Cut-In Detection

By the use of a Neural Network

S. Geboers
11065788
The Hague University of Applied Sciences
Bachelor Thesis of Applied Physics (B.Eng)
Performed at Volvo Car Corporation
Gothenburg, Sweden
September 2016

First reader:	R.H.M. Smit	Department:	Drive Me
Second reader:	R.A. Mantel	Supervisor:	M. Ali

Abstract

In 2017 Volvo will deliver 100 self-driving cars as part of the Drive Me project. These cars will be able to drive autonomously on the ring road of Gothenburg, without requiring any driver supervision. To realize this vision, the cars need different algorithms to recognize, and be able to react to, different traffic situations. One of these traffic situations is when a car unexpectedly cuts in, in front of the self-driving car. For the self-driving car to react to the cut-in, it first needs to detect the cut-in.

The main question of this research is: Is it possible to detect a cut-in by the use of a neural network? A neural network is a network with the ability to learn without being explicitly programmed. The neural network is provided with examples of a cut-in and examples of a non-cut-in and learns to recognize these examples. The neural network that is being used to detect a cut-in is provided with 70 samples of each cut-ins and non-cut-ins. In this thesis, the optimal conditions to train this neural network are examined.

The input variables used to create the cut-in samples are determined. The four parameters chosen as input variables to the neural network are the lateral position, lateral velocity, and trajectory angle of the car making a cut-in with respect to the road, and the width of the lane that the host vehicle is driving upon. These parameters contain 40 measuring points per second. The Research carried out in this thesis shows that this can be reduces to 20 measurements per second without influencing the performance of the neural network. This benefits the memory use of the neural network.

The detection of a cut-in is most useful when the cut-in is detected before it actually happens. This way the host vehicle has time to react to the situation. The usability of the network increases the earlier that a cut-in can be detected. The requirement states that a car executing a cut-in must be detected at least 4 seconds before the car crosses the lane marker. When the previous described input variables and this time interval of 4 seconds are used to train the neural network, it can reach a performance of 90.5%. This means that 90.5% of the cut-ins or non-cut-ins are correctly identified. This is considered to be a well trained neural network. Because the network becomes more useful for a shorter time interval, a time interval of 3 seconds is tested. This time interval starts 4 seconds before the car crosses the lane-marker and ends 1 second before the car crosses the lane marker. The network reaches a performance of 57.1%. This is a useless result,

so this time interval is too small. The last interval tested is 3.5 seconds. The interval starts 4 seconds before the car crosses the lane marker and stops half a second before the car crosses the lane marker. This network reaches a performance of 81.0%. This is an acceptable result, but it is significantly lower than the time interval of 4 seconds. Depending on the preference, one could choose the time interval of 4 or 3.5 seconds. If a secure detection is chosen to be more important, it is wise to choose the time interval of 4 seconds. If a fast detection is chosen, one could choose the time interval of 3.5 seconds.

There are a lot of different algorithms that can be used to train a neural network. This specific neural network used to detect cut-ins, is trained with the Levenberg-Marquardt backpropogation algorithm.

Contents

Chapter 1

Drive Me

In 2017 Volvo will deliver 100 self-driving cars as part of the Drive Me project. These cars will be able to drive autonomously on the ring road of Gothenburg, without requiring any driver supervision. The unique aspect of Drive Me is that it will not just involve prototypes which are used solely as technology demonstrators. Drive Me takes things a few steps further, since the technology will actually be used by real customers. Using a combination of cameras, lasers and radars to keep track of its surroundings, the cars will be able to navigate on their own. The car will feature a data upload to the Volvo Cloud, which will provide the car with a detailed map and a certification signal that specifies whether the car is allowed to drive autonomously.

The project is a central component of Volvo Cars' plan to achieve sustainable mobility and ensure a crash-free future by 2020. The small scale test on the ring of Gothenburg is a start. For the car to be able to drive on all roads and enable door-to-door autonomous driving, it should be able to handle situations such as busy intersections, pedestrians and cyclists. Also, the traffic laws have to be adapted in order for autonomous cars to participate in traffic.

Drive Me consist of several teams. The autonomous car has to sense its environment (Sensing System/Sensor Fusion), has to react to this and plan its path (Decision and Control). This plan has to be executed by the actuators in the car (Architecture and System Solution). Additionally there are teams that deal with the interaction between human and machine and teams for safety, verification and integration.

The research discussed in this thesis is carried out in the Decision and Control team. The Decision and Control team is responsible for both the high-level strategic and tactical planning algorithms as well as the lower level longitudinal and lateral control of the vehicle.

Chapter 2

Data processing

The Decision and Control team writes algorithms that control the reaction and the plan of the host vehicle. The host vehicle is the self-driving car in question. For example, when and how to make a lane change or when and how the car has to break. To make this decisions the car uses multiple radars, cameras, a laser and ultrasonic sensors to monitor the complete 360 degrees view of the surroundings and to generate data. This data is used to write and test algorithms. To obtain this data, employees of Drive Me go on 'expeditions' all around the world. The obtained data is broken down to processable files of 70 seconds. One file of 70 seconds is called a logfile.

There are different algorithms to give meaning to this data. For example: The car can distinguish 32 different objects in the surroundings of the car. Objects like a car, a cyclist or a pedestrian. Of these objects, different parameters are known. For example the lateral and longitudinal position, velocity and acceleration. There is also a lot of information about the road. For example: The kind of lane markers (solid, dotted etc.), the number of lanes or the degree of the curvature of the road. All these parameters have their own algorithm.

The main focus of this report will be to find a way to detect a cut-in. The definition of a cut-in is a car making a lane change into the lane the host-vehicle is driving on, in front of the host vehicle. There already is an algorithm that is used to detect cut-ins. However, this algorithm is not nearly as accurate as it is desired. Cut-ins are missed and false cut-ins are detected. An option to obtain a well working cut-in detection could be the use of a neural network. A neural network is a network with the ability to learn. By providing the network with examples of a cut-in, it could be able to learn to recognize cut-ins. The main question of this research will be: Is it possible to detect a cut-in by the use of a neural network?

Chapter 3

Neural Networks

The main focus of this thesis will concern the affect of machine learning towards the detection of a cut-in. Machine learning is a field of study that gives computers the ability to learn without being explicitly programmed. A way to accomplish machine learning is by using a neural network. Let's begin with the definition of the term 'Neural Network'.

A neural network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the network is stored in the interunit connection strengths, or weights, obtained by a process of adaptation to, or learning from, a set of training patterns [1].

Neural Networks are based on the brain. When someone writes down their phone number, most people effortlessly recognize the digits. The human head carries supercomputers, tuned by evolution. Neural Networks work roughly the same. The idea is to take a large number of handwritten digits, these are known as training examples. The neural network uses the examples to automatically infer rules for recognizing handwritten digits. The more training examples are available, the more accurate the network. To fully understand a neural network, first an artificial neuron called a perceptron will be explained.

3.1 Perceptrons

Perceptrons were developed in the 1950s and 1960s by the scientist Frank Rosenblatt. A perceptron takes several binary inputs, $x_1, x_2 \ldots$, and produces a single binary output. A schematic example can be seen in figure 3.1. Rosenblatt decided on a simple rule to compute the output. The importance of the inputs are expressed by weights: $w_1, w_2 \ldots$. A threshold is set. The output, 0 or 1, is determined by

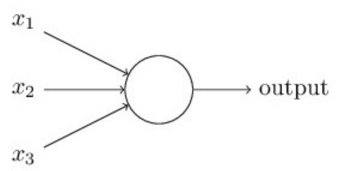


Figure 3.1: A schematic view of a perceptron. [15]

[1]Kevin Gurney. An introduction to Neural Networks. Number 0-203- 45151-1. UCL Press, 1997.

if the weighted sum $\sum_j w_j x_j$ is less or greater than the threshold value. In algebraic terms:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold}; \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold}. \end{cases} \tag{3.1}$$

The threshold is a real number which is a parameter of the neuron. So, a perceptron can weigh up different kinds of evidence in order to make decisions. A more complex network of perceptrons can be seen in figure 3.2.

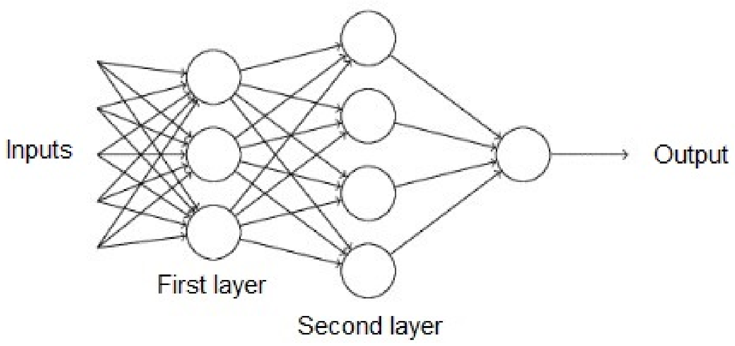


Figure 3.2: Network of perceptrons. [15]

In this network, there are different 'layers' of perceptrons. The perceptrons in the first layer make three simple decisions. The perceptrons in the second layer make decisions on a more abstract level. They still have one output. The multiple output arrows are merely a useful way of indicating that the output from a perceptron is being used as the input to several other perceptrons. To simplify equation 3.1 we write w and x as vectors whose components are respectively the weights and inputs, now the dot product can be used. We bring the threshold to the other side of the equation. The bias is introduced: bias $b = -\text{threshold}$. Now equation 3.1 can be written as:

$$\text{output} = \begin{cases} 0 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0; \\ 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0. \end{cases} \tag{3.2}$$

The bias can be seen as a degree of difficulty to 'activate' the perceptron.

It turns out that learning algorithms which can automatically tune the weights and biases of a network of artificial neurons can be devised. Suppose a small change in some weights or bias is made in the network. This small change of weight corresponds to a small change in the output of the network. This fact can be used to get the network to behave correctly. For example recognizing handwritten digits. When mistakenly the network classifies an image as a 3 when it should be a 9, the weights and biases need to be slightly adjusted. The network gets a little closer to classify the digit correctly. During the process of adjusting the biases and weights the output will become better and better. The network is learning. [9] [15] [19]

3.2 Sigmoid Neurons

For a network containing perceptrons, the output does not become better and better. A small change in weights or bias could cause a change of the output from, say, 1 to 0. Now 9 can be classified correctly but the behavior of the network on all the other images could be completely off. This problem can be overcome by introducing the sigmoid neurons. The sigmoid neurons look like perceptrons. Instead of the input being 0 or 1 the input can be any value between 0 and 1. This leads to a different output. The output becomes $\sigma(\mathbf{w} \cdot \mathbf{x} + b)$. Herein σ is called the sigmoid function, and is defined by:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}, \tag{3.3}$$

herein:

$$z \equiv \sigma(\mathbf{w} \cdot \mathbf{x} + b). \tag{3.4}$$

Just like the perceptron, the sigmoid neuron has weights for each input and an overall bias.

To understand the similarity to the perceptron model, suppose z is a large positive number: $e^{-z} \approx 0$ and $\sigma(z) \approx 1$. Suppose z is a very negative number: $e^{-z} \to \infty$ and $\sigma(z) \approx 0$. In extreme cases the behavior of a sigmoid neuron is a close approximation of a perceptron. It is only when z is a modest size that there is much deviation from the perceptron model.

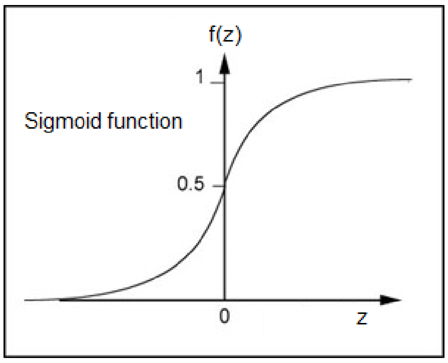


Figure 3.3: Shape of a sigmoid function.

The most important property of the sigmoid function is the shape. The shape is a smoothed version of a step function. This can be seen in figure 3.3. If σ would be the step function, the sigmoid neuron would act like a perceptron. Because of the sigmoid function, a small change in biases Δb and weights Δw_j will produce a small change in the output (Δoutput).

$$\Delta\text{output} \approx \sum \frac{\partial \text{output}}{\partial w_j} \partial w_j + \frac{\partial \text{output}}{\partial b} \partial b. \tag{3.5}$$

Δoutput is a linear function of the changes Δw_j and Δb in the weights and biases. This makes it easy to achieve any desired small changes in the output.

3.3 The structure of a Neural Network

In figure 3.4 an example of a neural network with multiple hidden layers can be seen. Neurons in a hidden layer are neither input neurons nor output neurons. This is the only reason why they are called hidden layers. The network in figure 3.4 has two hidden layers. In this network, the output of one layer is used as the input to the next layer. There are no loops in this network. This is called a *feedforward* neural network. A network with feedback loops is called a *recurrent* neural network. Recurrent neural networks will not be further discussed in this thesis.

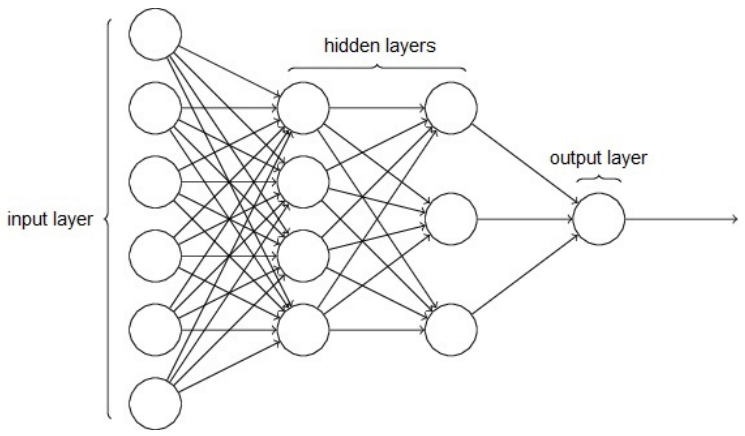


Figure 3.4: Structure of a neural network with hidden layers. [15]

3.4 Training a Neural Network

To learn how to recognize a specific pattern, a neural network needs a set of so called training data. This training data contains samples of the pattern that has to be recognized. The training data will be split into three categories: training data, validation data and testing data. The majority of the samples will be used as training data. Training data is presented to the network during training. Validation data is used to measure network generalization, and to halt training when generalization stops improving. Generalization is a measurement of how accurately an algorithm is able to predict the outcome values of the neural network. Testing data has no effect on training, it provides an independent measure of network performance during and after training. The network will also be presented with target data. Target data is a matrix with vectors of zeros and ones that define the desired network output. The notation of the training data:

- n Number of training samples.
- $\mathbf{x}$ Input variables, this is a vector.
- $\mathbf{y}$ Output, this is a vector of zeros and ones

For example a neural network could be used to classify if breast cancer is benign or malignant depending on the characteristics of sample biopsies. The training data could be a 10x700 matrix, defining ten parameters $\mathbf{x}$ of 700 biopsies n. The target data would be a 2x700 matrix where each column indicates a correct category with a one in either the benign or malignant row. These are 700 vectors $\mathbf{y}$.

An other way to look at the bias is as an extra input with a fixed value of one. This way the weight of this input becomes the bias. The vector of the combination of the weights and biases is called $\boldsymbol{\theta}$. The dot product of $\boldsymbol{\theta}$ and the input values $\mathbf{x}$ will look like:

$$\boldsymbol{\theta} \cdot \mathbf{x_i} = \theta_0 x_0 + \theta_1 x_1 + ... + \theta_j x_j \tag{3.6}$$

Here i is the i'th training sample and j is the number of inputs in one training sample. x_0 will be set to 1. From now on this notation will be used.

The output of the network must approximate $y(x_i)$ for every training input x_i. To quantify how well this is happening, the quadratic cost function is introduced:

$$C(\theta) \equiv \frac{1}{2n} \sum_{i=1}^{n} \Big(y(x_i) - a(x_i, \theta) \Big)^2 . \tag{3.7}$$

The aim of an training algorithm is to minimize the cost as a function of θ. In equation 3.7 $a(x_i, \theta)$ is called the activation of the output of the network. This activation is depending on the weights and biases. The sigmoid function of chapter 3.2 is applied. x_i is the i'th training sample. $C(\theta)$ goes to zero when $y(x_i)$ is approximately equal to the output $a(x_i, \theta)$ for all training inputs. Equation 3.7 is called the Mean Square Error (MSE). [13]

3.5 Neural Network training algorithms

There are many different algorithms to achieve a minimization of the cost function. Two of these algorithms are studied and compared in this thesis. The first is Gradient Descent. Gradient Descent is the most common approach for training neural networks. The second algorithm is the Levenberg-Marquardt algorithm. This algorithm is more robust, it finds a solution even when it starts far from the final minimum. The downside of the Levenberg-Marquardt algorithm is that it tends to be slower than the Gradient Descent algorithm.

3.5.1 Gradient Descent

Gradient Descent is an iterative optimization algorithm. In figure 3.5 an animation of gradient descent can be seen. When using the gradient descent algorithm, a point in the plot in figure 3.5 is chosen. From there, small steps will be taken to the local minimum. Figure 3.5 shows two examples of the path that can be taken. The gradient descent algorithm that is being used to choose this pad:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} C(\theta) \tag{3.8}$$

Equation 3.8 [2] will be repeated until convergence. α is called the learning rate (or tuning parameter). The learning rate controls the size of the steps that the algorithm takes.

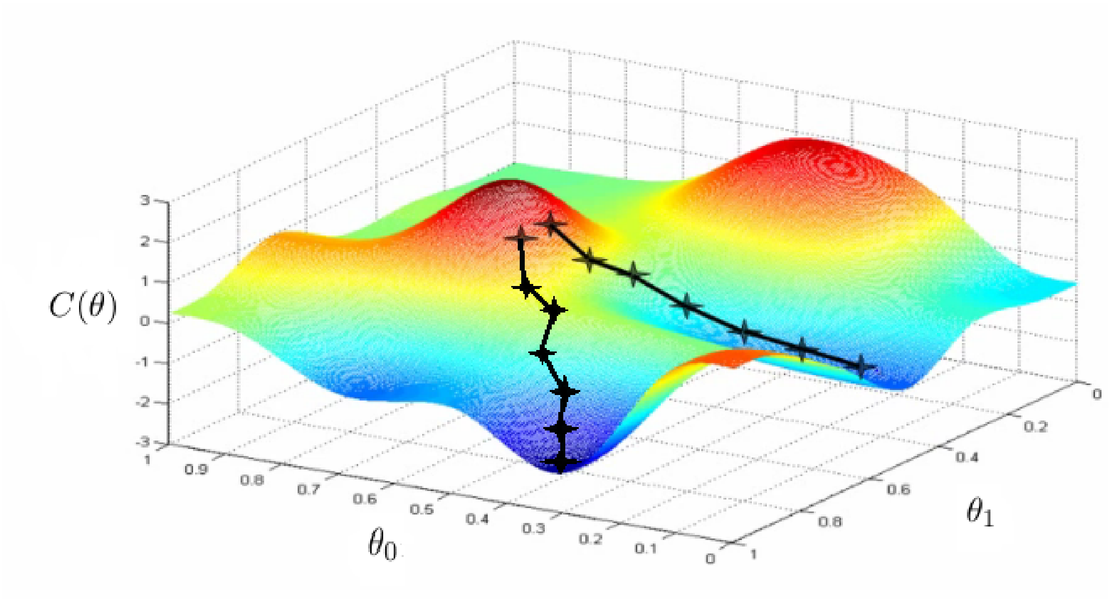


Figure 3.5: Animation of gradient descent. [13]

To clarify equation 3.8, a one dimensional example can be used. A schematic view of the example can be seen in figure 3.6. The black curve indicates $C(\theta)$ while the red dot indicates the position of θ_j on the x-axis. At the position of θ_j, the derivative of $C(\theta)$ is taken. When θ_j is on the right, the derivative will be positive. When θ_j is on the left, the derivative will be negative. This causes θ_j to move to the minimum. α controls the size of the steps that θ_j will take. When α is too small, it will take θ_j a lot of steps to reach the minimum. Gradient descent will be slow. When α is too large, θ_j will overshoot the minimum and gradient descent can diverge instead of converge. There is no need to decrease α over time. As the local minimum is approached, gradient descent will automatically take smaller steps because the derivative will get smaller.

[2]When $a := b$ is being used, as it is in equation 3.8, it means that b overwrites a.

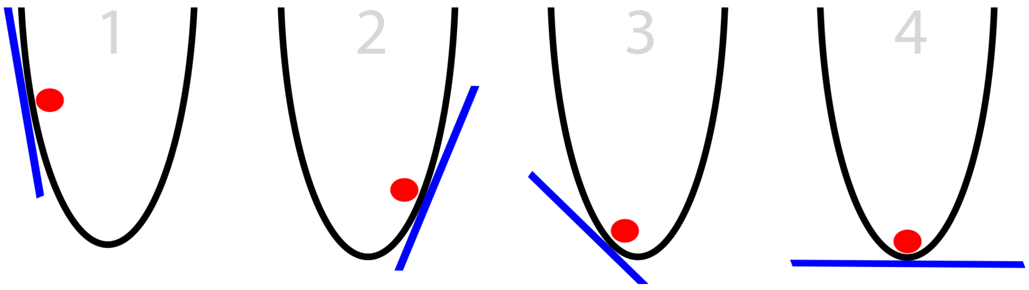


Figure 3.6: Schematic view of a one dimensional example of gradient descent. [5]

The key part of equation 3.8 turns out to be $\frac{\partial}{\partial\theta_j}C(\theta)$. To complete the gradient descent algorithm, this derivative has to be solved. When equation 3.7 is inserted in the derivative this gives:

$$\frac{\partial}{\partial\theta_j}C(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{2n}\sum_{i=1}^{n}\Big(y(x_i) - a(x_i,\theta)\Big)^2 \tag{3.9}$$

Executing this derivative yields:

$$\frac{\partial}{\partial\theta_j}C(\theta) = \frac{1}{n}\sum_{i=1}^{n}\Big(y(x_i) - a(x_i,\theta)\Big)x_i \tag{3.10}$$

This gives the Gradient Descent algorithm that will yield the local minimum of the cost function. [13] [14] :

$$\theta_j := \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\Big(y(x_i) - a(x_i,\theta)\Big)x_i \tag{3.11}$$

3.5.2 Levenberg-Marquardt backpropogation

The Levenberg-Marquardt (LM) algorithm is considered to be one of the most effective minimization algorithms and can also be used to solve non-linear problems. The LM algorithm is an iterative procedure. Again, θ_j will be optimized so that the cost function becomes minimal. This time the Sum of Squared Errors will be used as a cost function (equation 3.12) because of the derivation later this chapter.

$$C(\theta) \equiv \sum_{i=1}^{n}\Big(y(x_i) - a(x_i,\theta)\Big)^2. \tag{3.12}$$

To start minimization, an initial guess for θ_j has to be made. In each iteration step, θ_j

will be updated by an increment step λ (i.e. $\theta_j := \theta_j + \lambda_j$). To find a suitable λ, $a(x_i, \theta)$ is approximated by its Taylor expansion:

$$a(x_i, \theta + \lambda) = a(x_i, \theta) + J_{i,j}\lambda. \tag{3.13}$$

Where $J_{i,j}$ is the derivative of $a(x_i, \theta)$ with respect to θ_j:

$$J_{i,j} = \frac{\partial a(x_i, \theta)}{\partial \theta_j}. \tag{3.14}$$

Combining equation 3.12 and 3.13 gives:

$$C(\theta + \lambda) \approx \sum_{i=1}^{n} \Big(y(x_i) - a(x_i, \theta) - J_{i,j}\lambda \Big)^2. \tag{3.15}$$

The right hand sight of equation 3.15 is an approximation, based on the Taylor expansion for $a(x_i, \theta)$ plugged into equation 3.12.

Now, the λ that minimizes this expression can be found by rewriting equation 3.15 to the vector notation and set the derivative to zero. This derivation can be seen in appendix A and leads to:

$$\Big(\mathbf{J}^{\mathrm{T}}\mathbf{J}\Big)\lambda = \mathbf{J}^{\mathrm{T}}\Big[\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big]. \tag{3.16}$$

J is the Jacobian matrix. The Jacobian matrix is the matrix of all first-order partial derivatives of a vector-valued function. The i^{th} row equals J_i (equation 3.14). **a** and **y** are vectors of the i^{th} training sample respectively $\mathbf{a}(\mathbf{x_i}, \boldsymbol{\theta})$ and $\mathbf{y_i}$. λ can be solved for the set of equations in 3.16.

Lavenberg (1944) suggested to use a 'damped version' of equation 3.16:

$$\Big(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu\mathbf{I}\Big)\lambda = \mathbf{J}^{\mathrm{T}}\Big[\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big]. \tag{3.17}$$

Herein **I** is the identity matrix and μ is the non-negative damping parameter (or tuning parameter). The damping factor adjusts every iteration. The step λ is now defined as in equation 3.17. If the update parameter λ leads to a reduction of the cost function, the update is accepted and the process repeats with a decreased damping parameter μ. If reduction of the cost function is rapid, a smaller value for μ can be used. Equation 3.17 becomes closer to equation 3.16. If an iteration gives insufficient reduction of the cost function, μ can be increased. This can lead to a higher reduction. The reason for this is that an increase of the damping parameter typically leads to a shorter step towards the minimum. If either the length of the calculated step λ, or the reduction of the cost function falls below predefined limits, iteration stops and the last parameter vector $\boldsymbol{\theta}$ is considered to be the solution.

The LM algorithm actually solves a slight variation of equation 3.17. Marquardt replaced the identity matrix with the diagonal matrix consisting of the diagonal elements of $\mathbf{J^TJ}$. This matrix is called $\mathbf{N}$. This results in the Levenberg-Marquardt algorithm: [18] [8] [10] [12] [6]

$$\left(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu\mathbf{N}\right)\lambda = \mathbf{J}^{\mathrm{T}}\left[(\mathbf{y}) - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\right]. \tag{3.18}$$

3.6 Evaluate a Neural Network

When the neural network is trained, the performance has to be evaluated. Does the network have enough neurons, inputdata or training samples? There are different ways to draw these conclusions. Several plots or tables can be used to outline the performance of the neural network. A few of these evaluation methods are discussed in this chapter.

3.6.1 Performance plot

A performance plot shows the MSE or SSE dynamics for all datasets on a logarithmic scale. An example of a performance plot can be seen in figure 3.7. The lower the MSE or SSE at the end of the training phase, the better the network is trained. This means that the desired outputs and the neural network's output for the training set have become very close to each other.

An ideal sketch of the plot can be seen in figure 3.8. The MSE or SSE reduces after more epochs (iterations) of training, but might start to increase on the validation data set as the network starts over fitting the training data. This should be avoid so the training should stop when the validation error starts increasing instead of decreasing. However, for a real neural network training, the validation set error does not evolve as smoothly as seen in figure 3.8. Real validation error curves almost always have more than one local minimum. So, the training doesn't stop after the first increase of the validation error but after six consecutive increases. This number is set to 6 with the aid of trial and error. The minimum error is found at the minimum of the validation set. The best performance is taken from the epoch with the lowest validation error. [17]

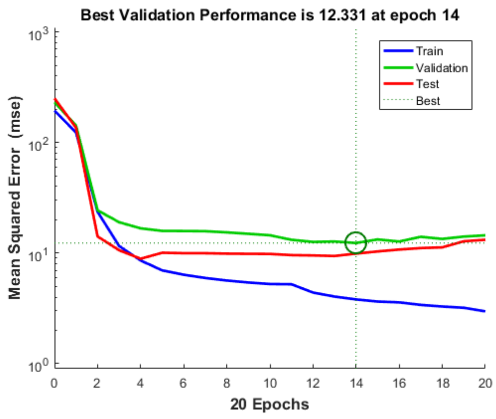


Figure 3.7: Performance plot example. [2]

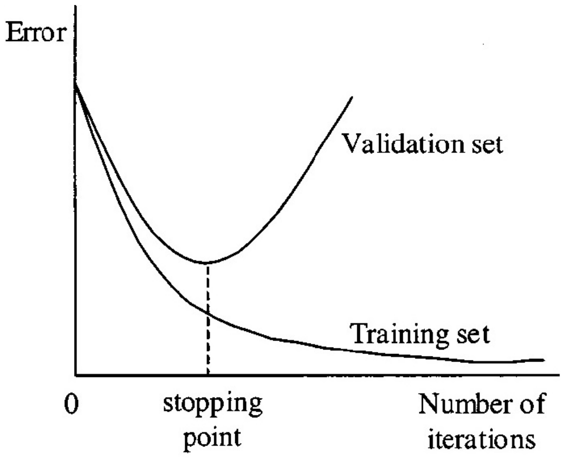


Figure 3.8: Ideal sketch of a performance plot.

3.6.2 Confusion matrix

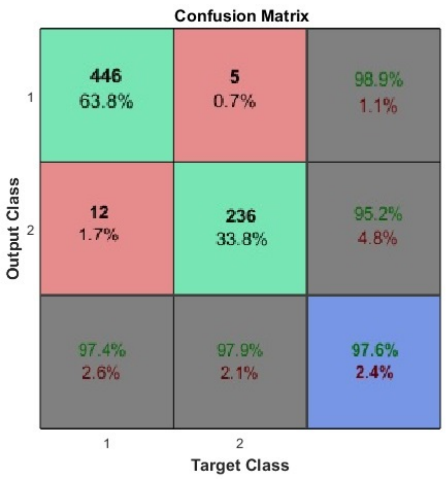


Figure 3.9: Example of a confusion matrix. [1]

A confusion matrix is a specific table layout that allows visualization of the performance of an algorithm, such as a neural network. Each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class. It is easy to see if the system confuses two classes. An example of an confusion matrix can be seen in figure 3.9. In this example there where $446 + 236 + 5 + 12 = 699$ training samples. 446 of these samples (63,8% of all the samples) where correctly detected as 1, 5 of these samples (0.7% of all the samples) where detected as 1 but where in fact a 2. 98.9% of the training samples that belong to 1 where correctly detected. Naturally, 1.1% was falsely detected. This information can be found in the first row of the table. In the same way, conclusions can be drawn from the second row of the table. In total, 97.6% of the training samples where correctly detected and 2.4% of the training samples where falsely detected. This can be seen in the blue box. [16]

3.6.3 Receiver Operating Characteristic

Receiver Operating Characteristics, or ROC, is used to evaluate the accuracy of a statistical model that classifies subjects into 1 of 2 categories. Like a neural network with 1 output leading to 0 or 1.

The curve is created by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). The TPR is the proportion of positives that are correctly identified as such, also known as the sensitivity. For example the percentage of cars making a cut-in that are correctly identified as making a cut-in. The FPR is 1 - specificity. Specificity is the proportion of negatives that are correctly identified as such. For example, the FPR is the percentage of cars that are not making a cut-in falsely identified as a cut-in. The FPR is also known as the fall-out. In figure 3.10, an example of the ROC-curve can be seen. The sensitivity as a function of the fall-out. For the curve to make sense, 1 - specificity is plotted on the x-axis instead of the specificity. As the sensitivity gets higher, the specificity goes down. Curve 'A' in figure 3.10 represents a perfect test, 100% sensitive and 100% specific. The surface area under the curve is 1. The diagonal line 'C' in figure 3.10 traces the curve of a useless test. The surface area under the curve is 0.5. It would be the same as a coin-toss. Curve 'B ' represents a more realistic outcome for a test. Thus, by looking at this curve an approximation of the performance of the neural network can be made. [11]

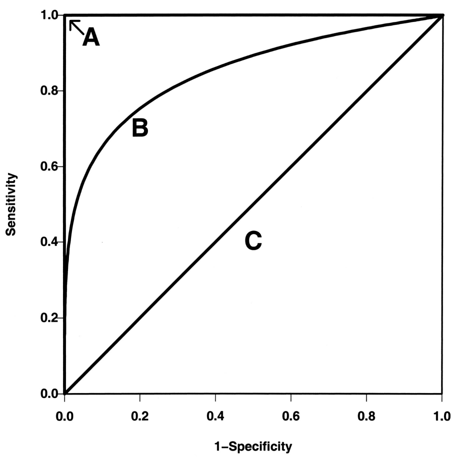


Figure 3.10: Example of a ROC curve. [11]

Chapter 4

Cut-in detection using a Neural Network

Now that the theory of a neural network is explained, it can be taken into practice. In this chapter, a neural network will be used to detect a cut-in. A cut-in is defined by a car changing lanes to end up into the lane of the host vehicle. An option to detect this movement is by pattern recognition. As seen in chapter 3, pattern recognition can be achieved by the use of neural networks. This chapter will be divided into three subjects: creating the training data, training the neural network and evaluation of the neural network.

4.1 Creating training data

To train a neural network, it must be provided with training samples $\mathbf{x}$ and target data $\mathbf{y}$, as is explained in chapter 3.4. To create this training data, logfiles are needed. The logfiles from the expedition from Kiel to Kassel (Germany) in January 2015 are used. These will be referred to as: Kiel_ Kassel logfiles. The neural network requires that it is provided with training samples of a cut-in and training samples of a non-cut-in. A non-cut-in is a car riding in the lane on the left or right of the host vehicle, that does not end up making a cut-in. In this section the creation of the training samples for both of these situations will be explained respectively. In this chapter, ‘the car’ always refers to the car making the cut-in. The final code that is being used to train the neural network can be found in appedix B. An impression of the product of the code can be seen at the end of this section in table 4.1.

4.1.1 Create training examples of a cut-in

To create training examples of a cut-in, it is necessary that there are examples of cars making a cut-in. These cut-ins need to be found manually. For the Kiel_ Kassel data, there is an excel sheet available with information about which logfiles contain a cut-ins and at what time this car crosses the lane marker. However, it is important to know which of the 32 optional objects is making the cut-in. This information is obtained by manually looking through the videos corresponding to the logfiles. Now all the data that is needed to create training examples is available. After all the cut-ins are identified in the available log data, they are processed into training data using the following algorithm:

- Loading logfiles that contain a cut in.
  The Kiel_ Kassel logdata contains 156 logfiles. 50 of these logfiles contain at least one cut-in. There are 70 cut-ins in total. The algorithm starts by only loading the logfiles that contain at least one cut-in.

- View every object making a cut-in.
  The object number of the car making the cut-in is selected. Only the data of this object number is viewed. If there is more than one cut-in in one logfile, the object numbers will be viewed one after another. When the car making a cut-in is selected, its data will be further processed.

- For this selected car, different parameters are viewed.

  - Select the lateral position of the selected car.
    The first parameter used to create training data is the lateral position of the car that is making a cut-in in the logged data. This is the lateral position of the car with regard to the road. This is preferred over the lateral position with regard to the host vehicle because otherwise a car in front of the host vehicle that is already in a curve could give the same values for lateral position as a car making a cut-in.
  - Select the lateral velocity of the selected car.
    The second parameter is the lateral velocity of the car. This is simply said the derivative of the lateral position. Information about the lateral velocity can be derived from the information about the lateral position. To help the neural network it is used as an input.
  - Select the angle of the selected car with regard to the road.
    The third parameter is the angle of the car with regard to the road. This data is not available without making some calculations first. The data that is directly available is the angle of the car with regard to the host vehicle. Because of the same reason the lateral position with regard to the host vehicle can't be used, this angle can't be used either. Therefore, the angle of the car with regard to the road is needed. This data can be obtained by subtracting the angle of the road with regard to the host vehicle from the angle of the car with regard to the host vehicle. The angle of the road with regard to the host vehicle has to be calculated. The road in front of the host vehicle is divided in 3 segments. The first segment is the segment that is closest to the car. For every segment estimates of the curvature of the road (1/m) and the curvature rate of the road (1/m^2) are known. The estimate of the angle of the road at the position of the host vehicle is also known. With this information, the angle of the road with regard to the host vehicle can be calculated at any point.

  - Select the width of the lane the car is driving on. The last parameter that is being used to create training data is the width of the lane the car is driving on. This data is important to give the lateral position of the car more meaning. If the lane width is small then a small difference in the lateral position could mean a cut-in. The same lateral position characteristics in a wide lane may be due to less-disciplined driving behavior which is tolerable from a risk perspective because of increased margins for path following error.

- Determine the time interval of the cut-in.
  Detection of a cut-in is most useful when it is detected before it actually happened. The further that the timing of a detection is advanced, the more useful the detection is. The neural network will be trained to be able to predict cut-ins, by presenting it with data from before the car crosses the lane marker. The training data presented to the neural network can be adjusted to this requirement. Different time intervals are presented to the network. The time interval used in appendix B ends at the moment the car crosses the lane marker and starts 4 seconds before this point. This time interval is assigned to the lateral position, lateral velocity and the angle of the car with regard to the road. It is crucial that the neural network is not presented with an interval that is too short. If this happens the network will not be able to detect a cut-in at all. Every training sample has to have the same amount of measuring points because the neural network has to be provided with a symmetric matrix.

- Provide a solution for samples that are too short.
  Not every logfile has 4 seconds before the cut-in. It can be the case that the cut-in starts before the logfile reaches 4 seconds. These situations are selected. Every cut-in with too little measuring points at the beginning of the logfile will fill the missing measuring points with the first measuring point the logfile has for that object. This will simulate the car driving straight in its own lane until the real measuring points can be used.

- The obtained training examples will be stored in a 482×70 trainingmatrix.

4.1.2 Create training examples of a non-cut-in

The training data of a neural network needs to contain samples of non-cut-in scenarios as well. In order to recognize the pattern of a cut-in the neural network needs a frame of reference. The training matrix will be extended with 70 samples of a non-cut-in. Again, the steps the most extensive algorithm takes to create a matrix of training examples are explained:

- The first 70 logfiles are loaded one after another.

- An object is selected.
  The first object number on the drivers side of the host vehicle with 160 measuring points is selected. If this situation does not occur, the first object number on the passengers side of the host vehicle with 160 measuring points is selected.

- For this selected car, different parameters are viewed.
  The same input values are selected for these selected objects. Those input variables are: the lateral position with regard to the road, the lateral velocity with regard to the road, the angle of the car with regard to the road and the lane width of the lane the car is driving on. These parameters are collected for a time interval of 4 seconds for the previous selected car.

- The obtained training examples will be stored in a 482×140 trainingmatrix.
  The obtained training samples will be stored in the same matrix as the training samples of the cut-in. This matrix will be extended to a 482×140 trainingmatrix. If there is no car on the left and the right for 160 samples, the training vector will fill with zeros. This is not a problem because a column of zeros also represents a non-cut-in.

4.1.3 Create target data

The target data will define the desired network output. For the training samples that represent a cut-in the target data will contain a 1. For the training samples that represent a non-cut-in the target data will contain a 0. This is 1×140 matrix where the first 70 columns contain a 1. The 71th till the 140th column will contain a 0 to represent a non-cut-in. When a neural network has more then 1 output, this matrix will have more rows, one for every output.

4.1.4 Impression of the training matrix

In table 4.1 an impression of the product of the code can be seen. This is what the set-up of the input matrix and the targetdata matrix looks like. The dark cells represent a cut-in and the light cells represent a non-cut-in. The blue cells contain information about the lateral position. The green cells contain information about the lateral velocity. The grey cells contain information about the angle of the car. The orange cells contain information about the width of the row. The yellow cells contain the desired network output.

Table 4.1: An impression of the structure of the matrix with training data and targetdata.

	Training sample 1	...	Training sample 70	Training sample 71	...	Training sample 140
Measurement 1 lateral position						
⋮						
Measurement 160 lateral position						
Measurement 1 Lateral velocity						
⋮						
Measurement 160 lateral velocity						
Measurement 1 angle of the car						
⋮						
Measurement 160 angle of the car						
Measurement lane width						

Target data	1	1	1	0	0	0

4.2 Train the Neural Network

When the training data and target data are complete, the network has to be designed. This leads to some decision making. How many hidden layers will the network get? How many hidden neurons will every layer have? How will the training data be distributed? And of course which minimization algorithm will be used? These questions will be answered in this section. Every different set of training samples has it's own answers to these questions. The decision making will be explained for the training set described in chapter 4.1

Minimization algorithm: For every set of training samples, the Levenberg-Marquardt backpropagation algorithm is going to be used to minimize the cost function. Minimization using the LM algorithm is slower then using the GD algorithm but it leads to better results. Also, the network does not need as much hidden layers when the LM algorithm is being used.

Number of hidden layers: There is no theory yet to tell how many hidden layers are needed. It is wise to start with one hidden layer. More complex problems have been solved using one hidden layer. Every extra hidden layer makes the network potentiallty unnecessarily complicated. For every set of training samples, one hidden layer is going to be used. [3]

Number of hidden neurons: The number of hidden neurons is based on a complex relationship between the number of input and output neurons, the amount of training data available, the complexity of the function that is trying to be learned and the training algorithm that is being used. Too few hidden neurons will lead to a high error for your system as the predictive factors might be too complex for a small number of hidden neurons to capture, this is called underfitting. Too many hidden neurons will lead to the problem of overfitting. Unfortunately, there is no hard rule for the number of hidden neurons. On the internet, many rules of thumb can be found. All these rules are invalidated as well. Trial and error will be used to estimate the right number of hidden neurons. The best way to estimate if the network is over or underfitting is by looking at the performance plot. In figure 4.1, 4.2 and 4.3, examples of respectivily a underfitting, overfitting and well working network can be seen. [4]

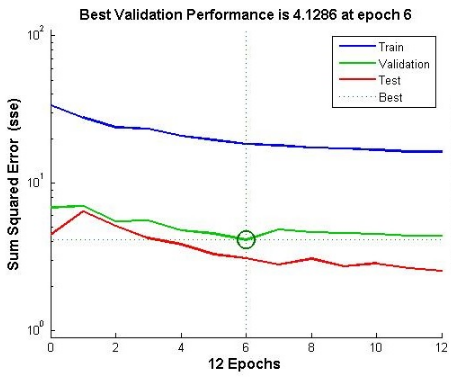


Figure 4.1: A underfitting network .

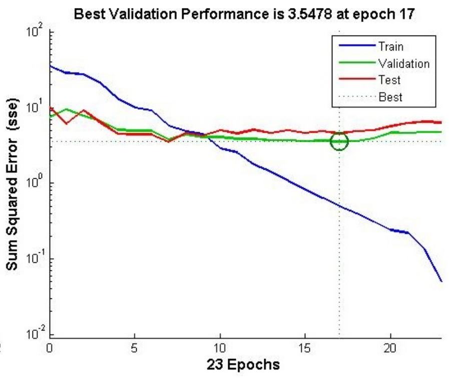


Figure 4.2: A overfitting network.

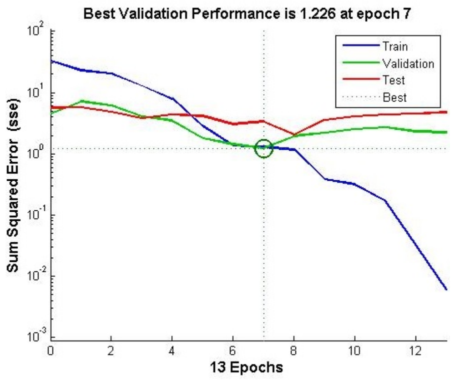


Figure 4.3: A network with a good fit.

In figure 4.1, underfitting is the problem. It can be seen that the training set has a high error. This performance was achieved when 5 hidden neurons where used. In figure 4.2, overfitting is the problem. Overfitting is not only the case when the validation error increases with the iterations as explained in chapter 3.6.1. There is also overfitting when the performance on the validation set is much lower than the performance on the training set. This is the case in figure 4.2. This performance was achieved when 25 hidden neurons where used. Figure 4.3 represents a good fit. This performance was achieved when 12 hidden neurons where used.

Distribution of the training data: The set of training samples must be divided into the three categories discussed in chapter 3.4. Training data, validation data and test data. This will be divided in respectively 70%, 15% and 15%. This is the most common way to do so.

4.3 Evaluate the Neural Network

When the network is trained, the performance of the network can be evaluated. In this section, different ways of training the neural network will be compared. First, it will be examined which parameters can be used best. Thereafter, it is examined how many seconds the neural network needs to recognize a cut-in before it actually occurs. As last, it will be examined how big the sample frequency of the training samples has to be.

4.3.1 Evaluation of the used parameters

To compare the importance of every different available parameter (lateral position, lateral velocity, width of the lane and angle of the car making a cut-in) it would be useful to train the neural network for different compositions of the parameters. However, training the network for the combination of lateral position, lateral velocity and lane width, does not lead to useful results. The performance plot of this network can be seen in figure 4.4.

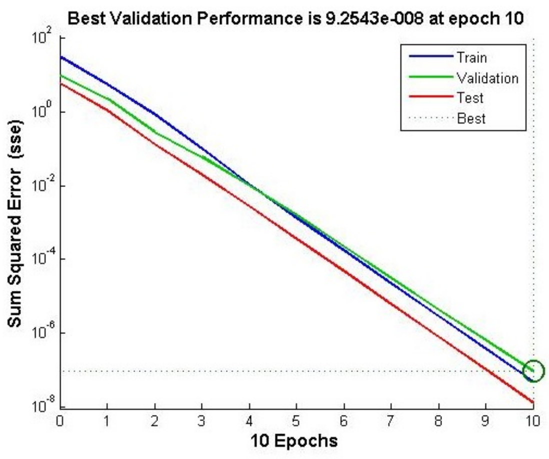


Figure 4.4: Performance plot for a network trained with the parameters: lateral position, lateral velocity, lanewidth.

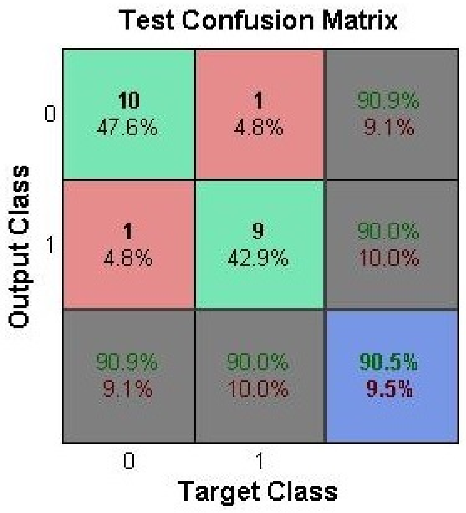
This performance plot gives the impression of a underfitting neural network. However increasing the number of hidden neurons does not lead to a better result.

There are two possible answers to this problem. It can be possible that the network does not have enough input values x_j. It can also be possible that there are not enough training samples available. The network does give a useful output when the angle of the car making a cut-in with regard to the host vehicle is added to the input values. When the sample frequency of this input vector is reduced, the network still has a useful output. From this information, it can be assumed that the the network has not enough training samples when presented with only lateral position, lateral velocity and lane width. For the rest of this section, the input vector will contain the lateral position, lateral velocity, lane width and angle of the car.

4.3.2 Evaluation of the time interval

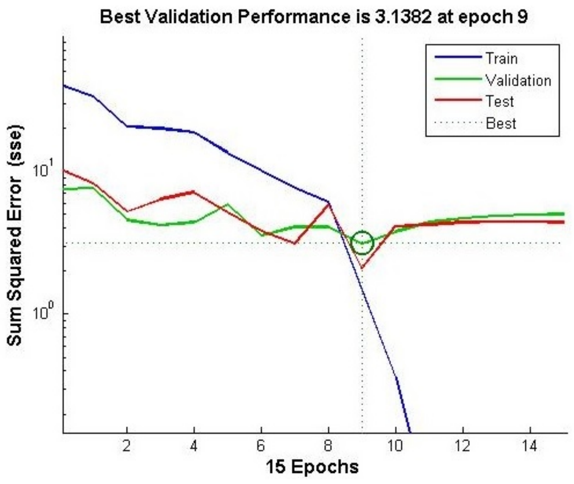
As mentioned before, the detection of the cut-in is most useful when it is detected before it actually happens. The usability of the neural network increases the earlier that a cut-in can be detected. When the neural network is provided with a time interval that is too small, the neural network will not be able to recognize a cut-in at all. A compromise has to be found. The minimal requirement [7]: the network must be able to recognize a cut-in when it is provided with a time interval of 4 seconds, ending at the moment the car is crossing the lane marker. First, the network is trained for this requirement.

Time interval of 4 seconds: The training input starts 4 seconds before the car crosses the lane marker and stops at the moment the car crosses the lane marker. 12 hidden neurons are used. The test confusion matrix can be seen in figure 4.5a.The test confusion matrix is only based on the training samples used for the test data. The rest of the confusion matrices can be seen in appendix C.1. These are the confusion matrices for the training data, validation data and for all the data combined. In the confusion matrix in figure 4.5a it can be seen that for the 15 input samples, 9 were correctly classified as a cut-in. 10 samples were correctly classified as a non-cut-in. 1 sample was detected as a cut-in but was a non-cut-in and the same happened the other way around. This lead to 90.5% of the test samples being correctly identified. This can be considered as a really good performance. In figure 4.5b the performance plot can be seen. From this plot it can be concluded that 12 hidden neurons lead to a good fit. All the confusion matrices and ROC-curves for this training input can be found in appendix C.1.



Test Confusion Matrix
Output Class
Target Class
0
1
10
47.6%
1
4.8%
90.9%
9.1%
1
4.8%
9
42.9%
90.0%
10.0%
90.9%
9.1%
90.0%
10.0%
90.5%
9.5%

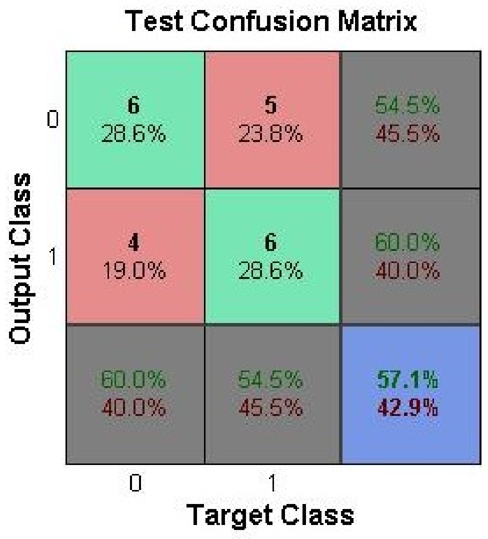
(a) Test confusion matrix.



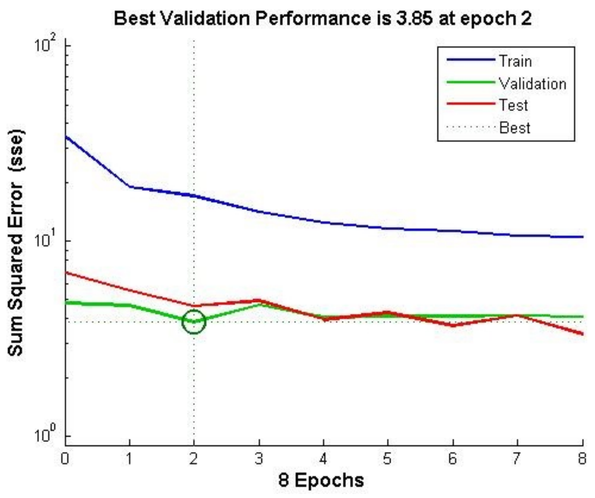
(b) Performance plot.

Figure 4.5: Evaluation plots for the time interval of 4 seconds. 12 hidden neurons. 40 measurements per second.

Time interval of 3 seconds: This training input starts 4 seconds before the car crosses the lane marker and stops one second before the car crosses the lane marker. 40 hidden neurons are used. The test confusion matrix can be seen in figure 4.6a. Of the 11 test samples that where non-cut-ins, 5 where detected as a cut-in. Of the 10 test samples that where cut-ins, 4 where detected as a non-cut-in. This leads to 57.1% of the test samples being correctly identified. For a neural network, this is a really poor outcome. In figure 4.6b, the performance plot can be seen. Underfitting can be detected in this plot. However, when the number of hidden neurons is increased, this does not change. This can be explained by the fact that a 3 second time interval is too short for the neural network to recognize a cut-in. This leads to the conclusion that this time interval is too short to recognize a cut-in. All the confusion matrices and ROC-curves for this training input can be found in appendix C.2.



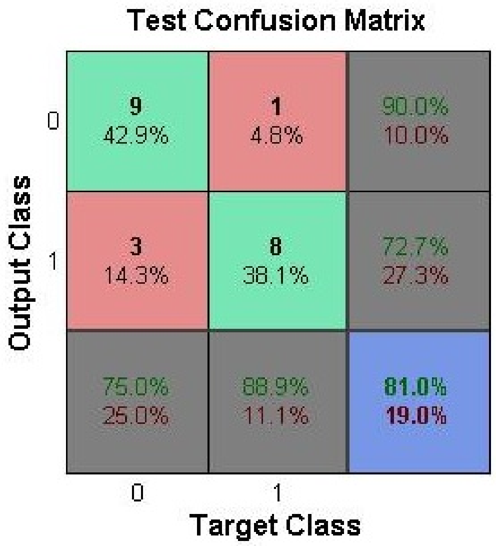
(a) Test confusion matrix.



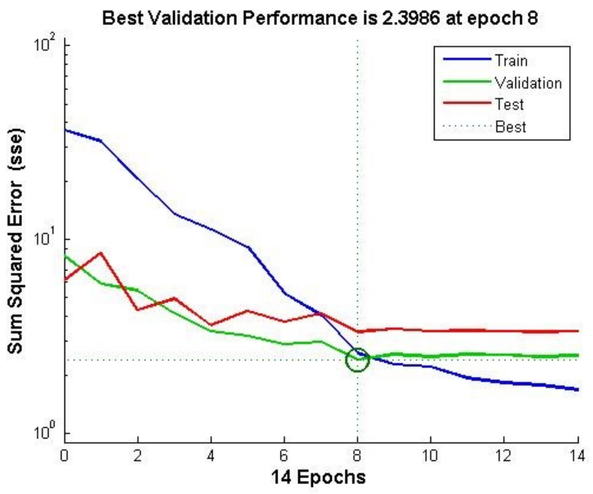
(b) Performance plot.

Figure 4.6: Evaluation plots for the time interval of 3 seconds. 40 hidden neurons. 40 measurements per second.

Time interval of 3.5 seconds: So, it is concluded that a time interval of 3 seconds is too short and a time interval of 4 seconds works really well. Therefore, the last time interval will be chosen in between these two intervals. The third time interval will start 4 seconds before the car crosses the lane marker and will stop 0.5 seconds before the car crosses the lane marker. In figure 4.7a, the test confusion matrix can be seen. For the 10 cars making a non-cut-in, 1 was classified as a cut-in. For the 11 cars making a cut-in, 3 where classified as a non-cut-in. This leads to 81.0% of the test samples being correctly identified. For a neural network this is still a reasonable performance, however it is a significantly lower result then the time interval of 4 seconds, which was correct for 90,5% of the test samples. In figure 4.7b the performance plot can be seen. From this plot it can be concluded that 30 hidden neurons lead to a good fit. All the confusion matrices and ROC-curves for this training input can be found in appendix C.3.



(a) Test confusion matrix.



(b) Performance plot.

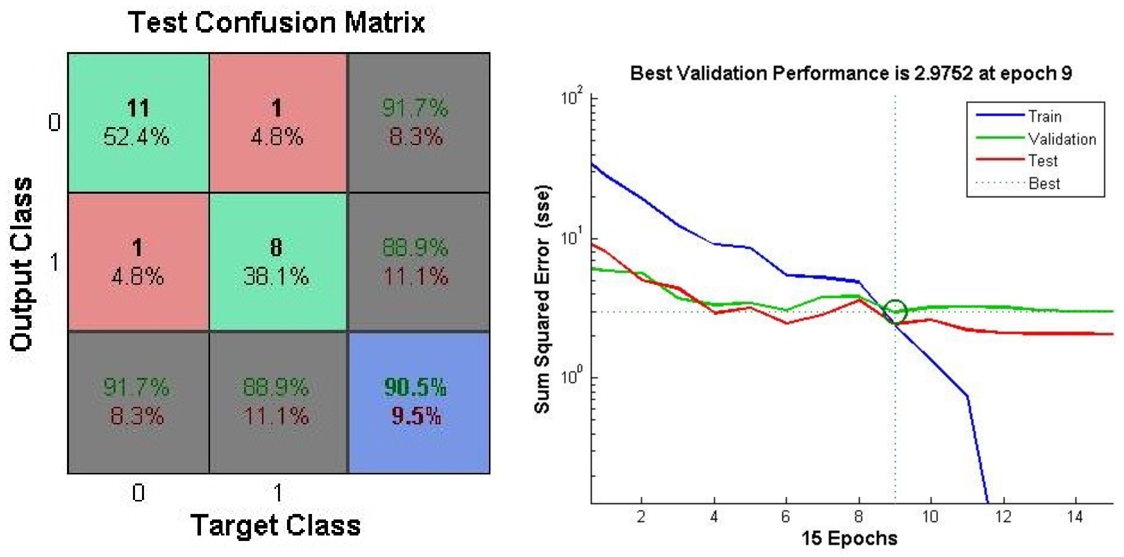
Figure 4.7: Evaluation plots for the time interval of 3.5 seconds. 30 hidden neurons. 40 measurements per second.

Depending on the preference, one could choose the time interval of 4 or 3.5 seconds. If a secure detection is chosen to be more important, it is wise to choose the time interval of 4 seconds. If a fast detection is chosen, one could choose the time interval of 3.5 seconds.

4.3.3 Evaluation of the sampling frequency

The optimal usage of the parameters and the time interval are known. One last variable that has to be optimized is the sampling frequency of the training data. The higher the sampling frequency, the more memory the network needs. When the sampling frequency becomes too low, the network could have too few input values to recognize the cut-in. Again, a compromise has to be found. The measurement frequency of the logfiles is 40 measurements per second. This is the maximum sampling frequency. This is also the frequency used up until now.

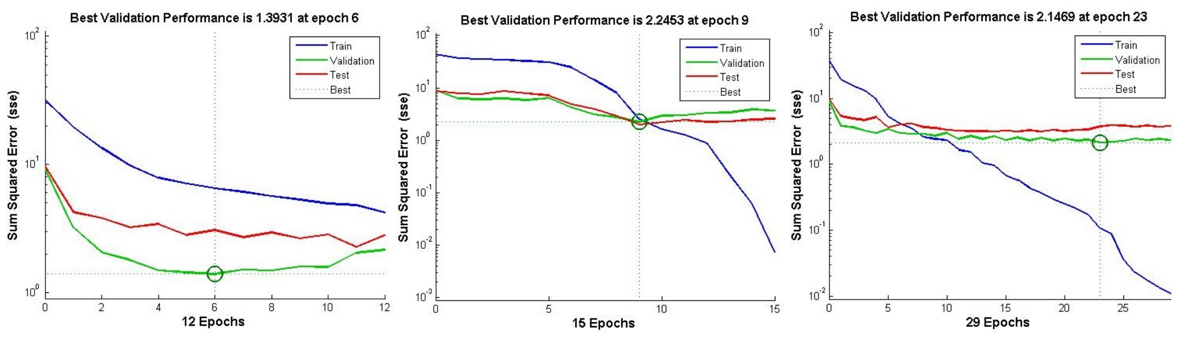
Sampling frequency of 20 measurements per second: When the sampling frequency is reduced by half, the performance of the neural network is not reduced. In figure 4.8a, the test confusion matrix can be seen. 11 of the 12 non-cut-in's where correctly detected as such. 1 was falsely detected as a cut-in. 8 of the 9 cut-ins where correctly detected as a cut-in. 1 was falsely detected as a non-cut-in. This leads to a performance of 90.5%. This is the same as the performance of the neural network trained with 40 measurements per second. This can be seen in figure 4.5a. The performance of a neural network varies every time it its trained. This is because of the starting values of $\boldsymbol{\theta}$. Thus, the performance of a network trained with 20 measurements per second is not necessarily always exactly the same as the network trained with 40 measurements per second. However, it can be concluded that the sampling frequency can be halved, without reducing the performance of the neural network.



(a) Test confusion matrix. (b) Performance plot.

Figure 4.8: Evaluation plots for the time interval of 4 seconds. 20 hidden neurons. 20 measurements per second.

Sampling frequency 16 measurements per second or less: When the network is provided with a sampling frequency of 16 measurements or less, a curious result surfaces. Every time the network is trained, the performance is remarkably different. The network underfits, overfits, or delivers a high performance of 90% or more. Three performance plots of the same neural network can be found in figure 4.9. This could be a result of too few input variables. So, to be certain of a reliable neural network, it is wise not to use less then 20 measurements per second.



(a) Underfitting. (b) Good fit. (c) Overfitting.

Figure 4.9: Performance plots of a underfitting, good fitting and overfitting network. Time interval of 4 seconds. 25 hidden neurons and 16 measurements per second.

Chapter 5

Conclusion

The main question of this thesis was: Is it possible to detect a cut-in by the use of a neural network? That question is easily answered. Yes it is possible to detect a cut-in by the use of a neural network. But what are the optimal conditions to gain the best performance?

To train this neural network, the input variables are to be determined. Four parameters are chosen to be used as input variables. First, the lateral position of the car making a cut-in with regard to the road. Second, the lateral velocity of the car making a cut-in with regard to the road. Third, the lane width of the lane the car making a cut-in was driving on. As last, the angle of the car making a cut-in with regard to the road. The sample frequency of these parameters is 40 samples per second. This can be reduced to 20 samples per seconds without influencing the performance of the neural network.

The detection of the cut-in is most useful when it is detected before it actually happened. The minimal requirement is an time interval of 4 seconds, ending when the car making the cut-in crosses the lane marker. When this requirement is used to train a neural network, it can reach a performance of 90.5%. This means that 90.5% of test samples are correctly identified as a cut-in or a non-cut-in. The second time interval tested is 3 seconds, ending one second before the car making the cut-in crosses the lane marker. A network trained with this time interval performs really poorly. The network reaches a performance of 57.1%. This is a useless result. The last interval tested is 3.5 seconds, ending half a second before the car making the cut-in crosses the lane marker. A neural network provided with this time interval can reach a performance of 81.0%. This is an acceptable result. Depending on the preference, one could choose the time interval of 4 or 3.5 seconds. If a secure detection is chosen to be more important, it is wise to choose the time interval of 4 seconds. If a fast detection is chosen, one could choose the time interval of 3.5 seconds.

There are a lot of ways to minimize the cost function of a neural network. The best ways to train this specific neural network, is using the Levenberg-Marquardt backpropagation algorithm to minimize the Sum of Squared Errors used as a cost function.

Chapter 6

Discussion and Recommendations

There still are a lot of features undiscussed. For example, are these the only parameters playing a role in the detection of a cut-in? Or, what is the importance of the number of training samples? There are two important reasons why these questions are not answered. The first and most important reasons is shortage of time. Because of a delay at the beginning of this graduation project, there are subjects left uninvestigated. The second reasons is a shortage of cut-in examples. There where only 70 examples of a cut-in available that where already manually obtained and ready to be used as cut-in examples. To increase this number, would be very time consuming and the decision is made to focus on the already existing examples.

So, a few recommendations to take into account when further research in this field is performed.

- Examining the influence of the number of training examples could be valuable. To obtain these extra training examples, cut-ins have to be detected manually. A algorithm to detect these cut-ins can be used, but the video's corresponding to the logfiles where these cut-ins are detected have to watched to be 100% sure that this cut-ins actually are cut-ins. Providing the network with False Positives could decrease the performance of the network drastically.

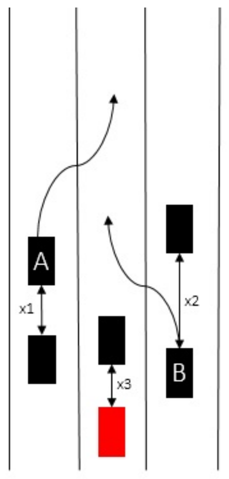


Figure 6.1: Definition of longitudinal distances between cars.

- Other parameters than the parameters already used could be the longitudinal distance between cars. In figure 6.1 different longitudinal differences are defined. The red vehicle is the host vehicle. When the longitudinal difference $x1$ becomes too small, changes are that vehicle A is going to make a cut-in. When distance $x2$ becomes too small, changes are that vehicle B is going to make a cut-in. When distance $x3$ is too small, the probability that a car is going to make a cut-in right in front of the vehicle becomes very small. However, this information can only be used as input parameters when this situations occur often enough in the set of cut-ins. When this is not the case, as it was in the Kiel_ Kassel logfiles, the neural network would be provided with random information and this could decrease the performance of the neural network.

- An important part of the detection of the cut-in is the time interval. In this thesis the results are discussed based on a manually adapted time interval. An other way to get information about the time interval the network needs to recognize a cut-in could be to increase the input. Instead of one output, stating if the training sample was a cut-in or a non-cut-in, there could be different outputs for different time periods. The neural network will give an output depending on the moment the cut-in was recognized. This is an way to get more insight about the time interval the neural network needs to recognize a cut-in.

Bibliography

[1] Mathworks Documentation: plotconfusion. `http://nl.mathworks.com/help/nnet/ref/plotconfusion.html`. Accessed: August 2016.

[2] Mathworks Documentation: plotperform. `http://nl.mathworks.com/help/nnet/ref/plotperform.html#zmw57dd0e24399`. Accessed: August 2016.

[3] How many hidden layers should I use? `http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-9.html`, March 2014. Accessed: August 2016.

[4] How many hidden units should I use? `http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html`, March 2014. Accessed: August 2016.

[5] A Neural Network in 13 lines of Python (Part 2- Gradient Descent). `http://iamtrask.github.io/2015/07/27/python-network-part2/`, January 2016. Accessed: August 2016.

[6] E-mail conversation with Magnus F. Nilsson, August 2016.

[7] Personal conversation with M. Ali and D. Jaller, August 2016.

[8] Henri P. Gavin. The Levenberg-Marquardt method for nonlinear leas squares curve-fitting problems. may 2016.

[9] Kevin Gurney. *An introduction to Neural Networks*. Number 0-203-45151-1. UCL Press, 1997.

[10] O.Tingleff K. Madsen, H.B. Nielsen. Methods for non-linear least squares problems. (2nd edition), april 2004.

[11] Laura Mauri Kelly H. Zou, A. James O'Malley. Receiver-Operating Characteristic Analysis for Evaluating Diagnostic Tests and Predictive Models. 2007.

[12] Manolis I. A. Lourakis. A Brief Description of the Levenberg-Marquardt Algorithm Implemented. Fabruary 2005.

[13] Andrew Ng. Machine learning, Gradient Descent. Stanford University Course.

[14] Andrew Ng. Machine learning, Gradient Descent for Linear Regression. Stanford University Course.

[15] Michael Nielsen. Neural Networks and Deep Learning. `http://neuralnetworksanddeeplearning.com/chap1.html`, January 2016. Accessed: August 2016.

[16] David M.W. Powers. Evaluation: From Precision, Recall and F-Factor to ROC, informedness, Markedness & Correlation. 2007.

[17] Lutz Prechelt. Early Stopping, but when?

[18] Ananth Ranganathan. The Levenberg-Marquardt Algorithm. June 2004.

[19] Ben Krose & Patrick van der Smagt.

Appendices

Appendix A

Levenberg-Marquardt derivation

To use the Levenberg-Marquardt backpropogation algorithm, a value for λ has to be found. An derivation for the step between equation 3.15 and equation 3.16 in chapter 3.5.2 can be found in this appendix. Equation 3.15 is:

$$C(\theta - \lambda) \approx \sum_{i=1}^{n} \Big(y(x_i) - a(x_i, \theta) - J_{i,j}\lambda \Big)^2. \tag{A.1}$$

The λ that minimizes this equation can be found by rewriting equation A.1 to the vector notation and set the derivative to zero. Write to the vector notation:

$$\mathbf{C}(\boldsymbol{\theta} - \boldsymbol{\lambda}) \approx \|\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i}) - \mathbf{J}\boldsymbol{\lambda}\|^{\mathbf{2}} \tag{A.2}$$

Which can be rewritten to:

$$(\mathbf{y}^{\mathrm{T}} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i}) - \mathbf{J}\boldsymbol{\lambda})^{\mathrm{T}}(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i}) - \mathbf{J}\boldsymbol{\lambda}) \tag{A.3}$$

With the use of the mathematical rules of the transposed matrix this leads to:

$$\begin{gathered} \Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big)^{\mathrm{T}} \Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big) \\ -\Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big)^{\mathrm{T}} \mathbf{J}\boldsymbol{\lambda} - \Big(\mathbf{J}\boldsymbol{\lambda}\Big)^{\mathrm{T}} \Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big) + \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\lambda} \end{gathered} \tag{A.4}$$

Which can be rewritten to:

$$\Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big)^{\mathrm{T}} \Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big) - \mathbf{2}\Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big)^{\mathrm{T}} \mathbf{J}\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\mathrm{T}}\mathbf{J}^{\mathrm{T}}\mathbf{J}\boldsymbol{\lambda} \tag{A.5}$$

Set the derivative of $\mathbf{C}(\boldsymbol{\theta} + \boldsymbol{\lambda})$ with regard to $\boldsymbol{\lambda}$ to zero:

$$-\mathbf{2}\Big(\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big)^{\mathrm{T}} \mathbf{J}\boldsymbol{\lambda}^{\mathrm{T}} \Big(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mathbf{J}\mathbf{J}^{\mathrm{T}}\Big) = 0 \tag{A.6}$$

Which can be rewritten to become equation 3.16 in chapter 3.5.2.

$$= \Big(\mathbf{J}^{\mathrm{T}}\mathbf{J}\Big)\lambda = \mathbf{J}^{\mathrm{T}}\Big[\mathbf{y} - \mathbf{a}(\boldsymbol{\theta}, \mathbf{x_i})\Big]. \tag{A.7}$$

Appendix B

Matlab code used to create training and target data

```
clear; clc;

%Information about the cut in (time, object number, number of the data,
    .. )
cutInDataInformation=xlsread('C:\work\cut in detection\cutinlogs\
    data_Kiel_Kassel_matlab.xlsx',1);

%path of the logdata
Path = 'C:\work\cut in detection\cutinlogs\Kiel_Kassel\roadestimation';
filePattern = fullfile(Path, 'CADS4_OTB882_20150112_104254_*_Replay.mat')
    ;
matFiles = dir(filePattern);


%select data files where a cut in occures and select cut in target.
nbrOfSamples = size(cutInDataInformation,1);
%number of training samples
timeSpan = 4;
%seconds before the car crosses the lane marker
timeSpanBeforeCutIn = 0.5;
%seconds substracted before the car crosses the lane marker
samplingFreq = 40;
%samples per second
rowsForLaneWidth = 1;
%number of inputs for rowwidth
endLatPos = (timeSpan-timeSpanBeforeCutIn)*samplingFreq+1;
%point in matrix lat pos ends
beginLatVel = (timeSpan-timeSpanBeforeCutIn)*samplingFreq+1;
%point in matrix lat vel begins
endLatVel = 2*((timeSpan-timeSpanBeforeCutIn)*samplingFreq)+1;
%point in matrix lat vel ends
beginCarAng = 2*((timeSpan-timeSpanBeforeCutIn)*samplingFreq)+1;
%point in matrix the angle of the car begins
endCarAng = 3*((timeSpan-timeSpanBeforeCutIn)*samplingFreq)+1;
%point in matrix the angle of the car ends
beginrowWidth = 3*((timeSpan-timeSpanBeforeCutIn)*samplingFreq)+1+
    rowsForLaneWidth;
%point in matrix for rowwidth


InputMatrix = zeros((3*((timeSpan-timeSpanBeforeCutIn)*samplingFreq)+1+
    rowsForLaneWidth),nbrOfSamples*2);
```

```

for k =  1:length(matFiles)

    if any(abs(k-cutInDataInformation(:,1))<1e-10);

        %empty

    else

        continue

    end
    matFilename = fullfile(Path, matFiles(k).name);
    cutInData = load(matFilename);
    %load data that contains a cut in


    indexOfObjectNumbers =find(~( cutInDataInformation(:,1) - k));
    %select car making the cut in
    for objIndex =  indexOfObjectNumbers(1):indexOfObjectNumbers(end)
        %takes care of one object that makes more than one cut in


        objNumber =  cutInDataInformation(objIndex,6);


        LatPosObject = cutInData.ReplayOutputLog.Obj.Info.
            DstLatFromMidOfLaneSelf (:,objNumber);
        %lateral position of the car making a cut in
        LatVelObject = cutInData.ReplayOutputLog.Obj.Estimn.VLat (:,
            objNumber);
        %lateral velocity of the car making a cut in
        time = cutInData.LogTime.hostlogTime;
        LaneWidth = cutInData.ReplayOutputLog.RoadPpty.LaneWidth;
        %lanewidth of ego lane
        longPositions = cutInData.ReplayOutputLog.Obj.Estimn.PosnLgt(:,
            objNumber);
        %longitudinal position of car making a cut in
        roadAngle = calculateRoadAngle(cutInData.ReplayOutputLog.RoadPpty
            , longPositions);
        %angle of the road (to hostvehicle) at the position of the car
            making a cut in
        carAngle = cutInData.ReplayOutputLog.Obj.Estimn.AgDir (:,
            objNumber);
        %angle of the car (to host vehicle) making a cut in
        carAngleadapted = carAngle - roadAngle;
        %angle of the car (to the road) making a cut in

        objTime = cutInDataInformation(objIndex,4);
        %time at the middle of the cutin
        [roundedTimeDiff, roundTimeIndex] = min(abs(time-objTime));
        %time where the car is crossing the lanemarker


        durationOfCutIn = (roundTimeIndex-(timeSpan*samplingFreq)):(
            roundTimeIndex-(timeSpanBeforeCutIn*samplingFreq));
                              %duration of the cut in,  seconds before andthe
              car crosses the line

        %enough sampels at beginning
        if roundTimeIndex > timeSpan*samplingFreq

            InputMatrix(1:endLatPos,objIndex) = LatPosObject(durationOfCutIn);
            %lateral position in matrix
            InputMatrix(beginLatVel:endLatVel ,objIndex) = LatVelObject(durationOfCutIn);
            %lateral velocity in matrix
            InputMatrix(beginCarAng:endCarAng, objIndex) = carAngleadapted(durationOfCutIn);
            %Car angle in matrix


                % too close to the beginning, fill with first data point
        else
                numberOfPointsBefore = roundTimeIndex;
                %Number of datapoints before the car is crossing the lanemarker
                numberOfFirstElement = find(durationOfCutIn >=0,1);
                %first datapoint
                tempLatPos = ones(numberOfFirstElement,1)*LatPosObject(10);
                %vector filled for latpos
                tempLatVel = ones(numberOfFirstElement,1)*LatVelObject(10);
                %vector filled for latvel
                tempCarAng = ones(numberOfFirstElement,1)*carAngleadapted(10);
                %vector filled for carangle
                mergedTempLatPos = [tempLatPos; LatPosObject(1:numberOfPointsBefore-(samplingFreq*timeSpanBeforeCutIn))];
                %vectors put together latpos
                mergedTempLatVel = [tempLatVel; LatVelObject(1:numberOfPointsBefore-(samplingFreq*timeSpanBeforeCutIn))];
                %vectors put together latvel
                mergedTempCarAngle = [tempCarAng; carAngleadapted(1:numberOfPointsBefore-(samplingFreq*timeSpanBeforeCutIn))];
                %vectors put together carangle
                InputMatrix(1:endLatPos,objIndex) = mergedTempLatPos;
                %lateral position in matrix
                InputMatrix(beginLatVel:endLatVel, objIndex) = mergedTempLatVel;
                %lateral velocity in matrix
                InputMatrix(beginCarAng:endCarAng, objIndex) = mergedTempCarAngle;
                %Car angle in matrix

        end
        InputMatrix(beginrowWidth, objIndex) = LaneWidth(roundTimeIndex);
        %lanewidth in matrix

    end
end
```

```


%create data that is not a cut in
 for k = 1:nbrOfSamples

         if k ~= 34  %something is wrong with file 34
        %empty
         else
         continue
         end


     matFilename = fullfile(Path, matFiles(k).name);
     cutInData = load(matFilename);

        %load data

     %select car on the left that stays there for at least 4 seconds
     carsLeft = cutInData.ReplayOutputLog.AccTgtAdjLLeft.Idn;
     [freq,objNumberCarLeft]=max(histc(carsLeft, 1:32));
     carLeftIndex = find(carsLeft == objNumberCarLeft, (timeSpan-
        timeSpanBeforeCutIn)*samplingFreq+1, 'first');


          %no car that stays for 4 seconds left, look on the right
          if freq < timeSpan*samplingFreq+1;
          carsRight = cutInData.ReplayOutputLog.AccTgtAdjLRight.Idn;
          [freq,objNumberCarRight]=max(histc(carsRight, 1:32));
          carRightIndex = find(carsRight == objNumberCarRight, (timeSpan
             -timeSpanBeforeCutIn)*samplingFreq+1, 'first');


           %when there is no car on the left and right, fill with zero's
           if freq < (timeSpan-timeSpanBeforeCutIn)*samplingFreq+1;
               filledwithzero = zeros((timeSpan-timeSpanBeforeCutIn)*
                  samplingFreq+1, 1);
               InputMatrix(1:endLatPos,k+nbrOfSamples) = filledwithzero;

           else

           latPosCarRight = cutInData.ReplayOutputLog.Obj.Info.
              DstLatFromMidOfLaneSelf(carRightIndex, objNumberCarRight);
           %Lateral position of the car on the right
           latVelCarRight = cutInData.ReplayOutputLog.Obj.Estimn.VLat (
              carRightIndex, objNumberCarRight);
           %Lateral velocity on the car on the right
           LaneWidthCarRight = cutInData.ReplayOutputLog.RoadPpty.
              LaneWidth;
           %lanewidth of ego lane
           longPositionsRight = cutInData.ReplayOutputLog.Obj.Estimn.
              PosnLgt(:,objNumberCarRight);
           %longitudinal position of car making a cut in
           roadAngleCarRight = calculateRoadAngle(cutInData.
              ReplayOutputLog.RoadPpty, longPositionsRight);
           %angle of the road (to hostvehicle) at the position of the
              car making a cut in
           carAngleCarRight = cutInData.ReplayOutputLog.Obj.Estimn.AgDir
               (:,objNumberCarRight);
```

```
            %angle of the car (to host vehicle) making a cut in
            carAngleadaptedCarRight = carAngleCarRight -
               roadAngleCarRight;
            %angle of the car (to the road) making a cut in


            InputMatrix(1:endLatPos,k+nbrOfSamples) = latPosCarRight;
            InputMatrix(beginLatVel:endLatVel, k+nbrOfSamples) =
               latVelCarRight;
            InputMatrix(beginCarAng:endCarAng, k+nbrOfSamples) =
               carAngleadaptedCarRight(carRightIndex);
            InputMatrix(beginrowWidth, k+nbrOfSamples) =
               LaneWidthCarRight(1000);



            end

         else
            latPosCarLeft = cutInData.ReplayOutputLog.Obj.Info.
               DstLatFromMidOfLaneSelf(carLeftIndex, objNumberCarLeft);
            %Lateral position of the car on the left
            latVelCarLeft = cutInData.ReplayOutputLog.Obj.Estimn.VLat (
               carLeftIndex, objNumberCarLeft);
            %Lateral velocity on the car on the left
            LaneWidthCarLeft = cutInData.ReplayOutputLog.RoadPpty.
               LaneWidth;
            %lanewidth of ego lane
            longPositionsLeft = cutInData.ReplayOutputLog.Obj.Estimn.
               PosnLgt(:,objNumberCarLeft);
            %longitudinal position of car making a cut in
            roadAngleCarLeft = calculateRoadAngle(cutInData.
               ReplayOutputLog.RoadPpty, longPositionsLeft);
            %angle of the road (to hostvehicle) at the position of the
               car making a cut in
            carAngleCarLeft= cutInData.ReplayOutputLog.Obj.Estimn.AgDir
               (:,objNumberCarLeft);
            %angle of the car (to host vehicle) making a cut in
            carAngleadaptedCarLeft = carAngleCarLeft - roadAngleCarLeft;
            %angle of the car (to the road) making a cut in


            InputMatrix(1:endLatPos,k+nbrOfSamples) = latPosCarLeft;
            InputMatrix(beginLatVel:endLatVel, k+nbrOfSamples) =
               latVelCarLeft;
            InputMatrix(beginCarAng:endCarAng, k+nbrOfSamples) =
               carAngleadaptedCarLeft(carLeftIndex);
            InputMatrix(beginrowWidth, k+nbrOfSamples) = LaneWidthCarLeft
               (1000);

           end

end
```

```
 %take only every 4th sample of the lateral velocity, the lateral
     position and the angle of the car.
 InputMatrixSmall = InputMatrix (1:4:endLatVel,:);




%create target data
targetdatacutin = ones (1, nbrOfSamples);
targetdatanocutin = zeros (1, nbrOfSamples);
targetdatatotal = [targetdatacutin , targetdatanocutin];
```

The function used to calculate the road angle.

```
function [roadAngle] = ...
    calculateRoadAngle(RoadPpty, longPositions) %#codegen
% RGF5_calculateRoadCoordinates calculates the coordinates for the road
% at a specific point. Expressed in the ego vehicle coordinate system.
%
% Inputs:
%   - StateVectors: The state vectors
%
%   - longPosition: Longitudinal position to evaluate at
%
% Outputs:
%   - StateVectors: Only used as output to create pointer in c-code
%
%   - roadCoordinates: Coordinates of the road at the longitudinal
   position
%
%   - roadAngle: Heading angle of the road at the longitudinal position
%
%   - roadCurvature: Curvature of the road at the longitudinal position
%

% Update information:
% ---------------------
% Created by: Alexander Larsson [CDSID alars11], dept. 96440
% Created: 2015-02-13
% Last changed by: Sanne Geboers
% Last update: 2016-08-30

% Copyright Volvo Car Corporation.
roadAngle = zeros(length(RoadPpty.OffsLat), 1);
for iSample = 1:length(RoadPpty.OffsLat)


longPosition = longPositions(iSample);

segmentLengths = RoadPpty.SegLen(iSample,:);

if longPosition >= segmentLengths(1)+segmentLengths(2)
    iSegment = 3;
```

```matlab
elseif longPosition >=  segmentLengths(1)
    iSegment = 2;
else
    iSegment = 1;
end

% Decide in wich segment the point is in.

eta_x1   = segmentLengths(1);
eta_x2   = segmentLengths(2);

psi_rel = RoadPpty.AgDir(iSample);
c0      = RoadPpty.Crvt(iSample);
c1_1    = RoadPpty.CrvtRate(iSample, 1);
c1_2    = RoadPpty.CrvtRate(iSample, 2);
c1_3    = RoadPpty.CrvtRate(iSample, 3);

if iSegment == 1
    %Adjust the length of the segment which the object belong too
    eta_xObj = longPosition;

    roadAngle(iSample) = atan(-psi_rel + 2*(c0/2)*eta_xObj + 3*(c1_1/6)*
       eta_xObj*eta_xObj);


elseif iSegment == 2
    %Adjust the length of the segment which the object belong too
    eta_xObj = longPosition - segmentLengths(1);

    roadAngle(iSample) = atan(-psi_rel + c0*(eta_x1+eta_xObj) + c1_1
       *((1/2)*eta_x1*eta_x1+eta_x1*eta_xObj) + c1_2*((1/2)*eta_xObj*
       eta_xObj));


elseif iSegment == 3
    %Adjust the length of the segment which the object belong too
    eta_xObj = longPosition - sum(segmentLengths(1:2));

    roadAngle(iSample) = atan(-psi_rel + c0*eta_x1 + c1_1*eta_x1*eta_x1/2
        + (c0+c1_1*eta_x1)*eta_x2 + c1_2*eta_x2*eta_x2/2 + ...
                         (c0+c1_1*eta_x1+c1_2*eta_x2)*eta_xObj + c1_3*
                            eta_xObj*eta_xObj/2);


end


end
end
```

Appendix C

Results for the evaluation of the time interval

C.1 Time interval of 4 seconds

The training input starts 4 seconds before the car crosses the lane marker and stops at the moment the car crosses the lane marker. 12 hidden neurons are used.

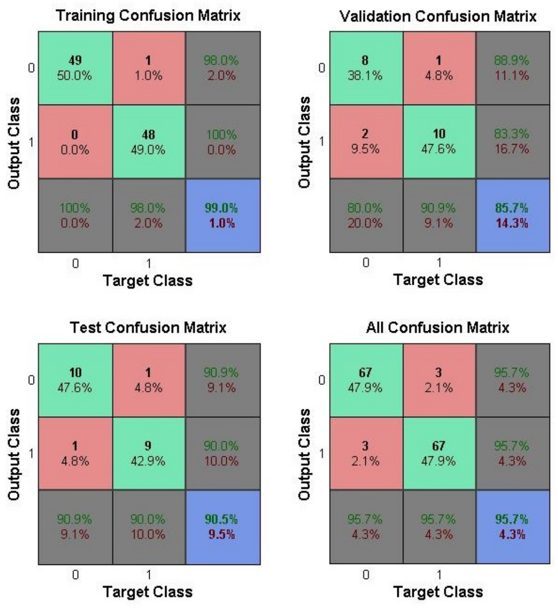


Figure C.1: All confusion matrix.

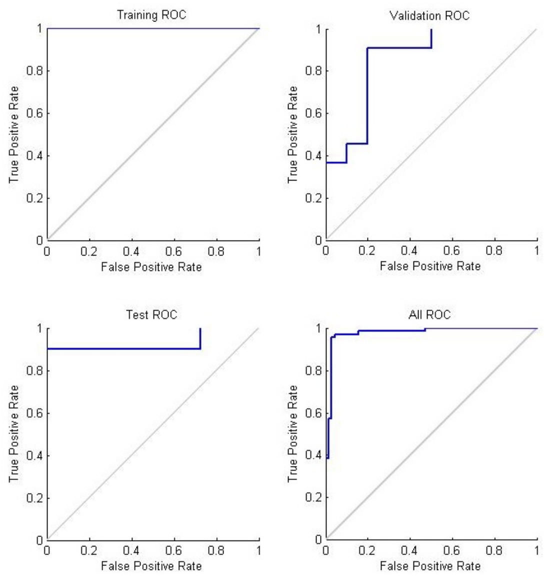


Figure C.2: Performance plot.

C.2 Time interval of 3 seconds

The training input starts 4 seconds before the car crosses the lane marker and stops one second before the car crosses the lane marker. 40 hidden neurons are used.

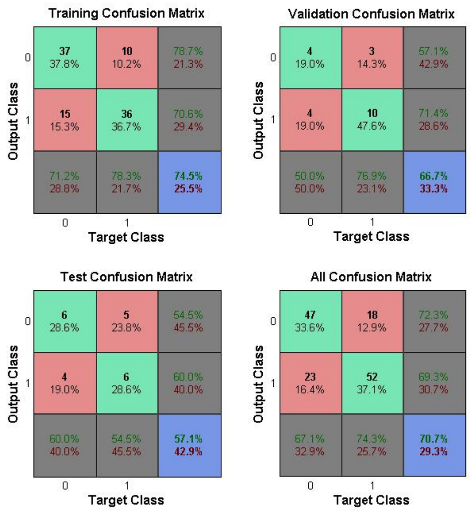


Figure C.3: All confusion matrix.

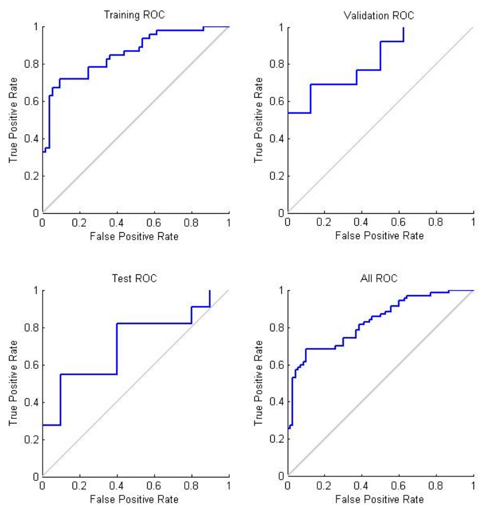


Figure C.4: Performance plot.

C.3 Time interval of 3.5 seconds

The training input starts 4 seconds before the car crosses the lane marker and stops half a second before the car crosses the lane marker. 30 hidden neurons are used.

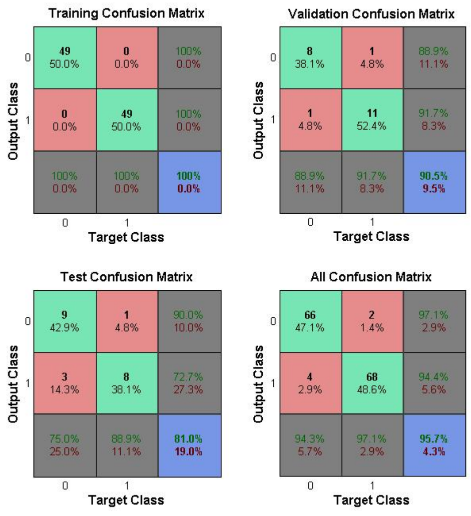


Figure C.5: All confusion matrix.

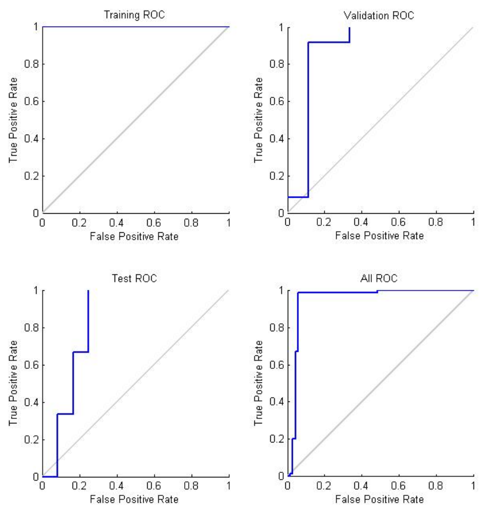


Figure C.6: Performance plot.

Appendix D

Extensive Job Description

In 2017 Volvo will deliver 100 self-driving cars as part of the Drive Me project. These cars will be able to drive autonomously on the ring road of Gothenburg, without requiring any driver supervision. To realize this project, the cars need different algorithms to recognize, and be able to react to, different traffic situations. One of these traffic situations is a car making a cut-in. A cut-in is a car making a lane change into the lane the host-vehicle is driving on, in front of the host vehicle. The host vehicle is the self-driving car in question. For the host-vehicle to react on a cut-in, it first needs to detect the cut-in.

To accomplish this, a neural network will be used. A neural network is a network with the ability to learn without being explicitly programmed. The assignment requires a lot of background information about neural networks. Prior to the assignment there will be an elaborate literature research. The rest of the knowledge will be acquired whilst working on the assignment.

The neural network has to be provided with input data, this data has to be generated. When the input data is generated the neural network can be trained an evaluated. The optimal conditions to train the neural network will be examined. This research will be presented in a thesis.