering shown in Figure 4.3 indicates that there is a correlation between barrier presence and the selected features. Some of the highly correlated features in Table 4.6 might seem strange at first glance: we noted that several signals related to the quality of road edge estimation were highly correlated with presence of barriers. A possible explanation for this might be that perception algorithms could confuse the top of barriers as the road edge markers, as mentioned in chapter 1. An interesting discovery is that many of highly correlated variables were not used at all by the proprietary algorithm used to create the labels.

5.2.1 Barrier ground truth

As noted in subsection 4.2.1, most false positives were in fact true positives when we looked at the actual camera footage[1]. The proprietary algorithm used as ground truth is not perfect and does make mistakes (which is why we are addressing the BD problem in the first place). Perhaps samples that were incorrectly labelled as *no barrier present* by the proprietary algorithm (false negatives, see subsection 3.4.1) were close to clusters of samples labelled as *barrier present* in the feature space. This might be indicated by the t-SNE projection of the features in the plane, as shown in Figure 4.3.

We believe that the available data can be used to create more accurate classifiers for detecting barriers, given that appropriate ground-truth labels are available. This should be possible using the same features and hyperparameters as we've used.

[1]Keep in mind that the test set with manual labels (around $5 \cdot 10^4$ samples), while not tiny, it is relatively small compared to the total data set.

5.2.2 Future work

Our suggestions for future work on the Barrier Detection problem:

1. Create and use better ground truth labels for training. We believe this is the most important point.
2. Include barriers excluded here, such as traffic cones (we had no reliable way to label them).
3. Try different hyperparameters, especially smaller networks.
4. Create one neural network which handle both the left and the right side. This could save memory compared to having two separate solutions.
5. Introduce several classes per side, sorted by lateral distance to an adjacent barrier. This could also be relevant for LP when weighing different factors.

Bibliography

[1] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization”, *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

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

[3] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, *et al.*, “Ad click prediction: A view from the trenches”, in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, pp. 1222–1230.

[4] T. Tieleman and G. Hinton, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”, *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[5] C. Mercer and T. Macaulaych, “Which companies are making driverless cars?”, 2018. [Online]. Available: https://www.techworld.com/picture-gallery/data/-companies-working-on-driverless-cars-3641537/ (visited on 06/05/2018).

[6] J. Kaplan, “Here’s every company developing self-driving car tech at ces 2018”, 2018. [Online]. Available: https://www.digitaltrends.com/cars/every-company-developing-self-driving-car-tech-ces-2018/ (visited on 06/05/2018).

[7] B. Schoettle and M. Sivak, “A survey of public opinion about autonomous and self-driving vehicles in the us, the uk, and australia”, 2014.

[8] E. Stepp, “More americans willing to ride in fully self-driving cars”, *American Automobile Association*, 2018. [Online]. Available: https://newsroom.aaa.com/2018/01/americans-willing-ride-fully-self-driving-cars/ (visited on 06/05/2018).

[9] R. Parasuraman, T. B. Sheridan, and C. D. Wickens, “A model for types and levels of human interaction with automation”, *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans*, vol. 30, no. 3, pp. 286–297, 2000.

[10] SAE On-Road Automated Vehicle Standards Committee and others, “Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems”, *SAE International*, 2014.

[11] *Drive me - the self-driving car in action*, https://www.volvocars.com/intl/buy/explore/intellisafe/autonomous-driving/drive-me, Accessed: 2018-06-05.

[12] *Autopilot | tesla*, https://www.tesla.com/autopilot, Accessed: 2018-06-05.

[13] K. Mays, *Which cars have self-driving features for 2017?*, https://www.cars.com/articles/which-cars-have-self-driving-features-for-2017-1420694547867/, Accessed: 2018-06-05, 2017.

[14] A. D. Rayome, *Dossier: The leaders in self-driving cars*, https://www.zdnet.com/article/dossier-the-leaders-in-self-driving-cars/, Accessed: 2018-06-05, 2018.

[15] *Make it real*, https://www.zenuity.com/make-it-real/, Accessed: 2018-06-05.

[16] J. Sjöberg, "Road positioning and lane identification for autonomous vehicles", 2016.

[17] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali, "Deeplanes: End-to-end lane position estimation using deep neural networks.", in *CVPR Workshops*, 2016, pp. 38–45.

[18] E. Alpaydin, *Introduction to machine learning*. MIT press, 2014.

[19] G. M. Fitzmaurice, "Regression", *Diagnostic Histopathology*, vol. 22, pp. 271–278, 2016.

[20] S. Shanmuganathan and S. Samarasinghe, *Artificial neural network modelling*. Springer, 2016, vol. 628.

[21] M. Wahde, *Biologically inspired optimization methods: an introduction*. WIT press, 2008.

[22] Onfido. (2018). How to implement a neural network intermezzo 2, [Online]. Available: http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/ (visited on 03/05/2018).

[23] E. Bendersky. (2016). The softmax function and its derivative, [Online]. Available: https://eli.thegreenplace.net/2016/the-softmax-function-and-its-derivative/ (visited on 03/05/2018).

[24] M. D. Zeiler, "Adadelta: An adaptive learning rate method", *arXiv preprint arXiv:1212.5701*, 2012.

[25] T. K. Ho, "Random decision forests", in *Document analysis and recognition, 1995., proceedings of the third international conference on*, IEEE, vol. 1, 1995, pp. 278–282.

[26] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1.

[27] L. Breiman, "Random forests", *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[28] L. Breiman, "Bagging predictors", *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[29] C. Zheng and Z. Wei, "Speeding up boosting decision trees training", in *AOPC 2015: Image Processing and Analysis*, International Society for Optics and Photonics, vol. 9675, 2015, 96750F.

[30] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest", *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[31] G. Ridgeway, "Generalized boosted models: A guide to the gbm package", *Update*, vol. 1, no. 1, p. 2007, 2007.

[32] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning.* Springer, 2013, vol. 112.

[33] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne", *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[34] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space", *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[35] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms.", *Journal of machine learning research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[36] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective", *Neurocomputing*, vol. 300, pp. 70–79, 2018.

[37] C. Granger, E. Maasoumi, and J. Racine, "A dependence metric for possibly nonlinear processes", *Journal of Time Series Analysis*, vol. 25, no. 5, pp. 649–669, 2004.

[38] J. S. Milton and J. C. Arnold, *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*, 4th. New York, NY, USA: McGraw-Hill, Inc., 2002, ISBN: 0071198598.

[39] M. A. Hall, "Correlation-based feature selection for machine learning", 1999.

[40] S. B. Thrun, J. W. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. A. De Jong, S. Dzeroski, D. H. Fisher, S. E. Fahlman, *et al.*, "The monk's problems: A performance comparison of different learning algorithms", Tech. Rep., 1991.

[41] K. Kira and L. A. Rendell, "The feature selection problem: Traditional methods and a new algorithm", in *Aaai*, vol. 2, 1992, pp. 129–134.

[42] I. Kononenko, "Estimating attributes: Analysis and extensions of relief", in *European conference on machine learning*, Springer, 1994, pp. 171–182.

[43] M. Robnik-Šikonja and I. Kononenko, "Theoretical and empirical analysis of relieff and rrelieff", *Machine learning*, vol. 53, no. 1-2, pp. 23–69, 2003.

[44] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification", in *Computational Intelligence, 2015 IEEE Symposium Series on*, IEEE, 2015, pp. 159–166.

[45] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: Improving classification performance when training data is skewed", in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, IEEE, 2008, pp. 1–4.

[46] Volvo, *Driveme research route.* [Online]. Available: https://www.media.volvocars.com/global/en-gb/media/photos/202076/drive-me-research-route-gothenburg-sweden-illustration.

[47] Q. Song, H. Ge, J. Caverlee, and X. Hu, “Tensor completion algorithms in big data analytics”, *arXiv preprint arXiv:1711.10105*, 2017.

[48] S. Weinberg, “Gravitation and cosmology”, *ed. John Wiley and Sons, New York*, 1972.

[49] TensorFlow. (2018). Using gpus | tensorflow, [Online]. Available: https://www.tensorflow.org/programmers_guide/using_gpu (visited on 06/01/2018).

[50] D.-C. Li, C.-W. Liu, and S. C. Hu, “A learning method for the class imbalance problem with medical data sets”, *Computers in biology and medicine*, vol. 40, no. 5, pp. 509–518, 2010.

[51] J. Prusa, T. M. Khoshgoftaar, D. J. Dittman, and A. Napolitano, “Using random undersampling to alleviate class imbalance on tweet sentiment data”, in *Information Reuse and Integration (IRI), 2015 IEEE International Conference on*, IEEE, 2015, pp. 197–202.

[52] Y. Sui, Y. Wei, and D. Zhao, “Computer-aided lung nodule recognition by svm classifier based on combination of random undersampling and smote”, *Computational and mathematical methods in medicine*, vol. 2015, 2015.

[53] MathWorks. (2018). Rank importance of predictors using relieff or rrelieff algorithm - matlab - mathworks nordic, [Online]. Available: https://se.mathworks.com/help/stats/relieff.html (visited on 06/02/2018).

[54] MathWorks. (2018). Create bag of decision trees, [Online]. Available: https://se.mathworks.com/help/stats/treebagger.html#bvf7_tc-1 (visited on 06/02/2018).

[55] M. W. Ahmad, M. Mourshed, and Y. Rezgui, “Trees vs neurons: Comparison between random forest and ann for high-resolution prediction of building energy consumption”, *Energy and Buildings*, vol. 147, pp. 77–89, 2017.

[56] S. Nawar and A. M. Mouazen, “Comparison between random forests, artificial neural networks and gradient boosted machines methods of on-line vis-nir spectroscopy measurements of soil total nitrogen and total carbon”, *Sensors*, vol. 17, no. 10, p. 2428, 2017.

[57] E. Raczko and B. Zagajewski, “Comparison of support vector machine, random forest and neural network classifiers for tree species classification on airborne hyperspectral apex images”, *European Journal of Remote Sensing*, vol. 50, no. 1, pp. 144–154, 2017.

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks”, in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

A

Appendix: Functions and Derivations

A.1 Activation functions

An activation function is the final step before the computed value of a neuron is passed to the next layer (possibly the output layer). Some of the most common activation functions are discussed below. Unless included, the derivative is left as an exercise to the reader. *Man* that feels good to write.

Tanh

The *hyperbolic tangent (tanh)* function is defined as:

$$\tanh(z) = \frac{\exp(2z) - 1}{\exp(2z) + 1} \tag{A.1}$$

The Rectified Linear Unit (ReLU) function

The *rectified linear unit (ReLU)* function is defined as:

$$\mathrm{ReLU}(z) = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases} \tag{A.2}$$

The function is used for a few reasons. It is fast to compute; about six times faster than using tanh neurons [58]. Since the gradient is constant (0 or 1), ReLU also solves the vanishing gradient problem. The final reason is that it is non-linear.

The leaky ReLU function

The *leaky rectified linear unit (LeakyReLU)* function is defined as:

$$\mathrm{ReLU}(z) = \begin{cases} z \text{ if } z > 0 \\ az \text{ otherwise} \end{cases} \tag{A.3}$$

where a is some *small, constant* value. This is used to avoid the gradient becoming zero for $z < 0$, which may cause convergence issues in some cases. There are other variants of the ReLU function where a is a tunable parameter.

The sigmoid function

The *sigmoid* function is defined as:

$$\text{Sigmoid}(z) = \frac{1}{1+e^{-z}} \tag{A.4}$$

The softsign function

The *softsign* function is defined as:

$$\text{Softsign}(z) = \frac{z}{1+|z|} \tag{A.5}$$

A.2 Softmax

The softmax function is defined as:

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_K \end{pmatrix} = \frac{1}{\sum_{j=1}^{K} e^{z_j^L}} \begin{pmatrix} e^{z_1^L} \\ \vdots \\ e^{z_K^L} \end{pmatrix}$$

y is a vector containing the estimated probabilities for each class, K is the number of classes, z^L is the local field vector of

A.2.1 Derivative of Softmax

$$\frac{\partial y_i}{\partial z_i^L} = \frac{e^{z_i^L} \sum_j^K e^{z_j^L} - e^{z_i^L} e^{z_i^L}}{\left(\sum_j^K e^{z_j^L}\right)^2} = \frac{e^{z_i^L}}{\sum_j^K e^{z_j^L}} \frac{\sum_j^K e^{z_j^L} - e^{z_i^L}}{\sum_j^K e^{z_j^L}} = y_i(1-y_i) = y_i - y_i y_i$$

$$k \neq i \quad \frac{\partial y_i}{\partial z_k^L} = \frac{0 - e^{z_i^L} e^{z_k^L}}{\left(\sum_j^K e^{z_j^L}\right)^2} = -\frac{e^{z_i^L}}{\sum_j^K e^{z_j^L}} \frac{e^{z_k^L}}{\sum_j^K e^{z_j^L}} = -y_i y_k$$

$$\implies \frac{\partial y_i}{\partial z_j^L} = \delta_{ij} y_i - y_i y_j \quad \forall i, j$$

A.2.2 Softmax cross-entropy derivation

When training a neural network with softmax classification, the network parameters can be tuned with a maximum likelihood estimation. Let θ denote a set of weights and biases in a neural network.

$$\underset{\theta}{\text{argmax}}\, \mathcal{L}\left(\theta \mid t, z^L\right)$$

Since only one $t_i = 1$ and all other $t_j = 0, \forall j \neq i$, the true labels t follows a categorical distribution. The likelihood function for categorically distributed variables can be written:

$$\mathcal{L}\left(\theta \mid t, z^L\right) = p_\theta(t, z^L) = \prod_{i=1}^{K} P\left(t_i, z^L \mid \theta\right)^{t_i}$$

This can be rewritten into: (Joint probability)

$$P\left(t, z^L \mid \theta\right) = P\left(t \mid z^L, \theta\right) P\left(z^L \mid \theta\right) \rightarrow P\left(t \mid z^L, \theta\right) \rightarrow P\left(t \mid z^L\right)$$

Now the likelihood function can be rewritten:

$$\mathcal{L}\left(\theta \mid t, z^L\right) = \prod_{i=1}^{K} P\left(t_i \mid z^L\right)^{t_i} = \prod_{i=1}^{K} y_i^{t_i}$$

By looking at the log-likelihood, the likelihood function gets simplified into a sum.

$$\log\left(\mathcal{L}\left(\theta \mid t, z^L\right)\right) = \log\left(\prod_{i=1}^{K} y_i^{t_i}\right) = \sum_{i=1}^{K} t_i \log\left(y_i\right)$$

Finding θ that maximizes the likelihood is the same θ that minimizes the negative log-likelihood function. It turns out that the negative log-likelihood function is a cross entropy function H.

$$H(t, y) = -\sum_{i=1}^{K} t_i \log\left(y_i\right)$$

A.2.3 Derivative of Softmax cross-entropy

$$\begin{aligned}
\frac{\partial H}{\partial z_i^L} &= -\sum_{j=1}^{K} t_j \frac{\partial \log(y_i)}{\partial z_i^L} = -t_i \frac{\partial \log(y_i)}{\partial z_i^L} - \sum_{j\neq i}^{K} t_j \frac{\partial \log(y_j)}{\partial z_i^L} \\
&= -\frac{t_i}{y_i}\left(\frac{e^{z_i^L}\sum_{k=1}^{K} e^{z_k^L} - e^{z_i^L} e^{z_i^L}}{\left(\sum_{k=1}^{K} e^{z_k^L}\right)^2}\right) + \sum_{j\neq i}^{K} \frac{t_j}{y_j}\left(\frac{e^{z_i^L} e^{z_j^L}}{\left(\sum_{k=1}^{K} e^{z_k^L}\right)^2}\right) \\
&= -\frac{t_i}{y_i}\left(\frac{\sum_{k=1}^{K} e^{z_k^L} - e^{z_i^L}}{\sum_{k=1}^{K} e^{z_k^L}}\right) \frac{e^{z_i^L}}{\sum_{k=1}^{K} e^{z_k^L}} + \sum_{j\neq i}^{K} \frac{t_j}{y_j} y_i y_j \\
&= -\frac{t_i}{y_i}(1 - y_i) y_i + \sum_{j\neq i}^{K} t_j y_i = -t_i + t_i y_i + \sum_{j\neq i}^{K} t_j y_i \\
&= -t_i + \sum_{j=1}^{K} t_j y_i = -t_i + y_i \sum_{j=1}^{K} t_j = y_i - t_i
\end{aligned}$$

A.3 Backpropagation

This section derives the equations of backpropagation for the derivatives in subsection 2.2.3.

A.3.1 Error of output layer

Equation 2.3a describes the error in the output layer, δ^L. The error is defined as the derivative of the cost function C respect to the local field z^L. Let n denote the number of neurons in the output layer L and a_i^L denote the activation of the output neuron with index i, $i = 1, 2, 3, \dots n$. Using the multivariate chain rule the error for output neuron i can be computed:

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L}\frac{\partial a_i^L}{\partial z_i^L} = \frac{\partial C}{\partial a_i^L}\frac{\partial \sigma^L(z_i^L)}{\partial z_i^L} \tag{A.6}$$

σ^L is the activation function, $a_i^L = \sigma^L(z_i^L)$. Equation A.6 can rewritten on matrix form:

$$\delta^L = \begin{pmatrix} \delta_1^L \\ \vdots \\ \delta_n^L \end{pmatrix} = \begin{pmatrix} \frac{\partial C}{\partial a_1^L} \\ \vdots \\ \frac{\partial C}{\partial a_n^L} \end{pmatrix} \odot \begin{pmatrix} \frac{\partial \sigma^L(z_1^L)}{\partial z_1^L} \\ \vdots \\ \frac{\partial \sigma^L(z_n^L)}{\partial z_n^L} \end{pmatrix} = \nabla_a^L C \odot \partial\sigma^L(z^L)$$

The $\odot$ symbol denotes element-wise multiplication.

A.3.2 Error of hidden layers

In this subsection, the error for the hidden layers $(l \neq L)$ will be derived, see Equation 2.3b. The error for neuron i in layer l are defined as the derivative of the cost function respect to the local field.

$$\delta_i^l := \frac{\partial C}{\partial z_i^l}$$

Now lets compute the error for the last hidden layer, $l = L - 1$.

$$\begin{aligned} \delta_i^{L-1} = \frac{\partial C}{\partial z_i^{L-1}} &= \sum_j \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial a_i^{L-1}}\frac{\partial a_i^{L-1}}{\partial z_i^{L-1}} = \sum_j \delta_j^L \frac{\partial z_j^L}{\partial a_i^{L-1}}\frac{\partial a_i^{L-1}}{\partial z_i^{L-1}} && \text{(A.7)} \\ &= \sum_j \delta_j^L w_{ji}^L \frac{\partial a_i^{L-1}}{\partial z_i^{L-1}} && \text{(A.8)} \end{aligned}$$

Every neuron j in layer L depends on z_i^{j-1}, this causes a sum over the j indicies to appear. Note that Equation A.8 is consistent with the matrix equation for layer $l = L-1$, see Equation 2.3b. Now lets attempt to compute the error for any hidden layer.

$$\begin{aligned} \delta_i^l = \frac{\partial C}{\partial z_i^l} &= \sum_j \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L}\frac{\partial z_j^L}{\partial z_i^l} = \sum_j \delta_j^L \frac{\partial z_j^L}{\partial z_i^l} = \sum_k \sum_j \delta_j^L \frac{\partial z_j^L}{\partial a_k^{L-1}}\frac{\partial a_k^{L-1}}{\partial z_k^{L-1}}\frac{\partial z_k^{L-1}}{\partial z_i^l} \\ &= \sum_k \sum_j \delta_j^L w_{jk}^L \frac{\partial a_k^{L-1}}{\partial z_k^{L-1}}\frac{\partial z_k^{L-1}}{\partial z_i^l} = \sum_k \delta_k^{L-1}\frac{\partial z_k^{L-1}}{\partial z_i^l} = \{\text{recursive pattern emerges}\} \\ \dots &= \sum_n \delta_n^{l+1} w_{ni}^{l+1}\frac{\partial a_i^l}{\partial z_i^l} = \sum_n \delta_n^{l+1} w_{ni}^{l+1}\frac{\partial \sigma^l(z_i^l)}{\partial z_i^l} \end{aligned}$$

Now the expression for the errors is a recursive formula:

$$\delta_i^l = \sum_j \delta_j^{l+1} w_{ji}^{l+1} \frac{\partial \sigma^l(z_i^l)}{\partial z_i^l} \tag{A.9}$$

Equation A.9 can be written on matrix form:

$$\delta^l = \begin{pmatrix} \delta_1^l \\ \vdots \\ \delta_n^l \end{pmatrix} = \left(\begin{pmatrix} w_{11}^{l+1} & \dots & w_{1n}^{l+1} \\ \vdots & \ddots & \vdots \\ w_{m1}^{l+1} & \dots & w_{mn}^{l+1} \end{pmatrix}^T \begin{pmatrix} \delta_1^{l+1} \\ \vdots \\ \delta_m^{l+1} \end{pmatrix} \right) \odot \begin{pmatrix} \frac{\partial \sigma^l(z_1^l)}{\partial z_1^l} \\ \vdots \\ \frac{\partial \sigma^l(z_n^l)}{\partial z_n^l} \end{pmatrix} = \left((w^{l+1})^T \delta^{l+1} \right) \odot \partial \sigma^l(z^l)$$

n, m is the number of neurons in the layer l, respective $l+1$.

A.3.3 Bias derivatives

In order to update the biases using gradient descent, an expression for the bias derivatives need to be derived. By using the errors the derivatives can be derived in similar fashion as Equation A.9.

$$\begin{aligned} \frac{\partial C}{\partial b_i^l} &= \sum_j \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_i^l} = \sum_j \delta_j^L \frac{\partial z_j^L}{\partial b_i^l} = \sum_k \sum_j \delta_j^L \frac{\partial z_j^L}{\partial a_k^{L-1}} \frac{\partial a_k^{L-1}}{\partial z_k^{L-1}} \frac{\partial z_k^{L-1}}{\partial b_i^l} \\ &= \sum_k \sum_j \delta_j^L w_{jk}^L \frac{\partial a_k^{L-1}}{\partial z_k^{L-1}} \frac{\partial z_k^{L-1}}{\partial b_i^l} = \sum_k \delta_k^{L-1} \frac{\partial z_k^{L-1}}{\partial b_i^l} = \{\text{recursive pattern emerges}\} \\ \dots &= \sum_n \delta_n^{l+1} w_{ni}^{l+1} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l} = \sum_n \delta_n^{l+1} w_{ni}^{l+1} \frac{\partial \sigma^l(z_i^l)}{\partial z_i^l} \cdot 1 = \delta_i^l \end{aligned}$$

Putting it together, we get:

$$\frac{\partial C}{\partial b_i^l} = \delta_i^l \tag{A.10}$$

Equation A.10 can be written on matrix form:

$$\nabla_b^l C = \begin{pmatrix} \frac{\partial C}{\partial b_1^l} \\ \vdots \\ \frac{\partial C}{\partial b_n^l} \end{pmatrix} = \begin{pmatrix} \delta_1^l \\ \vdots \\ \delta_n^l \end{pmatrix} = \delta^l$$

A.3.4 Weight derivatives

$$\begin{aligned} \frac{\partial C}{\partial w_{ij}^l} &= \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_k^L} \frac{\partial z_k^L}{\partial w_{ij}^l} = \sum_k \delta_k^L \frac{\partial z_k^L}{\partial w_{ij}^l} = \sum_m \sum_k \delta_k^L \frac{\partial z_k^L}{\partial a_m^{L-1}} \frac{\partial a_m^{L-1}}{\partial z_m^{L-1}} \frac{\partial z_m^{L-1}}{\partial w_{ij}^l} \\ &= \sum_m \sum_k \delta_k^L w_{km}^L \frac{\partial a_m^{L-1}}{\partial z_m^{L-1}} \frac{\partial z_m^{L-1}}{\partial w_{ij}^l} = \sum_m \delta_m^{L-1} \frac{\partial z_m^{L-1}}{\partial w_{ij}^l} = \{\text{recursive pattern emerges}\} \\ \dots &= \sum_n \delta_n^{l+1} w_{ni}^{l+1} \frac{\partial a_i^l}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1} \end{aligned}$$

This results in the following scalar equation:

$$\frac{\partial C}{\partial w_{ij}^l} = \delta_i^l a_j^{l-1} \tag{A.11}$$

Equation A.11 can be written on matrix form:

$$\nabla_w^l C = \begin{pmatrix} \frac{\partial C}{\partial w_{11}^l} & \cdots & \frac{\partial C}{\partial w_{1n}^l} \\ \vdots & \ddots & \vdots \\ \frac{\partial C}{\partial w_{m1}^l} & \cdots & \frac{\partial C}{\partial w_{mn}^l} \end{pmatrix} = \begin{pmatrix} a_1^{l-1} \\ \vdots \\ a_m^{l-1} \end{pmatrix} \begin{pmatrix} \delta_1^l & \ldots & \delta_n^l \end{pmatrix} = a^{l-1}(\delta^l)^T$$

n, m is the number of neurons in the layer l, respective $l-1$.

B
Appendix: Barrier Detection

Layers are read left to right: 1-2-3 denotes a network where the first layer (after the input layer) has 1 node, the second layer has 2 nodes, and the final layer (before the output layer) has 3 nodes.

B.1 Figures

Figure B.2 and Figure B.1 show the results of the hyperparameter search. The 8 best/worst networks are shown on the barcharts. Numbers are based on about 5000 time series.

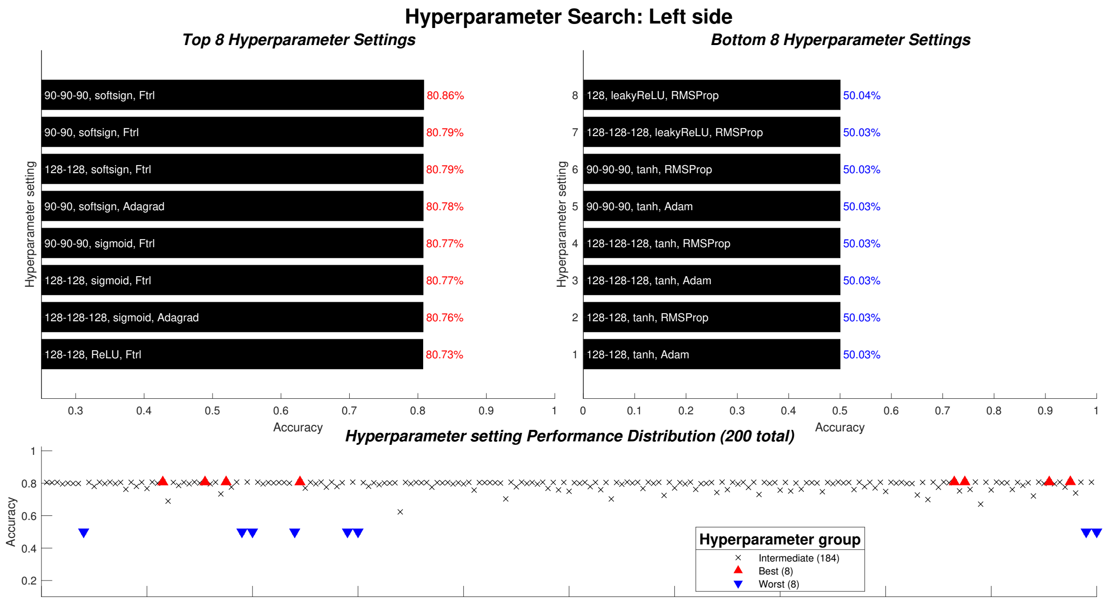

Figure B.1: Hyperparameter results for the left side. Set size: $2.3 \cdot 10^5$ samples.

B.2 Tables

Table B.1 and Table B.2 show the accuracy and error rates for a few networks. The networks in the tables are not sorted, but have been chosen to highlight a few typical networks. "Raw" refers to the proprietary algorithm labels without the eager

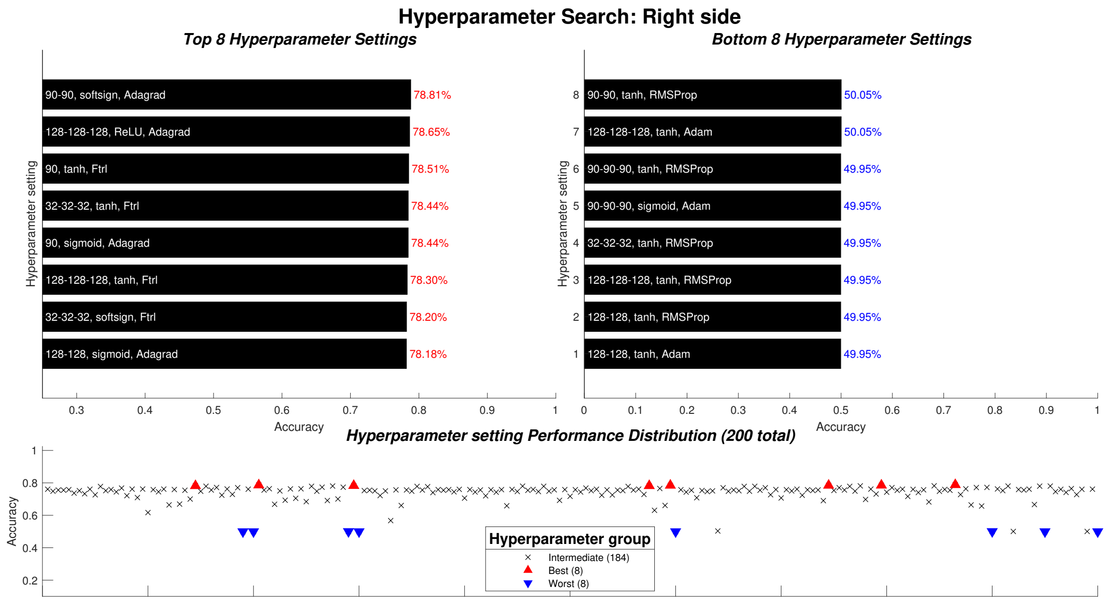

Figure B.2: Hyperparameter results for the right side. Set size: $2.3 \cdot 10^5$ samples.

heuristic. A 100% T2 error-rate means that all points with barriers were classified incorrectly. Numbers are based on about 15 time series, a subset of the ca 5000 time series used to produce the figures.

Hidden layers	Activation function	Optimizer	Manual A/T1/T2	Eager A/T1/T2	Raw A/T1/T2
90-90-90	softsign	Ftrl	86.62/14.31/11.95	75.90/29.42/4.43	74.59/30.66/3.92
90-90	softsign	Adagrad	78.79/27.18/12.10	68.18/39.23/4.45	66.90/40.25/3.87
90	sigmoid	Adagrad	86.70/13.97/12.29	76.17/29.03/4.59	74.87/30.27/4.07
128-128	tanh	Adam	60.40/0.00/100.00	78.70/0.00/100.00	80.35/0.00/100.00
90	tanh	Ftrl	87.10/13.30/12.29	76.50/28.57/4.76	75.25/29.79/4.15
80	sigmoid	Adagrad	86.81/14.09/11.81	75.97/29.33/4.45	74.67/30.57/3.92
90-90	softsign	Ftrl	86.77/13.71/12.49	76.23/28.85/5.01	74.97/30.07/4.42

Table B.1: Performance for a few different networks, left side. A/T1/T2: Accuracy/Type-1 error/Type-2 error, in percent. Sample size: $5 \cdot 10^4$ points.

Hidden layers	Activation function	Optimizer	Manual A/T1/T2	Eager A/T1/T2	Raw A/T1/T2
90	tanh	Ftrl	79.59/20.93/16.48	77.53/21.74/8.89	77.15/23.88/7.75
90-90-90	softsign	Ftrl	85.47/11.12/40.21	87.00/11.60/22.86	87.01/12.51/20.07
80	sigmoid	Adagrad	75.11/26.57/12.22	72.28/26.56/7.26	71.90/29.61/5.92
90-90	softsign	Adagrad	79.58/21.06/15.58	77.59/21.74/6.84	77.21/23.97/5.55
128-128	tanh	Adam	88.30/0.00/100.00	92.97/0.00/100.00	93.61/0.00/100.00
90-90	softsign	Ftrl	85.75/10.87/39.77	87.26/11.49/22.22	87.27/12.28/19.36
90	sigmoid	Adagrad	79.70/20.71/17.20	77.94/20.97/7.95	77.56/23.51/6.67

Table B.2: Performance for a few different networks, right side. A/T1/T2: Accuracy/Type-1 error/Type-2 error, in percent. Sample size: $5 \cdot 10^4$ points.

B.3 Network size estimates

Assumptions: Weights and biases are 4-byte double values. Each neuron has a single bias value and as many weights as the number of neurons in the previous layer. It is assumed that the input layer has 50 neurons (one per feature). We're assuming fully connected layers as well. Note that this is an *optimistic* estimate. In practice, a non-trivial amount of additional memory will be needed for the softmax layer and for execution. The formula used is shown in Equation B.1, and values calculated for the networks in Table B.1 and Table B.2 are shown in Table B.3.

$$\text{Approximate size} = \Sigma_{l=1}^{N}(O_{l-1} \cdot O_l + B_l) \tag{B.1}$$

where O_0 is the the number of inputs to the network, O_l is the number of outputs in layer l, and B_l is the number of biases in layer l. Note that $O_l = B_l$. The value $O_{l-1} \cdot O_l$ is simply the size of the weight matrix in layer l. For a network with the

Hidden layers	Size (KB)
80	16
90	18
90-90	50
90-90-90	82
128-128	90

Table B.3: Estimated size in kilobytes (1 KB = 2^{10} bytes) of networks with a given number of hidden layers. The leftmost layer connects to the input layer.

layer structure "90-90-90", Equation B.1 yields: Approximate size $= (50 \cdot 90 + 90) + 2 \cdot (90 \cdot 90 + 90) = 20970$, since we assume $O_0 = 50$. Using 4 bytes per variable, the final size is 83880 bytes $= 83880/1024$ KB $= 81.91$ KB.

C

Appendix: Signals

These section explains the selected signals as inputs as described in Table 4.6 and Table 4.1. Note that all signals are online estimations, not ground truths.

- **Adjacent lane type** - Lane type of neighbouring lanes, also flags if an adjacent lane does not exist.
- **Barrier present** - Barrier within 1.5 m left of the host car, output of the barrier detector.
- **Lane marker presence** - Check if the host car identify any nearby lane markers.
- **Lane marker track status** - Status of lane marker tracking. Is the tracking of adjacent lane markers reliable, have the current tracked markers been seen in a previous sample or is it a new marker.
- **Road edge quality** - Measurement of how reliable the road edge estimations are.
- **Road geometry** - Geometry of drivable area.
- **Local objects**: The signals below are related to local objects such as cars, bikes, road signs, etc.
  - **Motion pattern** - Enum if an object is oncoming, receding or stationary relative the host car.
  - **Object types** - Enum if an object is identified as a truck, motorcycle, pedestrian etc.