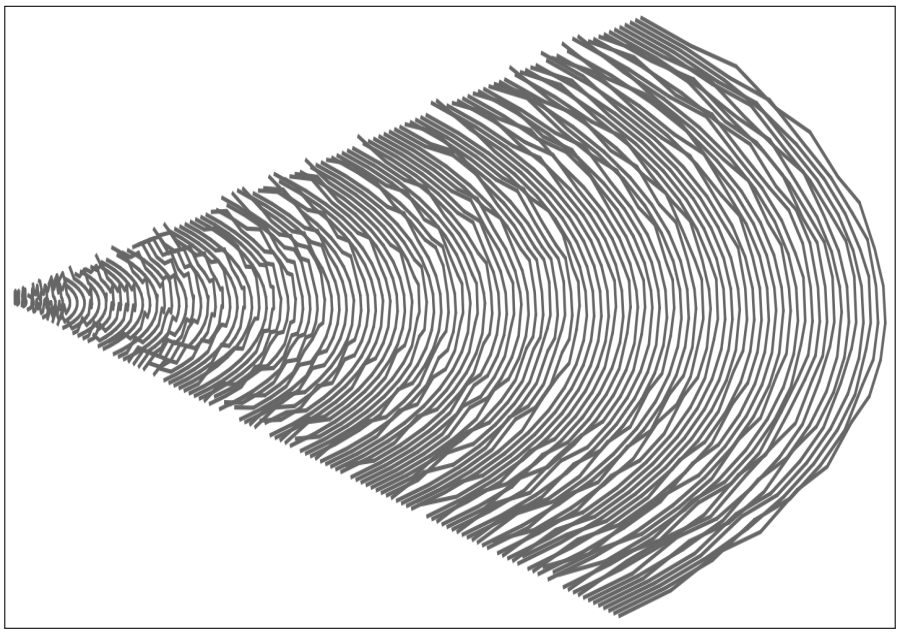
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Sensor Modeling with a focus on noise modeling in the context of Self-Driving Vehicles using Neural Network

Master's thesis in Complex Adaptive Systems

SIAMAK ESMI

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis TIFX05:2016

# Sensor Modeling with a focus on noise modeling in the context of Self-Driving Vehicles using Neural Network

**A quick step towards System Identification by Neural Network**

SIAMAK ESMI


CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2016

Sensor modeling with a focus on noise modeling in the context of Self-Driving Vehicles using Neural Network



Supervisors: Hang Yin, Federico Giaimo and Christian Berger, Department of Computer Science and Engineering
Examiner: Mats Granath, Department of Physics
Master's Thesis TIFX05:2016

Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Credence visualization in Cartesian coordinates, showing the credence of a sonar output detecting a high reflective object.

Typeset in LaTeX
Printed by [Name of printing company]
Gothenburg, Sweden 2016

Sensor Modeling with a focus on noise modeling in the context of Self-Driving Vehicles using Neural Network
Siamak Esmi
Department of Physics
Chalmers University of Technology

# Abstract

System Simulation has become an indistinct part of developments in which there are complexity involved. In this thesis, modeling of one of the most applicable sensors in automotive industry is carried out by applying a machine learning method known as neural networks. This report outlines a method of employing a combination of neural network techniques to model system behaviour; an applicable method compatible with any type of data. By integrating this sensor model into simulation environment (OpenDaVINCI), simulation test which plays an important role in testing of self-driving vehicles will become more realistic.

Keywords: System identification, neural network, proximity sensor, noise, self-driving vehicles

# Acknowledgements

I would like to appreciate Hang Yin and Federico Giaimo and Christian Berger for their precious support and technical advices. Also special thanks to F.V.Corswant and A.Karsolia who facilitate the experimentation process at ReVeRe lab.

Siamak Esmi, Gothenburg, September 2016

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

Sensors are often used to continuously monitor the status of an environment. These monitoring results are reported to an application for making decisions and answering user queries in different processes. For instance, for a vehicle to drive by itself, a sensor system is needed to perceive (detect, identify, localize) the environment around the vehicle. Various sensor network measurement studies have reported instances of transient noises in sensor readings [Sharma et al., 2010].

Environmental features will always play a dominant role in the type and prevalence of faults because they play a significant role in determining expected behavior. Environmental perturbations can also affect sensors in adverse ways [Ni et al., 2009].

Modeling Ultrasonic proximity sensors which consequently provides the ability to predict the sensor output is of an extensive interest since these sensors are widely being used in autonomous vehicles and basically in mobile robots. One particular rangefinder sensor, namely, *Devantech SRF08 UltraSonic Ranger* is chosen for this modeling based on its robustness, user friendly level, accuracy and low cost.

A modeling technique to some extent based on [Gutierrez-Osuna et al., 1998] is carried out which also compares different neural network architectures for non-linear regression/mapping.

The result of this modeling will be used as a "Noise Model" in automated simulations incorporated in OpenDaVINCI[1], which is an Open source Development Architecture for Virtual, Networked, and Cyber-Physical System Infrastructures. OpenDaVINCI is a compact middle-ware with several extended libraries for simulation and visualization.

This development environment is used on various self-driving vehicles at Chalmers/University of Gothenburg and the aforementioned sensor is the one which is employed in experimental miniature cars.

## 1.1 Background

Modeling is an important way of exploring, studying and understanding the world around. A model is a formal description of a system, which is a separated part of the world and describes certain essential aspects of a system [Horvath, 2003]. This

[1] http://www.opendavinci.org

separation is necessary when the influence of interaction between the system to be modeled and the rest of the world has to be reduced.

With respect to the aim of modeling, only some important aspects must be taken into account. This is why a model is always imperfect and only can "approximate" a system. It is always trivial to deal with models rather than systems since they are simpler. Nevertheless, the simplicity limits the validity of the model. Parsimony is another principle in modeling, it is formulated as the Occam's razor: "The most likely hypothesis is the simplest one that is consistent with all observations".

## 1.2 Goals

When it comes to "adaptive systems", the systems that can learn from the environment around, level of perception of surroundings is a determinant factor. The goal of this thesis is to systematically evaluate and improve model of a proximity sensor in the virtual test environment to significantly improve the level of "realism" in automated simulations incorporated in OpenDaVINCI.

The goal of this thesis is divided into the following research questions:

**RQ.1** Which parameters for deriving and evaluating representative fault/noise models for proximity sensors have to be considered?

**RQ.2** Which design considerations for a systematically experimentation and measurement of the vertically and horizontally field of view for proximity sensors need to be regarded?

These research questions seek to provide an answer for how this model can contribute in the development of real-time systems for self-driving vehicles.

## 1.3 Methodology

For this modeling purpose, the approach is to conduct an experiment and then estimate the process of proximity measurement by applying the "classic" and "modern" techniques of neural networks and benchmarking the performance of the exploited techniques in terms of accuracy and time efficiency.

### 1.3.1 Experiment Design

Clearly, more available information of a system results in a better model construction of that system. Lack of having solid and adequate information about attributes and structure of a system and relying just on observed data will lead to constructing a black-box or an input-output model. These constructions are highly based on relevant observations, hence, experiment design is a crucial part of a modeling. This designation determines which parameters can be measured and how largely they can affect the model.

### 1.3.2 Model Parameter Estimation

In order to estimate the desired parameters of a model, the relation between inputs and outputs of a system has to be formulated in a mathematical form. If the parameters to be estimated relate output variables to input variables, then the procedure can be used to model the system.

### 1.3.3 Artificial Neural Network Model

Using Artificial Neural Networks (ANNs) in system modeling is not the only approach. There are also other approaches to approximate non-linear mapping of different systems. However, among all these black box architectures, neural networks are far the most popular ones. The reasons – at least partly – come from the roots of neural networks: from their neurobiological origin, their ability to learn from examples and from the extremely good problem solving capability of "biological systems", which can be mimicked by artificial neural networks. The historical roots, however, would not be enough for this long time popularity. The real reasons come from the practical advantages of neural modeling.

In black box modeling, determining the proper size of a model structure is usually a hard task, and choosing improper size often leads to poor models. A too small model is not able to approximate a complex system well enough, a too large model with many free parameters, however, may be very prone to overfitting [Horvath, 2003].

In the next chapter, different Artificial Neural Network topologies and techniques which were used in this modeling are explained.

### 1.3.4 Validation

The final step is the validation of the model. To validate a system model, a proper criterion as a fitness of the model must be used. The choice of this criterion is extremely important as it determines a measure of the quality of the model. From the result of the validation one can decide whether or not the model is good enough for desired purpose.

### 1.3.5 Massive Parallel Programming Using CUDA

The Compute Unified Device Architecture, CUDA, is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). A performance comparison between GPU and multi-core CPU by implementation of neural networks is carried out in [Jang et al., 2008]. The result shows achieving computational times about 15 times faster than the analogous implementation using CPU and about 4 times faster than implementation on GPU alone. The probable advantages of using GPU for particular architectures of ANNs will be examined for this particular real-world problem.

## 1.4 Related Work

Neural network model is widely used in many areas, such as signal and image processing [Sun et al., 2013] and [Shen et al., 2013], wireless sensor network [Barabasi, 1999] and biological modeling [Maass, 1997].

In this thesis, the idea for experimentation is taken from [Gutierrez-Osuna et al., 1998] which also by some minor modification is used in [Gamarra-Diezma et al., 2015]. The researchers have utilized a very similar approach for data acquisition. The former has presented a probabilistic model of ultrasonic range sensors using feed-forward neural networks trained on experimental data where the latter focused on determination of the suitability of sonars in terms of sound cone, angle errors, crosstalk errors and field measurements.

In [Gutierrez-Osuna et al., 1998], regardless the experimentation, an approach of credence estimation is taken else than what is done in this thesis.

# 2
# Theory

Identification of nonlinear processes can insufficiently be modelled by conventional methods, therefore more sophisticated methods are required. Artificial networks of interconnected neurons with appropriate learning algorithms have been developed through last decades, promising a high rate of validity.

In this chapter, a basic mathematical approach for statistical modeling is provided, then fundamentals of ANNs with a focus on input-output mapping and different techniques with their applicability are presented.

## 2.1 Statistical Modeling

The goal of statistical modeling is to capture the general pattern of a relationship in a system and to present it as an equation. To derive the equation from acquired samples of a system, independent and dependent variables need to be known, also their dependency on time have to be studied. The equation produced by a modeling method can be considered as a mapping, because it allows the mapping of any point in the domain of independent variables onto a point in the domain of dependent variables.

Assuming that a system implements $f : R^N \rightarrow R$ mapping, the relation between the input and the output measurement data can be described as

$$y(i) = f(x(i)) + n(i) \tag{2.1}$$

where $n(i)$ is the noise, $\{x(i), y(i)\}_{i=1}^{i=p}$ are representing the input and output respectively.

This system will be modeled by a general model structure. The mapping of the model, $\hat{f(.)}$, will approximate in some sense the mapping of the system

$$y_M(i) = \hat{f}(\mathbf{x}(i), \mathbf{\Theta}) \tag{2.2}$$

where $y_M$ is the output of the model and $\mathbf{\Theta}$ is the parameter vector of the model structure.

Nevertheless, this mapping function produces some errors, since the dependent variable may not be equal to the actual result. This error can be measured in several ways such as the average of the squares of the difference between the estimated and the actual result. This approach yields to assigning more importance to those samples for which the error is larger.

In statistical modeling, there are two sources of error. As it is discussed in previous chapter, the first source of error is noise. In essence, the inaccuracies in the data, most likely introduced by the instrument itself or/and power resource or/and defective connections. It also includes inaccuracies due to the fact that independent variables do not contain all the information needed to determine the dependent variable; other factors not included in the model may play an important role.

Noise is not an inherent randomness or absence of causality in the world; rather, it is the effect of missing (or inaccurate) information about the world [Smith, 1993]. The second source of error is the fact that the mapping function may not have the same form as the target function. Target function is an idealized and unknowable function that express the true relationship between the independent and dependent variables.

The fields of Statistics and Neural Networks are closely related. The principle difference between the two fields is that historically statistic has focused on linear problems, which are relatively tractable, while Neural Networks has been forced to deal with nonlinearities.

## 2.2 Artificial Neural Network

As it is stated in [Haykin, 1994], a neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by network from its environment through a learning process.
- Interneuron connection strength, known as synaptic weights, are used to store the acquired knowledge.

Feed-forward neural network as a subset of fully connected neural networks, benefits of advantages which make it a good approach in input-output mapping.

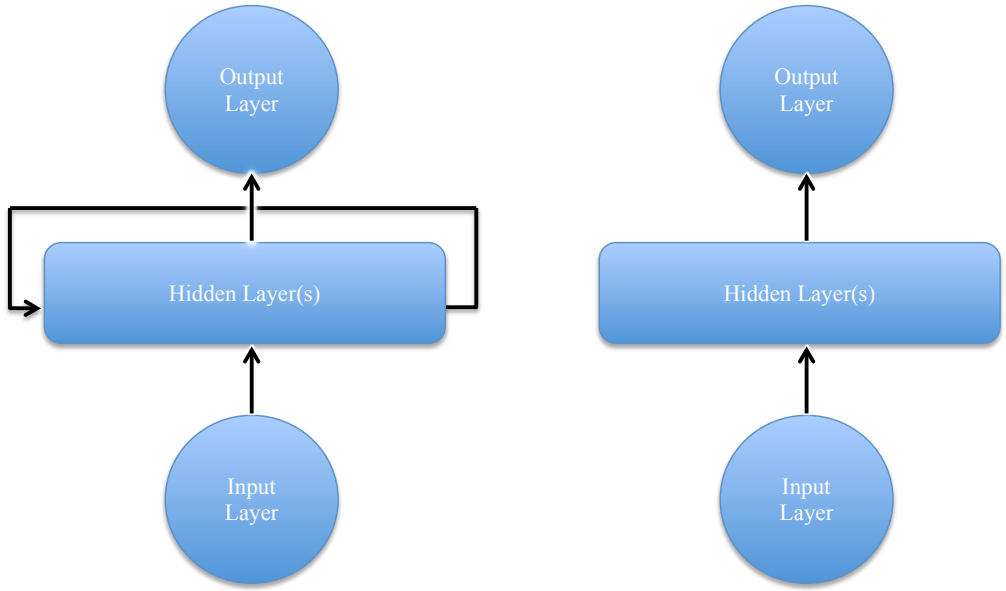
### 2.2.1 Feed-forward Neural Network

As the name implies, in feed-forward neural networks, neurons are arranged in layers in such a way that input samples are fed forward to the input layer and outputs are produced in the output layer as it is shown in Figure 2.1. The layer(s) in between have no connection with the external world, hence they are called the hidden layer(s). Each neuron in one layer is connected to every neurons in next layers but there is no connection among neurons in a same layer. The number of hidden layers and the number of neurons in each hidden layer depend on the problem in hand. The usual way of determining mentioned numbers is by trial and error. More difficult or strictly, higher degrees of freedom considered in the problem, the greater size of the neural network will be required.

### 2.2.2 Feed-back Neural Network

Basically, feed-back or recurrent neural networks (RNNs) are neural networks with one or more feedback loops, from either the output neurons or hidden neurons to the input layer as it is shown in Figure 2.1. These loops enable neural networks to do temporal processing and learn the sequences, e.g., perform sequence recognition/reproduction or temporal association/prediction.

The difference between feed-forward and recurrent ANN may seem trivial, the implications for sequence learning are rather extensive. The equivalent result to the universal approximation theory for feed-forward NN is that an RNN with a sufficient number of hidden units can approximate any measurable sequence-to-sequence mapping to arbitrary accuracy [Hammer, 2000].




(a) A Recurrent Neural Network model (b) A feed-forward Neural Network model

**Figure 2.1:** A general feed-foward neural network model versus a recurrent one

There are two major functional uses of recurrent networks:

- Associative memories
- Input-output mapping networks

Utilizing a feed-forward neural network with aforementioned loops, the ability of nonlinear mapping of feed-forward neural network can be exploited beside some form of *memory*. This added property, comes from the fact that more information of previous input is added to the current input and consequently is being fed to the neural network.

### 2.2.3 Learning Process

A Neural Network which is made up of an interconnection of nonlinear neurons, is itself nonlinear, however, it is capable of solving both linear and nonlinear problems.

As it is shown in Figure 2.2, the equation

$$y = \Phi\left(\sum_{i=1}^{N} w_i x_i + b\right) \tag{2.3}$$

represents a mathematical description of a neuron. In this example, the input vector is given by $\mathbf{x} = [x_1, x_2, x_3]^T$, whereas $\mathbf{w} = [w_1, w_2, w_3]^T$ is referred to as the *weight* vector and $b$ is the *Bias*. The function $\Phi$ is an *Activation* or *Squashing* function which usually is continuous. This function keeps outputs in a certain desired range.

The principle that a network of neurons follows, which consequently leads to an approximation of a process based upon the provided information, is not a complex one, yet it simply presumes an arbitrary approximation and then tries to tune it up to a more robust approximation. In order to reach that, there are number of methods and techniques which optimize either of the pre-assumptions and/or fine-tuning.

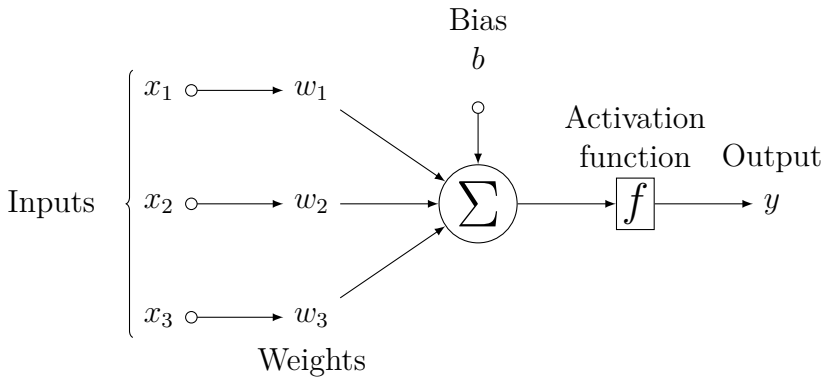



**Figure 2.2:** A simple structure of a neuron

#### Activation Function

The most common activation functions are *Sigmoid* and *Hyperbolic tangent* which have the feature of non-linearity. Sigmoid has the property of *Step* function with an added region for uncertainty and yields output values in the range $[0, 1]$.

Sigmoid function is given by the relationship below.

$$\sigma(t) = \frac{1}{1 + e^{-\beta t}} \tag{2.4}$$

where $\beta$ is in the range of $[0, 1]$.

Simplicity of derivative calculation of Sigmoid function is another property which distinguishes this function from other activation functions.

Yielding output values in the range $[-1, 1]$, hyperbolic tangent function is related

to Sigmoid function by the following linear transform

$$tanh(t) = 2\sigma(2t) - 1 \tag{2.5}$$

This means that these two functions are equivalent as activation functions in a neural network with a hidden layer of activation function. As it is shown in Figure 2.3, the difference is the output range, the factor that determines the proper one for the problem in hand. There are other activation functions which are being used for different purposes such as classification task which is out of the scope of this work.

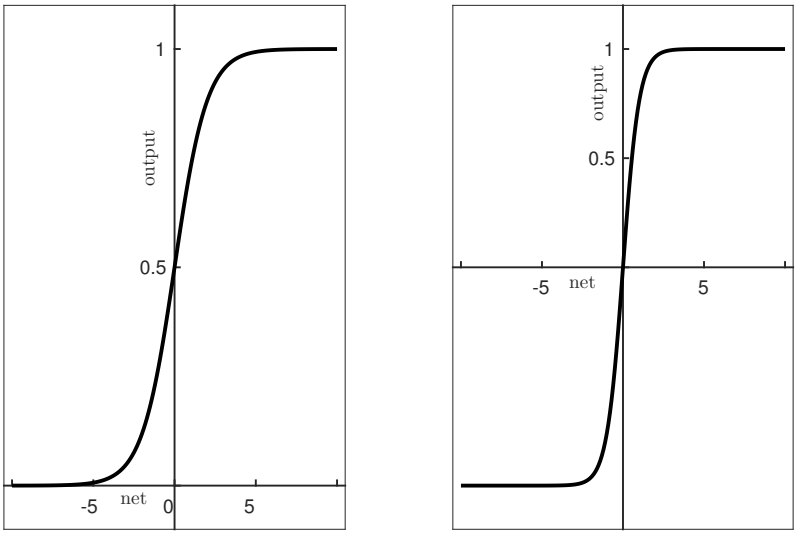



**Figure 2.3:** For $\beta = 1$; left: Sigmoid non-linearity squashes real values to $[0, 1]$, right: Hyperbolic tangent non-linearity squashes real values to $[-1, 1]$

### Weight & Bias

The procedure used to perform the learning process is called a *learning algorithm*, the function of which is to modify the synaptic weights of the network in an orderly fashion to attain a desired design objective [Haykin and Network, 2004]. The synaptic weights, which are learnable parameters of a NN, usually are randomly initiated to small values, evenly distributed around zero which refers as symmetry breaking. If two hidden units have exactly the same bias and exactly the same incoming and outgoing weights, they will always get exactly the same gradient, consequently, they can never learn to be different features [G. Hinton and Swersky, 2012].

Weights can get adapted after feeding each sample (online/sequential learning), or once the whole training samples or a part of that is fed to the neural network, the latter is called *batch* learning.

### Saturated Sigmoid

Using sigmoid as activation function has its own disadvantages which leads to a slow convergence and it occurs most likely when the variance of input features is high. Referring to [Glorot and Bengio, 2010], researchers found out that sigmoid activation function is not suitable for a deep NN with random initialization of weights

because of its mean value, which can drive especially the top hidden layer into saturation. Nevertheless, using the method they recommend should be investigate for shallow NN to see its effect on the convergence speed. When activation of a neuron saturates at either zero or one, the gradient of *cost function* at these regions is almost zero. The term *vanishing gradient* is derived from the fact that this very small value (almost zero) as a local gradient will be multiplied to the gradient of this gate's output for the whole objective during *backpropagation*, which will cause that almost no signal will flow through the neuron to its weight and recursively to its data. However, as it is explained in [He et al., 2015], there are other methods for weight initialization which avoid reducing or magnifying the magnitudes of input signals exponentially and keep the data flow to the latter layer with a mean of zero.

### Backpropagation

In *supervised learning*, which is a popular paradigm of learning, the modification of synaptic weights of a neural network is performed by applying a set of labeled training samples, i.e. *optimising* a *cost function* on the training set

$$E(w) = \sum_{i=1}^{N} E_i(w) \tag{2.6}$$

where the parameter $w$ which optimizes the cost function $E_i(w)$, is to be estimated by means of the $i-th$ input pattern in training set. A very common cost function is *Mean Squared Error* (MSE), which has the following calculation equation:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.7}$$

This equation assesses the quality of the predictor $\hat{y}$ in predicting $y$.

### Gradient Descent

In the case of complex non-linear functions computed by NN, there is no practical way of directly calculating the optimal values of the weights [Smith, 1993]. The simplest method is known as *gradient descent*, which its algorithm is to repeatedly take a small, fixed-size step in the direction of the negative error gradient of the cost function. This method has the following iteration:

$$w_t \leftarrow w_{t-1} - \eta \nabla_{w_{t-1}} E(w_{t-1}) \tag{2.8}$$

where $\eta$ is the step size or *learning rate* in the range of $[0, 1]$. The learning rate determines the size of the steps we take to reach a minimum and usually is set to 0.01.

In batch gradient descent, the gradient of the cost function is computed for the entire training data set, while stochastic gradient descent (SGD) in contrast performs a parameter update for each training example (pattern). This more frequent update with a high variance causes the objective function to fluctuate heavily and

consequently, enables it to jump to another potentially better local minima. Nevertheless, by mini-batch gradient descent, it is possible to take advantage of the both mentioned variants. Mini-batch gradient descent performs a weight update for every mini-batch of training examples.

For an efficient calculation of the gradient, a technique known as *backpropagation* is being used. The name backpropagation is derived from the process of propagating the error information backward from the output nodes to the hidden ones to optimize the weights. Backpropagation provides a way of using examples of a target function to find the coefficients that make a certain mapping function approximate the target function as closely as possible. This technique is basically a repeated application of chain rule for partial derivatives.

### Nesterov's Accelerated Gradient

A major issue with gradient descent algorithm in non-linear problems is that it easily gets stuck at local minima. By addition of a *momentum* term which simply adds a fraction of the previous weight update to the current one, the issue can get mitigated [David E. Rumelhart, 1986].

The momentum, which effectively adds inertia to the motion of the algorithm through weight space, helps to escape from local minima and hence, speeds up convergence. By introducing a momentum vector $m$

$$m_t = \mu\, m_{t-1} + \nabla_{w_{t-1}} E(w_{t-1}) \tag{2.9}$$

where $\mu$ is an exponential decay factor (momentum coefficient) in the range of $[0, 1]$, this method has the following iteration in its "standard" form

$$w_t \leftarrow w_{t-1} - \eta\, m_t \tag{2.10}$$

where $\eta$ is the learning rate.

Momentum coefficient determines the relative contribution of the current gradient and earlier gradients to the weight change.

Learning rate is an important term which determines the magnitude of changes in weights at each step. The importance of choosing a proper learning rate is that high such that choosing a small one will delay the convergence and choosing a large one will cause divergence in the algorithm.

Gradient descent is the only learning method which has been mathematically proven to converge on the set of weights producing minimum error. Nevertheless, *adaptive learning rate* is a technique which benefits of several advantages but the guarantee of convergence. This method is much faster than gradient descent and does not suffer of getting stuck at local minima. Having the learning rate set to an arbitrary value, if the direction in which the error decreases at this weight change is the same as the direction in which it has been decreasing recently, make the learning rate larger; if the direction in which the error currently decreases is the opposite of the direction in which it has been decreasing recently, make learning rate smaller [Smith, 1993].

*Adaptive sub-gradient descent* known as AdaGrad [J. Duchi, 2011], slows down learning along dimensions that have already changed significantly and speeds up learning along dimensions that have only changed slightly by dividing the learning rate of every step by the $l2$norm of all previous gradients. The weight update has the following iteration

$$w_t \leftarrow w_{t-1} - \eta \frac{\nabla_{w_{t-1}} E(w_{t-1})}{\sqrt{n_t} + \epsilon} \tag{2.11}$$

where $n_t$ is the norm vector $(\nabla E(w)^2)$ and $\epsilon$ is in the range of $[0, 1]$.

The recommended values for learning rate and $\epsilon$ are 0.01 and $1e^{-08}$ respectively.

However, as an issue in Adagrad, norm vector eventually becomes so large such that prevent reaching the local minima. As an extension to Adagrad, a per-dimension learning rate method for gradient descent is called *Adadelta*. The method dynamically adapts over time using only first order information and has minimal computational overhead beyond vanilla stochastic gradient descent. The method requires no manual tuning of a learning rate and appears robust to noisy gradient information, different model architecture choices, various data modalities and selection of hyperparameters [Zeiler, 2012].

In this method, the sum of gradients is recursively defined as a decaying average of all past squared gradients instead of storing previous squared gradients. It has the following iteration

$$w_t \leftarrow w_{t-1} - \frac{\mathrm{RMS}(\Delta w_{t-1})}{\mathrm{RMS}(\nabla E(w_t))} \nabla E(w_t) \tag{2.12}$$

In order to make adaptive learning rate technique perform better, an unpublished method called *RMSProp* can be employed which allows the model to continue to learn indefinitely. Technically, root mean square propagation, is an optimizer that utilizes the magnitude of recent gradients to normalize the gradients, i.e. divides the gradient by a running average of its recent magnitude [Tieleman and Hinton, 2012] as follows

$$\mathrm{MS}(w_t) = \nu\, \mathrm{MS}(w_{t-1}) + (1 - \nu)(\nabla E(w_t))^2 \tag{2.13}$$

$$w_t \leftarrow w_{t-1} - \frac{\eta}{\sqrt{\mathrm{MS}(w_t))} + \epsilon} \nabla E(w_t) \tag{2.14}$$

where $\nu$ is a decaying mean recommended to be set to 0.9.

In order to take advantage of properties of both momentum and RMSProp, *Adam* is introduced in [D. Kingma, 2014] which computes adaptive learning rates for each parameter. In adaptive moment estimation, performance of optimization is improved by combining momentum (using a decaying mean instead of a decaying sum) with RMSProp. By denoting $m_t$ as the estimates of the first moment (the mean) and $v_t$ as the second moment (the un-centered variance) of the gradients

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla E(w_t) \tag{2.15}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \nabla E(w_t)^2 \tag{2.16}$$

where $\beta_1 = 0.9$ and $\beta_2 = 0.999$, the authors of Adam computed bias-corrected estimates such that

$$\hat{m_t} = \frac{m_t}{\beta_1^t} \tag{2.17}$$

$$\hat{v_t} = \frac{v_t}{\beta_2^t} \tag{2.18}$$

consequently, the update iteration has the form of

$$w_t \leftarrow w_{t-1} - \eta \frac{\hat{m_t}}{\sqrt{\hat{v_t}} + \epsilon} \tag{2.19}$$

where the recommended learning rate in Adam is 0.001.

*Nesterov mometum* [Nesterov, 1983], is a slightly different version of standard momentum which is exploited in *Nesterov's accelerated gradient* [Sutskever et al., 2013]. This accelerated gradient is a first-order optimization method with better convergence rate guarantee than gradient descent in certain situations by a minimal increase in complexity and on contrary a drastic increase in performance. In Nesterov's accelerated gradient (NAG)[Dozat, 2014], by denoting $g_t$ as $\nabla E(w_{t-1})$

$$\hat{g_t} = \frac{g_t}{1 - \prod_{i=1}^{t} \mu_i} \tag{2.20}$$

$$m_t = \mu m_{t-1} + (1 - \mu) g_t \tag{2.21}$$

$$\hat{m_t} = \frac{m_t}{1 - \prod_{i=1}^{t} \mu_i} \tag{2.22}$$

$$n_t = \nu n_{t-1} + (1 - \nu) g_t \tag{2.23}$$

$$\hat{n_t} = \frac{n_t}{1 - \nu^t} \tag{2.24}$$

$$\hat{m_t} = (1 - \mu_t)\hat{g_t} + \mu_{t+1}\hat{m_t} \tag{2.25}$$

where $\nu = 0.999$, $\epsilon = 1e^{-08}$, and a momentum schedule given by

$$\mu_t = \mu(1 - 0.5 \times 0.96^{\frac{t}{250}}) \tag{2.26}$$

where $\mu = 0.99$, the weight update are calculated as follows

$$w_t \leftarrow w_{t-1} - \eta \frac{\hat{m_t}}{\sqrt{\hat{n_t}} + \epsilon} \tag{2.27}$$

### Pre-Processing

Activation functions used in neural networks are centred around a certain value in their output space; as it is depicted in Figure 2.3, the hyperbolic tangent is centered around 0 and sigmoid is centered around 0.5 by setting $\beta = 1$. Hence, transformation or *pre-processing* of input samples (input to the network) is necessary for an efficient mapping. This pre-processing of input samples matches them to the range of the activation function which is being used. For this purpose, several operations can be performed such as *normalisation* to zero variance and unity standard deviation or *rescaling* based on minimum and maximum values of input samples.

### Generalisation

Training the NN proceeds iteratively and each iteration is called an *Epoch*. Many epochs are usually required before the weights gradually converge to an optimal set of values. With input-output mapping, the real goal is to optimise performance on a test set of untrained data (validation set). The issue of whether training set performance carries over to the test set is referred to as *generalisation* and is of fundamental concepts in machine learning [Bishop, 2006]. Generally, the larger training set, the better generalisation, but in the case of small training sets, there are also methods that mitigate the undesired effect of it.

During the training phase, the error rate typically decreases at first on both training and validation data sets, but after a certain point it begins to rise on the validation sets, while continuing to decrease on the training set. This behaviour, is known as *overfitting*. There are number of methods to prevent this behaviour.

*Regulariser* are one of the proposed methods to improve the generalisation to prevent overfitting by applying penalties on layer parameters or layer activity during weight optimization.

A different way to constrain a network, and thus decrease its complexity, is to limit the growth of the weights through some kind of *weight decay*. It can be realized by adding a term to the cost function that penalizes large weights, as equation below

$$E(w) = E_0(w) + \lambda \left( \sum_i w_i^2 \right) \tag{2.28}$$

where $E_0$ is unregularized cost function and $\lambda$ is a parameter governing how strongly large weights are penalized [Anders Krogh, 1992]. Applying this method simply prevents the weights from growing too large unless it is really necessary.

*Lasso* or $l1$norm has the form of

$$E(w) = E_0(w) + \lambda \left( \sum_i |w_i| \right) \tag{2.29}$$

which in essence does both continuous shrinkage and automatic variable selection simultaneously [Tibshirani, 1996].

*Elastic net*, is the other regularization and variable selection method proposed in [Zou and Hastie, 2005], which is particularly useful when the number of predictors is much bigger than the number of observations.

Another method is *early stopping*. In this method, after assigning a part of input data as validation set, all stopping criteria are then tested on the validation set instead of the training set. The 'best' weight values are also chosen using the validation set.

*Dropout* is another technique for addressing overfitting. The key idea is to randomly drop units (along with their connections) from the neural network during training, hence, this prevents units from co-adapting too much [Srivastava et al., 2014].

In this thesis two different in nature NN are put to examination for benchmarking; a feed-forward and a feed-back NN.

### 2.2.4 MLP

Multi Layer perceptron (MLP) is a feed-forward neural network with one or more layers between input and output layer. This type of network is trained with the backpropagation learning algorithm. MLPs are widely used for pattern classification, recognition, prediction and approximation. The learning algorithm is more and less similar to what mentioned above.

### 2.2.5 Long-Short Term Memory

Long-short term memory (LSTM) NN as is introduced in [Schmidhuber, 1997], is a RNN capable of learning long term dependencies by combining fast training with efficient learning. Further on, it was improved with forget gates and peephole connections [F. A. Gers and Cummins, 2000]. Added peephole connections enables gate layers to get the state of the cells. By coupling forget and input gates, the decision of forgetting or remembering new information is being made.

Hidden layer in LSTM NN consists of memory block cell assemblies. Each memory block is composed of memory cell units that retain state across time-steps, as well as three types of specialized gate units that learn to protect, utilize, or destroy this state as appropriate. The LSTM training algorithm back-propagates errors from the output units through the memory blocks, adjusting incoming connections of all units in the blocks, but then truncates the back-propagated errors [Monner and Reggia, 2012]. What distinguishes LSTM NN from "standard" RNN is that LSTM NN "remember" information for long periods of time by means of memory block cells.

## 2.3 Sensor and Noise

Data noise that have been observed in real deployments which cause the faulty sensor readings to deviate from the perfect pattern exhibited by non-faulty sensor readings, are generally generated by either internal or external factors; such as battery failure or defective connections.

According to [Sharma et al., 2010] and [Ni et al., 2009], the fault in real-world sensor data can be categorised in four groups. These four groups are defined in [Fang and Dobson, 2013] as below:

- SHORT: A sharp momentary change in the measured value between normal consecutive readings. Hardware failures like fault in the analog-to-digital convert board may lead to short faults;

- NOISE: sensor readings exhibit an unexpectedly high amount of variation for a period of time. The noisy variance is beyond the expected variation of the underlying phenomenon. Usually high noise is due to a hardware failure;

- CONSTANT: also known as "Stuck-at" fault. The readings remain constant for a period of time greater than expected. The reported constant value usually is out of the possible range of the expected normal readings and uncorrelated to the underlying physical phenomena. Constant faults occur due to clipping, hardware failure or low battery;

- CALIBRATION: sensor readings may have offsets or incorrect gain, rendering reported data deviating from the true value. Drift faults occur when the offset or gain change with time.

Mentioned faults are of innate disadvantages which might occur in any sensor driven system. Nevertheless, there are other typical measurement errors needed to be taken into account when it comes to beam-based proximity sensors. For instance, interferences on sonar sensors known as cross-talk, leading to irrelevant measurements which are the result of reflected beams from an obstacle at completely different paths. Even when the beam comes from a real obstacle, there will be still noise in that measurement. However, the consistency of manufacturing tolerance and claimed range of the sensor also should be evaluated.

In order to develop a fault-tolerant perception architecture, the reliability of sensor readings needs to be systematically tested. The concept of fault tolerant systems refers to the systems that are able to compensate faults in order to avoid unplanned behaviours [Isermann, 2011].

# 3
# Methods

In this chapter the employed proximity sensor for this work is introduced. Considered parameters for deriving and evaluating representative noise model for the sensor beside the designation of the experiment are outlined to address the RQ.1 and RQ.2. The pre-processing of the acquired data samples and the Neural Network model are also explained.

## 3.1 Ultrasonic Sensor

Ultrasonic transducers are one of the most popular sensors which have widely been used in robots and automated driving. These inexpensive sensors, convert ultrasound waves to electrical signals and vice versa. By measuring the time of flight of the emitted beam, distance to reflector can be measured. Devantech SRF08 UltraSonic Ranger, is the proximity sensor employed for this modeling. According to the data sheet of this sensor, a change of processor made more timers available in compare with previous model like SRF04. Previous models of this sonar family suffer from cross-talk. However, one of the problems with terminating the ranging is that the in-flight "ping" does not know this. For instance, it simply bounces off a far wall and returns. Now if it happens to return just after a new ranging have started, the sonar will pick up this earlier "ping" and think there is an object much closer than there really is. The SRF08 allows the maximum gain to be limited to reduce this possibility. Nevertheless, this developed feature has to be evaluated as well.

**Table 3.1:** Technical specification of Davantech SRF08

| | |
|---|---|
| *voltage* | 5v |
| *current* | 15mA Typ. 3mA Standby |
| *frequency* | 40KHz |
| *max. range* | 6000mm |
| *min. range* | 30mm |
| *max analogue gain* | variable to 1025 in 32 steps |
| *connection* | standard IIC |
| *echo* | Multiple echo - keeps looking after first echo |

According to *a priori* knowledge, mounting sonars in a way that its transceivers are parallel with the ground, sonars are prone to first "detect" the ground for a

distance less than $200 - 300$mm between sensor and the ground. Proximity sensors return a value else than zero in the case there is an object in their field of view which has reflected the emitted beam from the sensor. This value is calculated by measuring the time of flight of the sound waves in air which is the longest among fluids and solids. Time of flight is calculated based on the distance that a beam (mechanical wave) with velocity of sound (ca. 343m/s at 20° C) can travel. In the context of "robotics", the term "detected" refers to that value, which means that in a distance of that value there is an object. The field of view for SRF08 is depicted in Figure 3.1, for a range of approximately 3000mm and 90° horizontal field of view. Observing the graph, it is obvious that for instance, objects located at a further distance than 1500mm and with an incident angle of $30 - 35°$, are not detectable. The settings of the range or the analogue gain for graph in Figure 3.1 is not specified in the data-sheet. Knowing that according to the specification of the sonar which is summarized in Table 3.1, behaviour of the sonar highly depends on the setting of the aforementioned factors. Nevertheless, this beam pattern provides an intuition of what should be expected of this sonar. Until this point, it is clear that the place which the experiment is going to take place in, shall be engineered in such a way that it has a certain amplitude from ground, and from the ceiling as well. Conducting a number of experiments in a room with a ca. 3000mm height, the sonar is prone to detect ceiling for a distance of ca. 1500mm. The vertical field of view also is needed to be evaluated to give a better understanding of the sensor behaviour.

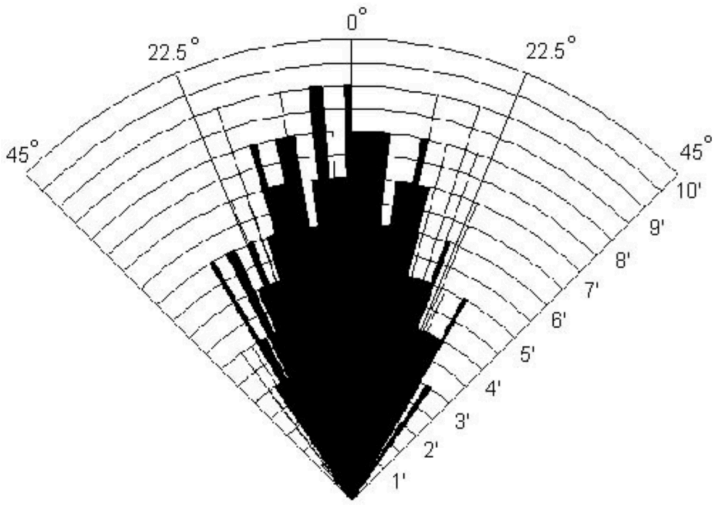



**Figure 3.1:** The beam pattern of SRF08

## 3.2 Experiment Setup

In order to perform an input-output mapping, obtaining experimental input is required. This is carried out by collecting a data set at discrete points in the input space. Two inexpensive, metal stand structures with 1500mm height were assembled with reasonable resilience. The idea for the structure of the stands are taken from [Gutierrez-Osuna et al., 1998] with some modifications. A sonar was mounted on a

stand and another stand was prepared to hold a light object (reflector) steady. The sensor mounted on the stand, meeting the center of the reflector mounted on the another stand. The detection of the vertical part of the stand is completely notable; as it is depicted in Figure 3.2, the mounted reflector is 300mm ahead of the vertical part of the structure.

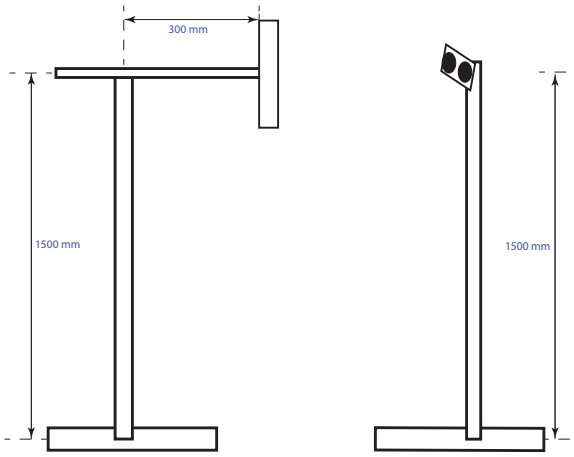



**Figure 3.2:** reflector stand and sonar stand

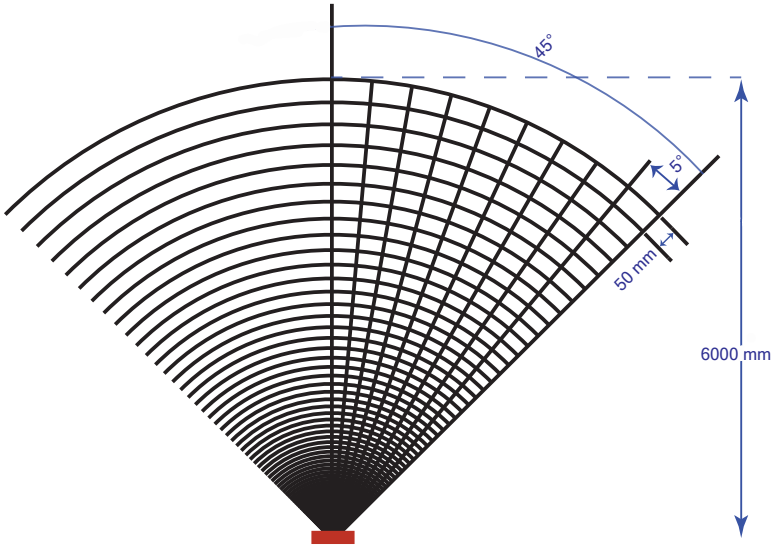



**Figure 3.3:** Grid pattern, the red rectangle resembles the sonar with a range set to 6000mm

Two different object (reflector), including a rectangular rubber (120x130x15mm) and a circular mirror of diameter 200mm and depth of 3mm were considered; representing low and high reflectance respectively. A value of 0.9 is considered for high reflectivity object (mirror), and 0.1 for low reflectivity object (rubber). These values are chosen and set in order to comply with data rescaling. Else than reflectance (R) of the

objects, the incident angle ($\theta$) to the objects and the sensor reading (SR) are the other parameters to be estimated. The SR is the value returned by the sensor (SR = d $\cos\theta$, where d is the distance between the sensor and the reflector).

A grid (see Figure 3.1), was precisely measured and marked on the floor, according to the range and beam pattern of the sonar. The sonar was set to operate in a maximum range of 6000mm and an analogue gain with register value of 0B experienced to match the specified range. The minimum and the maximum of $d$ is 30 and 6000mm respectively and the maximum of $\theta$ is 45°. The increment or step size are set to 50mm and 5° for distance and angle respectively.

In this experiment, the room temperature is kept around 19°C and the detection pattern is assumed to be symmetric, so only a half cone is considered.

### 3.2.1 Data Acquisition

The sonar is connected to a micro controller (Arduino Mega board) as a communication interface, transferring the sampled data to be stored. To acquire high resolution data from the Devantech sensor, the embedded IIC communication bus is used. IIC is a two-wire, bi-directional serial bus that provides a simple and efficient method of data exchange between devices. It is most suitable for applications requiring occasional communication over a short distance between many devices. The IIC standard is a true multi-master bus including collision detection and arbitration that prevents data corruption if two or more masters attempt to control the bus simultaneously [Herveille, 2003].

Below is the algorithm used for communication between a sonar and a micro controller.

**Algorithm 1** Sonar-MicroController Communication

**Input:** Maximum Range, Analogue Gain
**Output:** Proximity Range

```
1: procedure RANGING
2:     start Serial communication
3:     while True do
4:         Instruct sensor to read echoes
5:         Wait for sensor reading
6:         Instruct sensor to return a particular echo reading
7:         Request reading from sensor
8:         Receive reading from sensor
9:     end while
10: end procedure
```

In this setup, the analogue gain register is set to 0x0E and the maximum range is limited to 6000$mm$. Letting the sonar stand to be fixed, the reflector stand were displaced according to the grid pattern (Figure 3.3). The sampling repeated 50 times for each intersection in the grid, starting at 30mm up to 6000mm with intervals/increment of size 50mm and from 0° up to 45° with intervals of 5°. The

data were stored as a vector of estimated parameters; angle, reflectance and sensor reading. With this setting, the mathematical equation modeling the system would get the form of

$$f(\theta, R, SR) = range \tag{3.1}$$

The connections between the sonar and the micro controller were not soldered in order to resemble defective connections.

Identification of the type and the source of the errors is possible by means of statistical methods which is out of the scope of this thesis.

## 3.3 Neural Network Model

Knowing how Neural Networks can solve almost any non-linear regression problem, there are some dominant factors one should be aware of prior the initialization of a neural network.

In order to finely approximate the factors, a couple of questions need to be posed: What type of information should be learnt by the neural network? Float or binary digits? How the data should be pre-processed? Re-scaled or normalized? Does increasing the degree of freedom in input data lead to a better result? How Weights and Bias are set? What kind of Activation function should be used then? What kind of output is expected? Does a negative output value indicate any feature of the data? How the topology of the network should look like? What is the number of neurons and layers?

Answer to these questions give rise to the parameters which play crucial role in the training phase, performance of the network and hence the complexity level of the problem. For instance, initialization and the momentum are of high importance since poorly initialized networks cannot be trained with momentum and well-initialized networks perform markedly worse when the momentum is absent or poorly tuned [Sutskever et al., 2013].

Prior to NN training, data needs to be pre-processed to increase the performance of the NN. The parameter vector including the angle values $(0 - 45°)$, the reflectance level (0.1 or 0.9), the credence of sampled sensor readings$(0 - 1)$ and the sampled data (sensor readings) are obviously positive real numbers (of type float). Normalizing to zero mean and unit variance and rescaling to the range of $[0, 1]$ are the usual alternatives for pre-processing data for NNs. With the made assumptions for the range of value of reflectance and credence, and the fact that gradient descent is to be employed in the NN, rescaling the data to the range of $[0, 1]$ seems to be an appropriate approach. Data rescaling contributes to the transformation of data set, so that it has unit variance over the whole data set.

Relying exclusively on geometric information, such as the relative position and orientation of the sonar and the reflector, the reflectance level of the reflector and the credence of sensor readings, yields to a multidimensional input space. The output has a one dimensional space, which is the mapping of the input space.

The data set is labeled by the actual distance between the sonar and the reflector

as this is a problem of supervised learning.

Shuffling the data set, it has to be split nine to one, the larger part is the training set and the other is the validation set. Different configurations based on the weight/bias initialization, number of layers, number of neurons in each layer and feeding direction are being considered to find the best result. The NN models are also implemented to run on both CPU and GPU, hence their efficiency can be evaluated in terms of training time.

The next step is to initialize weights and biases both classic approaches and also according to [He et al., 2015] regarding its promising result.

Mean squared error which is the difference between the estimator and what is estimated, is the selected loss function for this problem. Various optimization methods need to be evaluated to get the best result. Performance of optimizers such as Stochastic Gradient Descent (SGD), Adadelta, Adagrad, Adam and Nadam are to be examined for feed-forward NN and for recurrent NN, RMSProp is the optimizer to be evaluated because it benefits the efficiency of mini batch learning and consequently speeds up the learning process.

Regularizers are employed in NN models to support one another, i.e. two weight regularization penalty methods such as weight decay and dropout are considered. Early stopping criterion is exploited to store the weight which has the "best" result on validation set, i.e. the one with the least error.

Bias regularization is not considered in NN models for this problem. The reason is that bias regularizers do not interact with the data through multiplicative interactions, and therefore do not have the interpretation of controlling the influence of a data dimension on the final objective.

Rescaling the data set to the range of $[0, 1]$, the appropriate activation function is Sigmoid.

Using the validation set to determine the number of hidden units, two topologies are evaluated on the data set. The evaluation started by implementing a Multi-layer perceptron (MLP) as a feed-forward NN and a robust feed-back NN architecture, called *Long-Short term Memory*, to compare their performance in this particular task. Below is a general algorithm used for the training of NN model.

**Algorithm 2** Neural Network Model

**Input:** Experimental data, Split ratio, Number of Data Features, Maximum number of Epochs, Number of Hidden Layers,
**Output:** Trained NN Model

**procedure** LOAD DATA & INITIALIZE THE MODEL
    Load Data
    Rescale Data
    Split Data to training and validation set
    Initialize the model based on data features
**end procedure**
**procedure** BUILD & TRAIN THE MODEL
    Initialize Weights & Biases
    Set Activation Function
    Set Loss Function
    Set Optimizer
    Set Dropout Ratio
    Compile the Model
    Save the Model
    **for** number of Epochs **do**
        Train the model on Training set
        Evaluate the model on Validation set
        Save weights
    **end for**
    Save the best weights
**end procedure**

Having the NN trained, re-instructing the model with the saved weights, the input-output mapping has the following algorithm.

**Algorithm 3** Input-Output Mapping

**Input:** Experimental Data, Saved Model, Saved Weights
**Output:** Mapped Data

**procedure** LOAD THE MODEL & WEIGHTS
    Load the Model
    Load the best weights
    Compile the Model
    Map Experimental Data
**end procedure**

The scripts written in Python to be run on CPU are modified to be applied to a graphic processing unit as well.

# 4
# Results

Regarding the constraints of using ultrasonic sensors, the experiment took place at ReVeRe (*Research Vehicle Resource* [1]) lab, where the desired range of the sensor could be covered.

Neural network models are implemented in Python using *Keras* library [Chollet, 2015] with Theano backend on a 2.2GHz Intel Core *i*7 processor with 16GB 1600MHz DDR3 memory and also on a GTX 750 Ti with standard memory configuration of 2048 MB.

The NN models are available at `https://github.com/speloot/MasterThesis`, for further use, following the provided algorithms for NN training and mapping will suffice.

## 4.1 Experimental Samples

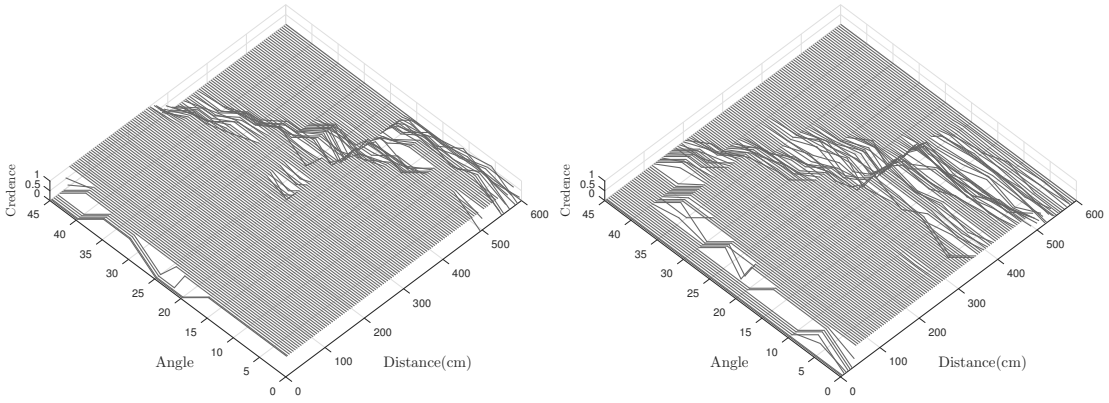
During roughly 20x8 hours, a total number of 11x2x121x50 samples, for 11 angles, 2 reflectance and 121 distances were collected and stored. An additionally angle of 22.5° is considered for a better resolution at "boundaries" as depicted in the beam pattern (Figure 3.1). These digits are the acquired samples of different angles, reflectance and distance respectively. Each 50 sampling took an average time of 2 minutes regarding the sonar and reflector position; the longer distance to the reflector, the longer elapsed time.

Sampled data was redundant as it was expected and in certain incident angle and distances, sonar was unable to detect any object and therefor output zero. In order to take this "phenomena" into account, the credence of measured sample were calculated and considered as another parameter to be estimated. Credence (Cr) is calculated as a fraction of number of correct readings and the number of samples. A range reading is labeled as correct if it falls within ±10% of the nominal distance between sonar and reflector. Consequently, the mathematical modeling of the system will get the form of following equation

$$f(\theta, R, SR, Cr) = range \tag{4.1}$$

A snippet of the prepared input data to be fed to neural network is provided in Table 4.1.

[1] https://www.chalmers.se/safer/EN/projects/pre-crash-safety/projects/revere-research-vehicle




(a) High reflectance (Mirror) (b) Low reflectance (Black rubber)

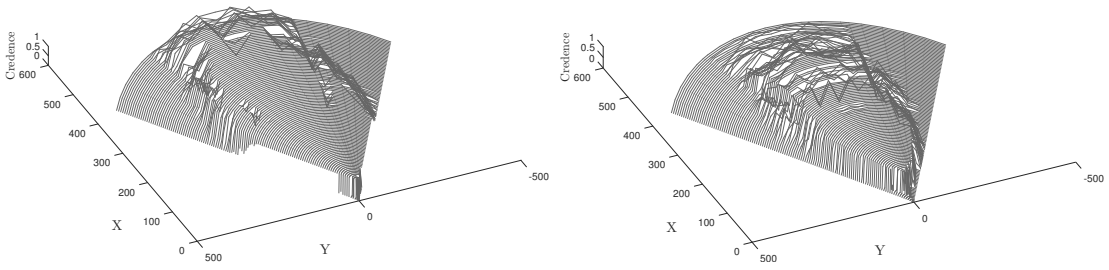
**Figure 4.1:** Experimental credence surface of the sampled data in polar coordinates

As it is depicted in Figure 4.1, the credence of the experimental data for both mirror and rubber reflector is reasonable for a distance of 300mm but for further distance, sampled data is highly redundant.

By zooming in the credence graph of the mirror, it is clear that even for close ranges (less than 500mm) the sonar output has a very low credence, and the same result complies to the rubber reflector.

In the case of rubber reflector, the credence of sampled data is getting lower for further distance as the incident angle increases.

According to the data sheet of the sonar, the detection pattern of the sonar can be assumed symmetric, hence, the credence of sampled data in Cartesian coordinate can be presented in a graph, giving an intuition of the horizontal field of view of the sonar (see Figure 4.2).




(a) High reflectance (Mirror) (b) Low reflectance (Black rubber)

**Figure 4.2:** Experimental credence surface of the sampled data in Cartesian coordinates, X axis are in "cm"

**Table 4.1:** NN input vector for a degree of 30, a reflectance of 0.9 and a distance of 3900mm

| Angle | Reflectance | Sensor Reading | Credence | Label |
|---|---|---|---|---|
| 0.6667 | 0.9 | 0.6583 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6583 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6583 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6583 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6616 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6616 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6616 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6616 | 0.88 | 0.6482 |
| 0.6667 | 0.9 | 0.6600 | 0.88 | 0.6482 |

In order to study the effect of cross-talk, the sampling was done first by mounting two sonars with the least distance from one another and also with 50mm distance both vertically and horizontally. The first one was set to "fire" in a regular frequency while the other one was set to fire more frequent. Observing the acquired data, no trace of cross-talk between the sonars were detected.

## 4.2 Neural Network Models

After feeding the sampled data to a MLP, different optimizers are employed to train the network. In this comparison, a single hidden layer considered with different number of neurons.

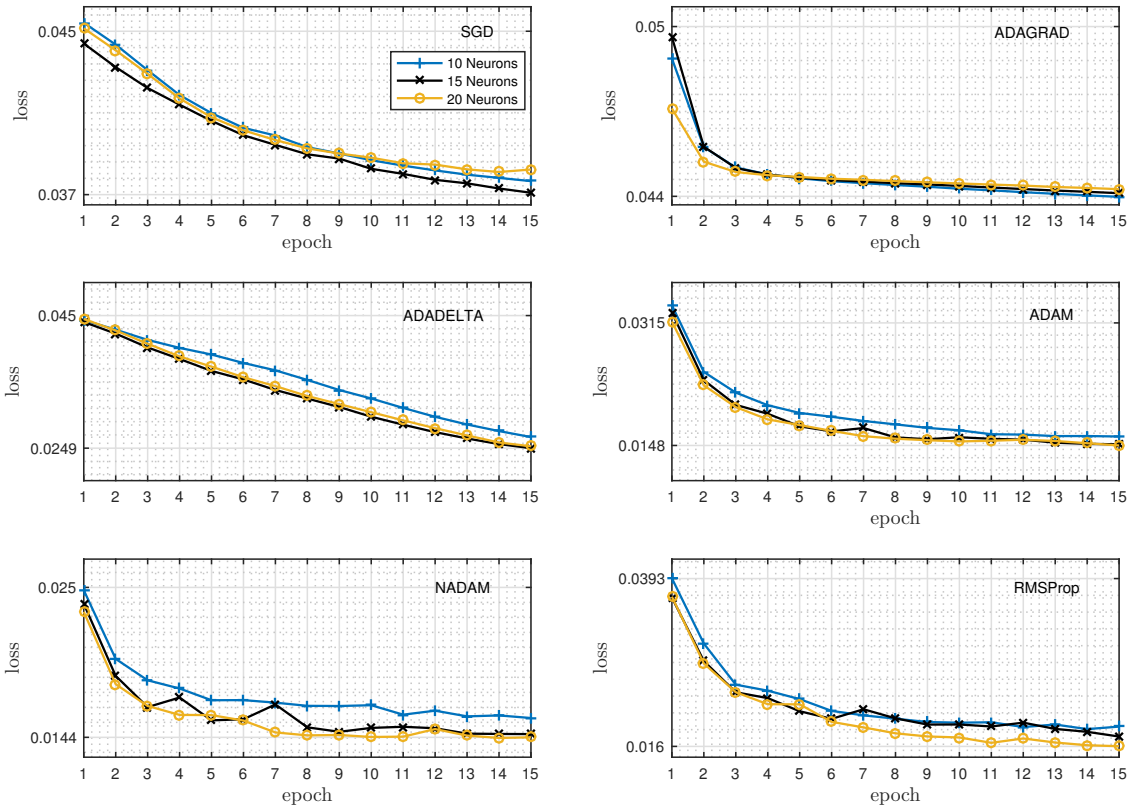



**Figure 4.3:** Validation loss graph for different optimizers, the minimum and maximum values for each optimizer are indicated.

As the number of neurons increases, the number of parameters of the NN, such as weights and biases also increase. Increment in the number of computations leads to a slightly longer elapsed time of training the NN. Nevertheless, with greater number of neurons, training the NN yields lower validation loss.

For the NN using Nadam optimizer, as it is shown in Figure 4.3, the validation loss in the first epoch is considerably lower than NNs with other optimizers. It means that the optimizer succeeded to escape local minima and consequently quicken the convergence with a lower loss value than the other optimizers after 15 epochs.

The final value of training and validation loss after 15 epochs and for different number of neurons and optimizers are gathered in Tables 4.2, 4.3 and 4.4. The elapsed time indicates that for these shallow NNs (NNs with low number of hidden layers), the optimization process takes longer time for Nadam since there are more parameters to be calculated.

**Table 4.2:** Loss function result for 10 neurons after 15 epochs

| *Optimizer* | train. loss | valid. loss | ET(min) |
|---|---|---|---|
| *SGD* | 0.0390 | 0.0376 | 8.06 |
| *ADAGRAD* | 0.0456 | 0.0439 | 7.93 |
| *ADADELTA* | 0.0305 | 0.0265 | 9.18 |
| *ADAM* | 0.0206 | 0.0160 | 8.47 |
| *RMSProp* | 0.0220 | 0.0187 | 8.08 |
| *NADAM* | 0.0198 | **0.0157** | 9.20 |

**Table 4.3:** Loss function result for 15 neurons after 15 epochs

| *Optimizer* | train. loss | valid. loss | ET(min) |
|---|---|---|---|
| *SGD* | 0.0385 | 0.0370 | 7.92 |
| *ADAGRAD* | 0.0454 | 0.0441 | 7.97 |
| *ADADELTA* | 0.0285 | 0.0249 | 8.98 |
| *ADAM* | 0.0179 | 0.0149 | 8.64 |
| *RMSProp* | 0.0221 | 0.0173 | 8.23 |
| *NADAM* | 0.0174 | **0.0146** | 9.33 |

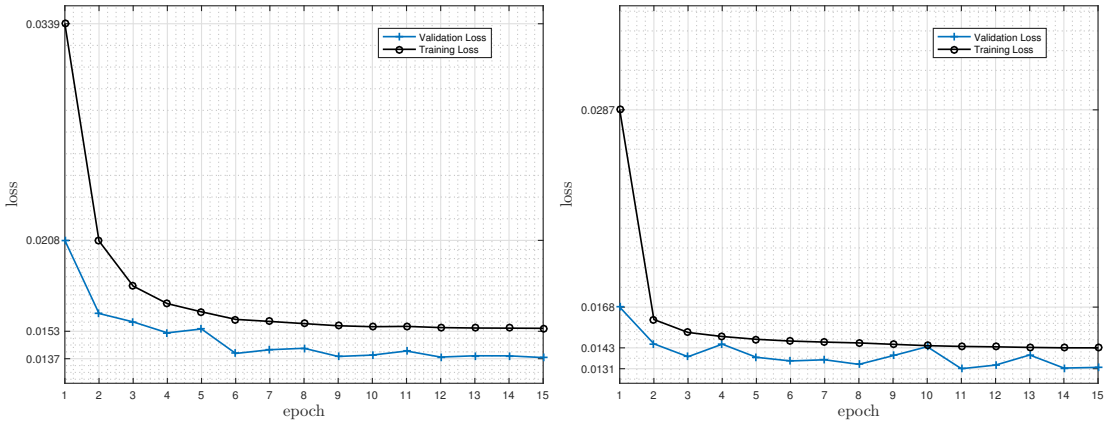
**Table 4.4:** Loss function result for 20 neurons after 15 epochs

| *Optimizer* | train. loss | valid. loss | ET(min) |
|---|---|---|---|
| *SGD* | 0.0393 | 0.0382 | 8.35 |
| *ADAGRAD* | 0.0458 | 0.0442 | 8.29 |
| *ADADELTA* | 0.0283 | 0.0252 | 9.20 |
| *ADAM* | 0.0177 | 0.0147 | 8.52 |
| *RMSProp* | 0.0197 | 0.0160 | 7.96 |
| *NADAM* | 0.0169 | **0.0144** | 9.11 |

Since each pattern of the training set is a selected pattern out of randomly shuffled data set, i.e. there should not be any connection between a pattern at time $t$ and another one at $t + 1$, therefore it is decided to update the weights online.

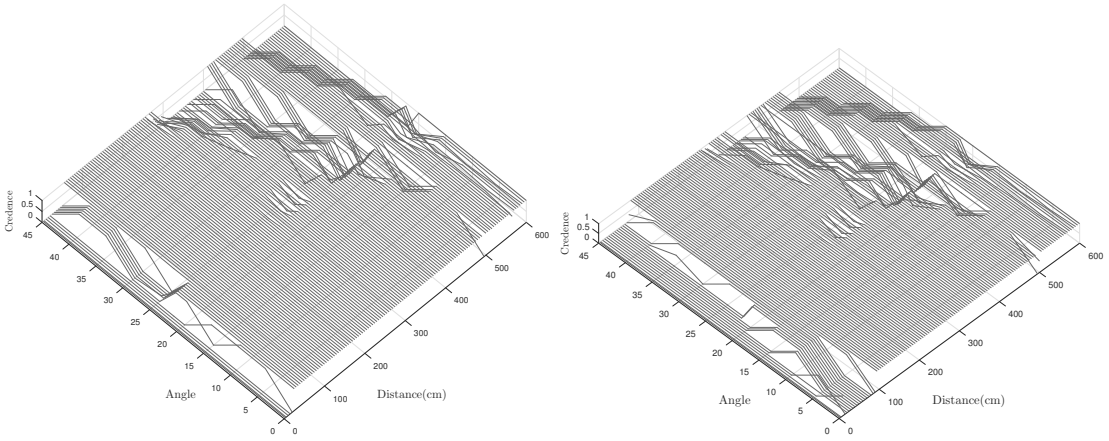
The loss function results given in tables above show that, the Nadam optimizer has a better performance in comparison with other ones. Hence, with selecting Nadam as the optimizer, other number of neurons in the hidden layer were examined/evaluated to decrease the validation loss as much as possible.

In Figure 4.4, training and validation loss of the MLP and the LSTM trained NN for 25 hidden neurons are depicted.




(a) MLP(ET=9.42min) (b) LSTM(ET=42.39min)

**Figure 4.4:** Training and validation loss graph for 25 neurons, using Nadam optimizer



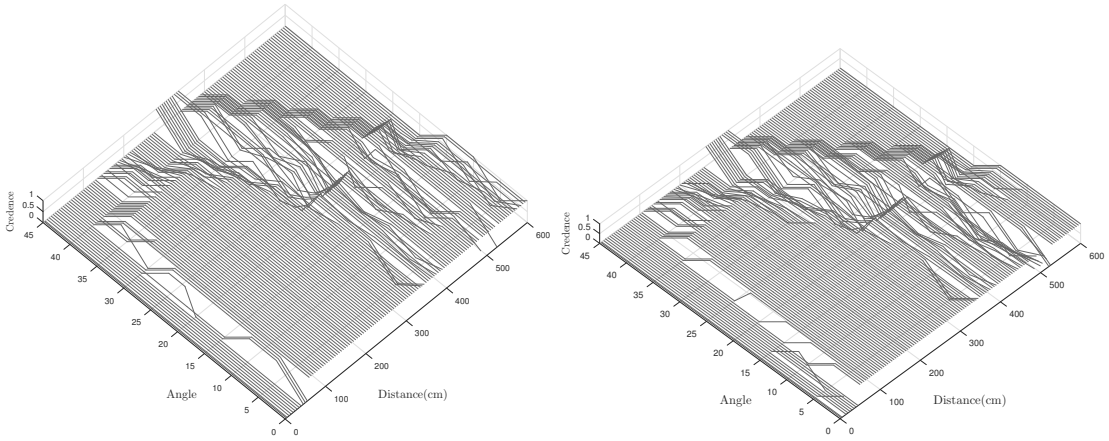

(a) Mapped samples by LSTM (Mirror) (b) Mapped samples by MLP (Mirror)

**Figure 4.5:** Credence surface of the mapped sampled data in polar coordinates for the mirror reflector

The LSTM model outperforms MLP with a less validation loss, but training of LSTM model surprisingly takes almost 5 times longer than MLP model.

As it is expected, both NN models yield a better result for the mirror reflector since its sampled data is less redundant.

Calculating the credence of the mapped data, it turns out that LSTM model has better performance in mapping mirror sampled data for shorter ranges as it is shown in Figure 4.5 and Figure 4.6.




(a) Mapped samples by LSTM (Black Rubber) (b) Mapped samples by MLP (Black Rubber)

**Figure 4.6:** Credence surface of the mapped sampled data in polar coordinates for the black rubber reflector

Different number of hidden units and learning techniques parameter were also evaluated but none of them outperformed the specified configuration above.

Training the NNs in mini batches did not yield a better result than online training.

It is needed to mention that training the specified neural networks on both CPU and GPU took almost the same time; matrix multiplication, convolution, and large element-wise operations can be accelerated by employing a GPU.

The effect of cross-talk term was highly eliminated in this model of sonar as it is described previously but a side effect can be the huge uncertainty in sonar output for close ranges.

# 5

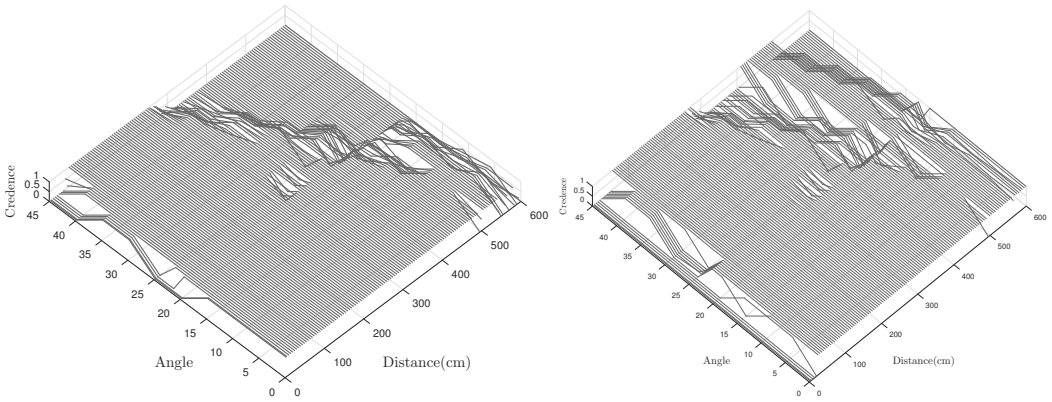
# Discussion

The result of applying neural networks in system identification highly depends on the information/data which is to be learned by the network. In real-world problems, such as the one proceeded in this thesis, there might be many affecting parameters that either we are not aware of or it is not feasible to model.

With the 4 major parameters taken into account for this model-based system identification, a neural network with 25 hidden neurons approximate the relation between mentioned parameters and the sensor output.

As a justification for the performance of the neural networks employed in this work, one can point out the data which the NN was trained with. For a certain value of incident angle and reflectance and distance, the sensor tends to output values with high variance (see Table 4.1). This variety in labeling the same pattern (e.g. zero labeled patterns) is what "confuses" NN and causes difficulties in learning process no matter how efficient NN is provided with advanced learning techniques. Mini-batch normalization or mini-batch training has no application in this problem, since relating a number of patterns which has the same label but different attributes is not reasonable.

By comparing the sampled data and the mapped data, it is clear that NN to high extent succeeded to learn the patterns in the redundant sampled data. As it is depicted in Figure 5.1, the NN has identified (learnt) and slightly eliminated the noise for further distance than 5000mm and higher incident angle, while the same applies for the noisy samples at $22^\circ$ between $3000 - 4000$ distance, where the credence of mapped data is highly increased.




(a) Sampled Data (Mirror) (b) Mapped data (Mirrror)

**Figure 5.1:** Comparison between experimental credence surface of the sampled data and the mapped data in polar coordinates

By monitoring the loss of training set and the respective loss of validation set as the one in the appendix, one can promptly deduce if a certain configuration of hidden neurons and learning techniques will yield a better result than the one before in very first iterations. Taking this approach can save time in the process of determining a proper configuration of number of hidden units where one can see where the NN is being over-fitted or under-fitted.

Identifying the vertical field of view demands a more advanced designation of experiments; with a simple triangular calculation, to catch the effective vertical field of view, one needs to cover a field with 6000mm length, approximately 4600mm width and most likely more than 1500mm height (height of the used stand). Fulfilling this purpose demands a more complicated structure than the one used in this work. A more complicated experiment would target the parsimony of the model, hence evaluating the vertical field of view is left for future work.

Taking a smaller interval than what is considered in this modeling, for the incident angle and the distance between the sonar and the reflector, will result in less perturbation in sampled data. Although by providing more training data (patterns), training of NNs will be improved, but it consequently will take much more time, which is a trade-off that should be taken into account.

# 6

# Conclusion

In this thesis I evaluated noise in an ultrasonic sensor by modeling the one which is employed in experimental miniature cars at Chalmers/University of Gothenburg. This noise modeling is to be integrated into OpenDaVINCI to increase the level of reality in simulation, and hence, contribute the test process of autonomous driving.

Acquiring sampled data by conducting the experiment, neural network is taken as the approach for this sensor modeling. The horizontal field of view of the sonar was covered according to the boundaries provided in the sonar data sheet. To evaluate neural networks proficiency, multilayer perceptron and long-short term memory are put into practice.

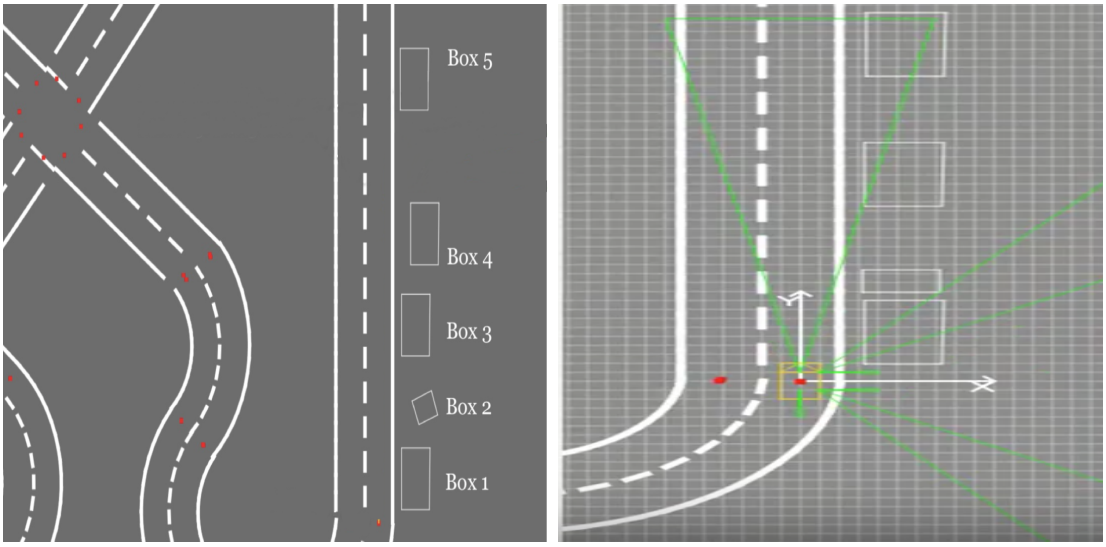
As the result proves, fewer parameter calculation in MLP in comparison with LSTM makes it a better choice when the time is a determining term, but one should take into account the fact that LSTMs perform reasonably better and yield better result than MLP even in input-output mapping, where LSTM is known for its result achievement in speech/handwriting recognition.

Regardless the robustness of neural network or generally machine learning techniques in input-output mapping, the non-trivial and time consuming experiential part, plays the most important role in modeling. It is the field measurements which build up the patterns that are to be "taught" to a network of neurons; a network whose performance level is just as well as the quantity and quality of its input.

## 6.1 Future Work

A miniature vehicle (or a robot) does not necessarily have information about the substance, texture or the angle of objects around, nevertheless, in a pre-defined environment, one could take advantage of the result of this thesis.

The results could be utilized in a virtual test environment, particularly in OpenDaVINCI, where the model of the environment can be pre-defined, for instance, when the goal is to handle a scenario like side-way parking as it is depicted in Figure 6.1.




(a) Parking scenario (b) Simulation environment in OpenDaVINCI

**Figure 6.1:** The self-driving miniature vehicle follows the lane and once it has found a sufficiently wide parking spot moves into it avoiding any collision by the boxes.

Placing the self-driving miniature vehicle at the starting point, the vehicle starts following the lane and meanwhile detecting the "obstacles" on the right side and measuring the distance to them.

By designing the model of the environment and obstacles (boxes in this case) in a way that the reflectivity level of the obstacles (the texture), the incident angle between the sonar and the obstacles and the distance to the obstacles are pre-defined and known, the sonar readings and its credence could be replaced by the Noise Model implemented in the virtual test environment.

On the other hand, studying proximity sensors and based on my own experiences in utilizing sonars (in a humanoid robot project), to cover a particular area at the vicinity of a miniature vehicle, beside other proximity sensors, usually more than a sonar is being used. Knowing the credence of the sensor output in certain circumstances, and by considering the position (distance and incident angle) of each sensor to a wall or to an object, sensor fusion techniques can make use of this credence. Sensor fusion techniques such as Bayesian, Extended Kalman Filter or Fuzzy Logic are tools that could exploit the credence of fused sensors to improve the accuracy of vehicle perception of surroundings.

The efficiency of the shallow neural network (the MLP) and its small size could contribute to improve the motion of vehicle in simulation environment where having the NN trained, the mapping of a new pattern can be attained in milliseconds.

A very promising approach which is not investigated in this work is the implementation of a hybrid of aforementioned NN models, where one can train a NN with MLP first, and then use LSTM to reduce the residual errors and tune up the model. There are a few difficulties accomplishing the hybrid model, such as initialization

of backward weights in the recurrent part of the hybrid model. These investigation might open the doors towards a better approximation of proximity sensors.

# Bibliography


[Anders Krogh, 1992] Anders Krogh, J. A. H. (1992). Simple weight decay can improve generalization in advances in neural information processing systems.

[Barabasi, 1999] Barabasi, Albert, R. (1999). Emergence of scaling in random networks. *science*, 286(5439):509–512.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Chollet, 2015] Chollet, F. (2015). Keras.

[D. Kingma, 2014] D. Kingma, J. B. (2014). Adam: A method for stochastic optimization.

[David E. Rumelhart, 1986] David E. Rumelhart, Geoffrey E. Hinton, R. J. W. (1986). Learning representations by back-propagating errors nature, vol. 323. *Nature. 323 (6088)*, pages 533—-536.

[Dozat, 2014] Dozat, T. (2014). Incorporating nesterov momentum into adam.

[F. A. Gers and Cummins, 2000] F. A. Gers, J. S. and Cummins, F. (2000). Learning to forget: Continual prediction with lstm.

[Fang and Dobson, 2013] Fang, L. and Dobson, S. (2013). In-network sensor data modelling methods for fault detection. In *International Joint Conference on Ambient Intelligence*, pages 176–189. Springer.

[G. Hinton and Swersky, 2012] G. Hinton, N. S. and Swersky, K. (2012). Overview of mini-batch gradient descent. 15.

[Gamarra-Diezma et al., 2015] Gamarra-Diezma, J. L., Miranda-Fuentes, A., Llorens, J., Cuenca, A., Blanco-Roldán, G. L., and Rodríguez-Lizana, A. (2015). Testing accuracy of long-range ultrasonic sensors for olive tree canopy measurements. *Sensors*, 15(2):2902–2919.

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256.

[Gutierrez-Osuna et al., 1998] Gutierrez-Osuna, R., Janet, J. A., and Luo, R. C. (1998). Modeling of ultrasonic range sensors for localization of autonomous mobile robots. *IEEE Transactions on Industrial Electronics*, 45(4):654–662.

[Hammer, 2000] Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, 31(1):107–123.

[Haykin, 1994] Haykin, S. (1994). Neural networks, a comprehensive foundation.

[Haykin and Network, 2004] Haykin, S. and Network, N. (2004). *A comprehensive foundation*, volume 2.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.

[Herveille, 2003] Herveille, R. (2003). I2c-master core specification. technical report. Open-Cores.org.

[Horvath, 2003] Horvath, G. (2003). Neural networks in system identification. *Nato Science Series Sub Series III Computer And Systems Sciences*, 185:43–78.

[Isermann, 2011] Isermann, R. (2011). *Fault-diagnosis applications: model-based condition monitoring: actuators, drives, machinery, plants, sensors, and fault-tolerant systems*. Springer Science & Business Media.

[J. Duchi, 2011] J. Duchi, E. Hazan, Y. S. (2011). Adaptive subgradient methods for online learning and stochastic optimization.

[Jang et al., 2008] Jang, H., Park, A., and Jung, K. (2008). Neural network implementation using cuda and openmp. In *Digital Image Computing: Techniques and Applications (DICTA), 2008*, pages 155–161. IEEE.

[Maass, 1997] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.

[Monner and Reggia, 2012] Monner, D. and Reggia, J. A. (2012). A generalized lstm-like training algorithm for second-order recurrent neural networks. *Neural Networks*, 25:70–83.

[Nesterov, 1983] Nesterov, Y. (1983). A method of solving a convex program- ming problem with convergence rate o(1/sqr(k)). page 372–376.

[Ni et al., 2009] Ni, K., Ramanathan, N., Chehade, M. N. H., Balzano, L., Nair, S., Zahedi, S., Kohler, E., Pottie, G., Hansen, M., and Srivastava, M. (2009). Sensor network data fault types. *ACM Transactions on Sensor Networks (TOSN)*, 5(3):25.

[Schmidhuber, 1997] Schmidhuber, S. H. J. (1997). Long short-term memory.

[Sharma et al., 2010] Sharma, A. B., Golubchik, L., and Govindan, R. (2010). Sensor faults: Detection methods and prevalence in real-world datasets. *ACM Transactions on Sensor Networks (TOSN)*, 6(3):23.

[Shen et al., 2013] Shen, J., Su, P.-C., Cheung, S.-c. S., and Zhao, J. (2013). Virtual mirror rendering with stationary rgb-d cameras and stored 3-d background. *IEEE Transactions on Image Processing*, 22(9):3433–3448.

[Smith, 1993] Smith, M. (1993). *Neural Networks for Statistical Modelling*. International Thomson Computer.

[Srivastava et al., 2014] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

[Sun et al., 2013] Sun, Q., Yu, W., Kochurov, N., Hao, Q., and Hu, F. (2013). A multi-agent-based intelligent sensor and actuator network design for smart house and home automation. *Journal of Sensor and Actuator Networks*, 2(3):557–588.

[Sutskever et al., 2013] Sutskever, I., Martens, J., Dahl, G. E., and Hinton, G. E. (2013). On the importance of initialization and momentum in deep learning. *ICML (3)*, 28:1139–1147.

[Tibshirani, 1996] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5 - rmsprop. COURSERA: Neural Networks for Machine Learning.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[Zou and Hastie, 2005] Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.

# A

# Appendix 1

Implementing the Algorithm 2 in Python with Keras libraries, it is possible to directly monitor the result of validation loss calculation. Modifying parameters such as number of hidden neurons, learning rate, dropout ratio and optimizing method, will lead to different result. By monitoring the validation loss value, possible improvement in performance of the NN can be assessed. For instance, in the case that after third or forth epoch, validation loss value is not improved, one can terminate training of the NN.

Below is an example of a MLP model with 30 neurons using Nadam optimizer. After 20 epochs, validation loss reaches its minimum value (see Figure A.1) which is slightly higher than the result of LSTM with 25 neurons after 15 epochs and same optimizer, although the elapsed time (ca. 13min) is much less than the elapsed time of LSTM model.

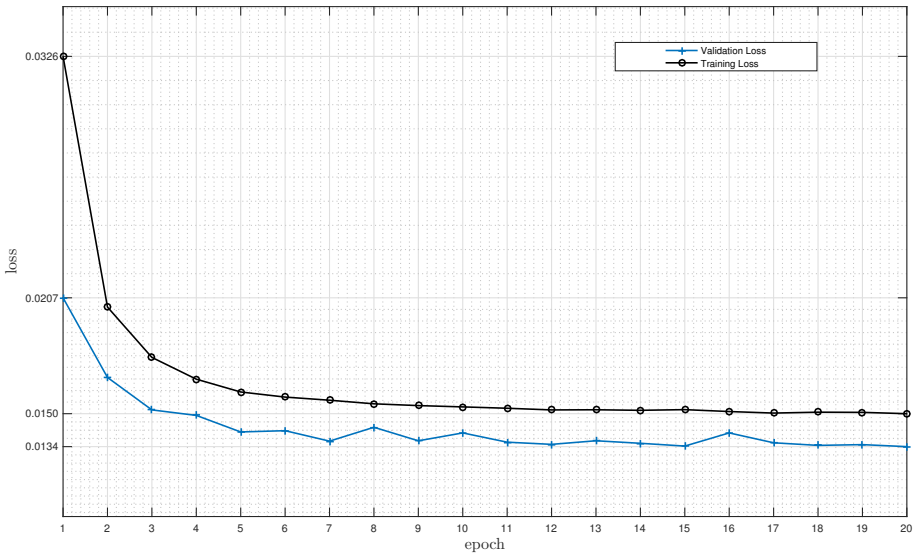



**Figure A.1:** Training and validation loss for MLP model with 30 neurons after 20 epochs

**Listing A.1:** Tyraining result of MLP model on CPU

```
Using Theano backend.
Loading DATA...
123100  training sequences
10000  validation sequences
Number of Neurons = 30
Batch Size = 1
20  Iterations
Building the model...
Training the model...
Train on 123100 samples, validate on 10000 samples
Epoch 1/20
123070/123100 [============================>.] - ETA: 0s - train_loss: 0.0326
 val_loss improved from inf to 0.02072, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0326 - val_loss: 0.0207

Epoch 2/20
123094/123100 [============================>.] - ETA: 0s - train_loss: 0.0203
 val_loss improved from 0.02072 to 0.01682, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0203 - val_loss: 0.0168

Epoch 3/20
123069/123100 [============================>.] - ETA: 0s - train_loss: 0.0178
 val_loss improved from 0.01682 to 0.01523, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0178 - val_loss: 0.0152

Epoch 4/20
123082/123100 [============================>.] - ETA: 0s - train_loss: 0.0167
 val_loss improved from 0.01523 to 0.01496, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0167 - val_loss: 0.0150

Epoch 5/20
123096/123100 [============================>.] - ETA: 0s - train_loss: 0.0161
 val_loss improved from 0.01496 to 0.01415, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0161 - val_loss: 0.0141

Epoch 6/20
123088/123100 [============================>.] - ETA: 0s - train_loss: 0.0159
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0159 - val_loss: 0.0142

Epoch 7/20
123073/123100 [============================>.] - ETA: 0s - train_loss: 0.0157
 val_loss improved from 0.01415 to 0.01370, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0157 - val_loss: 0.0137

Epoch 8/20
123089/123100 [============================>.] - ETA: 0s - train_loss: 0.0155
```

```
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0155 - val_loss: 0.0144

Epoch 9/20
123074/123100 [============================>.] - ETA: 0s - train_loss: 0.0154
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0154 - val_loss: 0.0137

Epoch 10/20
123095/123100 [============================>.] - ETA: 0s - train_loss: 0.0154
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0154 - val_loss: 0.0141

Epoch 11/20
123069/123100 [============================>.] - ETA: 0s - train_loss: 0.0153
 val_loss improved from 0.01370 to 0.01365, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0153 - val_loss: 0.0136

Epoch 12/20
123085/123100 [============================>.] - ETA: 0s - train_loss: 0.0152
 val_loss improved from 0.01365 to 0.01354, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0152 - val_loss: 0.0135

Epoch 13/20
123085/123100 [============================>.] - ETA: 0s - train_loss: 0.0152
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0152 - val_loss: 0.0137

Epoch 14/20
123078/123100 [============================>.] - ETA: 0s - train_loss: 0.0152
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0152 - val_loss: 0.0136

Epoch 15/20
123083/123100 [============================>.] - ETA: 0s - train_loss: 0.0152
 val_loss improved from 0.01354 to 0.01345, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0152 - val_loss: 0.0135

Epoch 16/20
123083/123100 [============================>.] - ETA: 0s - train_loss: 0.0151
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0151  - val_loss: 0.0141

Epoch 17/20
123078/123100 [============================>.] - ETA: 0s - train_loss: 0.0151
 val_loss did not improve
123100/123100 [==============================] - 39s
```

```
- train_loss: 0.0151  - val_loss: 0.0136

Epoch 18/20
123086/123100 [============================>.] - ETA: 0s - train_loss: 0.0151
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0151  - val_loss: 0.0135

Epoch 19/20
123084/123100 [============================>.] - ETA: 0s - train_loss: 0.0151
 val_loss did not improve
123100/123100 [==============================] - 39s
- train_loss: 0.0151 - val_loss: 0.0135

Epoch 20/20
123090/123100 [============================>.] - ETA: 0s - train_loss: 0.0150
 val_loss improved from 0.01345 to 0.01343, saving weights
123100/123100 [==============================] - 39s
- train_loss: 0.0150 - val_loss: 0.0134
 9892/10000 [============================>.] - ETA: 0

score:[0.013427657528076325, 0.014800000000000001]

Elapsed time: 13.31573736667633 min

Model saved.
Best weights saved.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 30)                150
_________________________________________________________________
dropout_1 (Dropout)          (None, 30)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 31
=================================================================
Total params: 181
_________________________________________________________________
```