CHALMERS
UNIVERSITY OF TECHNOLOGY

Neural Networks for modelling of a virtual sensor in an engine

A comparison of LSTM and CNN structures

Master's thesis in Systems, Control and Mechatronics

JONAS ALEXANDERSSON, ELIAS SONNSJÖ LÖNEGREN

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:EENX30

Neural Networks for modelling of a virtual sensor in an engine

A comparison of LSTM and CNN structures

JONAS ALEXANDERSSON, ELIAS SONNSJÖ LÖNEGREN

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2019

Neural Networks for modelling of a virtual sensor in an engine
A comparison of LSTM and CNN structures
JONAS ALEXANDERSSON, ELIAS SONNSJÖ LÖNEGREN

Supervisor: Mauro Bellone, Senior Researcher, Department of Mechanics and Maritime Sciences, Chalmers University of Technology
Examiner: Yiannis Karayiannidis, Assistant Professor, Department of Electrical Engineering, Chalmers University of Technology

Master's Thesis 2019:EENX30
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Neural Networks for modelling of a virtual sensor in a dynamical system
A comparison of LSTM and CNN structures
JONAS ALEXANDERSSON, ELIAS SONNSJÖ LÖNEGREN
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Virtual models are commonly used in the automotive industry to increase the efficiency in the development process. Such models are often highly complex and new methods for developing them are constantly being evaluated. A promising approach is to use data-driven machine learning models for virtual testing. Machine learning makes use of neural networks in order to represent a function which maps an input to an output.

In this thesis, two different types of neural networks, Convolutional Neural Network (CNN) and Long-Short Term Memory (LSTM), were evaluated and compared for some of the more complex sensors of an engine. We show that deep learning is a viable option for generating data-driven models with high accuracy, sometimes exceeding the physical modelling of complex signals. Furthermore, it is shown that data-augmentation may be used to increase the robustness and the models' ability to generalise. Finally, we show that transfer learning can be used to retrain a model, making it able to perform with high accuracy on a new case.

Both the CNN and LSTM models show promising results, with the latter providing slightly higher accuracy and benefit more from data-augmentation. The CNN, however, appears to be more suitable for transfer learning.

Keywords: LSTM, CNN, deep learning, machine learning, artificial neural networks, virtual modelling, sensor modelling.

Acknowledgements

This thesis has been carried out for 20 weeks at Chalmers University of Technology and in collaboration with Volvo Penta. We would like to thank our examiner Yiannis Karayiannidis for his endless support and guidance throughout the writing of this thesis. The meetings have provided us with knowledge and a lot of interesting discussions.

A big thanks to our supervisor at Chalmers, Mauro Bellone, for always being there, ready to brainstorm ideas and provide tips and tricks. It has been a pleasure discussing everything between neural networks and football with you!

We give our warmest thanks to our supervisor at Volvo Penta, Ethan Faghani, who provided us with the necessary contacts and data at Volvo and guidance along the way. A special thanks to all of Volvo Penta, for the use of your facilities and computers. Many thanks go out to the VIRTEC team at Volvo Penta, for answering our questions regarding data-sets and engine-behaviour.

We would like to thank our families, specific others and friends for their support and understanding during this spring. We would also like to thank our thesis colleagues: Emma Berglund, Jesper Andersson and Reema Pinto for all great conversations during lunch and the fun we have had together at Volvo Penta.

Jonas Alexandersson, Elias Sonnsjö Lönegren, Gothenburg, May 2019

Contents

List of Figures **xi**

List of Tables **xvii**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem description . . . 2
1.3 Limitations . . . 2
1.4 Related Work . . . 3
1.5 Thesis structure . . . 4

2 Background on Neural Networks **5**
2.1 Neural Networks . . . 5
2.1.1 Objective of a neural network . . . 5
2.1.2 Data-set and splits . . . 6
2.1.3 Components of the network . . . 7
2.1.4 Model Initialisation . . . 9
2.1.5 Propagation and learning . . . 10
2.1.6 Multi-Layer Perceptron . . . 11
2.1.7 Feature Scaling . . . 13
2.1.8 Regularisation . . . 13
2.1.9 Batch Normalisation . . . 14
2.1.10 Learning Rate . . . 15
2.2 Recurrent Neural Network . . . 15
2.3 Long Short-Term Memory . . . 17
2.4 Convolutional Neural Network . . . 18
2.4.1 The Convolution Operator . . . 20
2.4.1.1 Convolutional Filters and activation maps . . . 20
2.5 Hyperparameter Iteration Techniques . . . 22
2.5.1 Grid Search . . . 22
2.5.2 Random Search . . . 23

3 Data Processing **25**
3.1 The Physical Model . . . 25
3.2 Data-set and test cycles . . . 26
3.3 Target signals . . . 27
3.4 Selection of input-signals . . . 28

3.5 Feature Scaling and processing . . . 30
3.6 Reshaping the data . . . 33
3.7 Data-set split . . . 35

4 Network structures **39**
4.1 Loss Functions and Regression Metrics . . . 39
4.2 Grid Search . . . 41
4.2.1 Grid Search: CNN . . . 41
4.2.2 Grid Search: LSTM . . . 44
4.3 Random Search . . . 48
4.4 Selection of Hyperparameters and functions . . . 49
4.4.1 Convolutional hyperparameters . . . 49
4.4.2 General Parameters and functions . . . 49
4.5 Final Network Structures . . . 50

5 Robustness **53**
5.1 Introducing noise . . . 53
5.2 Grid search with noise . . . 54

6 Transfer Learning **57**
6.1 Grid search with transfer learning . . . 57

7 Results **61**
7.1 Baseline model performance . . . 61
7.2 Robustness . . . 68
7.3 Transfer Learning . . . 70

8 Discussion **73**

9 Conclusion **77**

A Appendix 1 **I**
A.1 Input signals on the *NRTC-ww* . . . I
A.2 Histogram of the input signals . . . VI

List of Figures

2.1 The relationship between the inputs, x_i, the weigths, w_i and the bias, b of a single neuron in a hidden layer. 8
2.2 The neuron is a linear combination, z_1, of inputs, weights and the bias. Once calculated, the neuron is activated by a function, $g(.)$, such that the output of the neuron in a hidden layer is $a = g(z)$. . . . 8
2.3 The forward propagation, $a = g(z(x_i, w_i))$, according to Eq. (2.6) and backward propagation, $\frac{dL}{da}$, according to Eq. (2.17). The propagation may be seen with respect to a single neuron. The loss is followed through the operations and finally calculated for each input, with respect to each weight and bias. 11
2.4 An example of a multi-layer perceptron with one input layer, two hidden layers, and one output layer. 12
2.5 A Recurrent Neural Network unfolded over time in order to visualise the time dependencies in the network and how to work with them. . . 16
2.6 An LSTM Cell with the recurrent output from the previous time step and the output of the previous layer as input. Also seen is the input gate, forget gate and the output gate. The picture is copied from the Deep Learning textbook by Ian Goodfellow, Yoshua Bengio and Aaron Courville [1]. 18
2.7 How convolutional networks make use of shared weights. Here only 3 weight are in use, when for a fully connected layer it would require 7 times the weights. 19
2.8 Convolution between an image and filter, providing an output. 21
2.9 The "sliding" of a filter over the input to the convolutional layer. Here with a stride set to 1. 21
2.10 The concept of grid search considering two parameters, one on each axis. 22
2.11 The concept of random search considering two parameters, one on each axis. 23

3.1 The structure of the data, as provided from the test-cycles. Each cycle consists of 80 signals and is sampled with $10Hz$ 26
3.2 The structure of the data, when sorting out the 13 most significant signals. Each cycle consists of 13 signals and is sampled at $10Hz$ . . 29
3.3 The behaviour of the *Exhaust Temperature* signal throughout the test-cycle *NRTC-ww*. 29

3.4 The behaviour of the *Rail Pressure* signal throughout the test-cycle *NRTC-ww*. 30
3.5 The distribution of two signals with different units and magnitude throughout the test-cycle *NRTC-ww*. 31
3.6 The behaviour of two signals with different units and magnitude throughout the test-cycle *NRTC-ww*. 32
3.7 A signal processed by using the interpolation described in Eq. (3.1). The orange dotted line is the synthetic values calculated. 33
3.8 The reshaping of the test-cycles creates an input-array of size $(n \times 13)$ with a target value of (1×1). 34
3.9 The reshaped input is saved along with the target value as a training example. 34
3.10 The reshaped input sequence and target data is saved as a training example and stacked in an array. 35
3.11 The region where the training sequence of the base-set and the NRTC 4 data operates. 36
3.12 The region where the NRTC 9 (training data) and the NRTC 10 (test data) operates. 37

4.1 The value observed by the network, $\hat{y}$, and the true sequence, y, against time. On the closed set between a and b, the sum of the true sequence is highlighted in dark-grey while the sum of the error (the sum of the difference between $\hat{y}$ and y) is highlighted in bright-grey. . 40
4.2 How the RPE is affected by the number of (depth) of convolutional layers for three models with varying number of filters in each layer. The average (trend) displays how the RPE is decreased as the depth of the neural network increases. 42
4.3 How the RPE is affected by the number of filters in each convolutional layer, for three models with different depth in regards to the convolutional layers. The average between the models (the trend) may be seen plotted in red. 43
4.4 How the RPE is affected by the kernel size of the filters in each model. Seen is Convolutional models with 1-, 2- and 3-layer depth. Also plotted is the average (trend), in red. 44
4.5 How the RPE is affected by the number of neurons in each fully connected layer, for three models with varying amount of fully connected layers. The average between the models (the trend) may be seen plotted in red. 45
4.6 How the RPE is affected by the number of cells in each LSTM layer, for four models with varying amount of LSTM layers. The average between the models (the trend) may be seen plotted in purple. 46
4.7 How the RPE is affected by the amount of Batch Normalised layers, for three models with varying learning rate. The average between the models (the trend) may be seen plotted in red. 47

4.8 89 models, generates by randomly selected values for the hyperparameters Kernel Size, Number of Convolutional Filters and number of fully connected layers, evaluated against the test-set. 48
4.9 A Figure over the structure of the baseline model for LSTM. 51
4.10 A Figure over the structure of the baseline model for CNN. 51

5.1 A grid search performed where different setup of α and β was iterated over and evaluated against RPE. 55

6.1 Strategies of using transfer learning by freezing different amount of layers. In the left staple all layers are trainable, while the middle stable only has some of the base layers (Convolutional or LSTM) trainable and to the right is an example of when only the output layers and the fully connected layers are trainable, keeping the base layers untouched. 58
6.2 The performance of the CNN model's configurations on the test cycle (*NRTC-10*), presented in RPE on the y-axis against the number of trainable layers on the x-axis. 59
6.3 The performance of the LSTM model's configurations on the test cycle (*NRTC-10*), presented in RPE on the y-axis against the number of trainable layers on the x-axis. 59

7.1 A prediction made by the CNN baseline model for Conc_NOx against the true sequence of NRTC cycle 4. 62
7.2 A prediction made by the LSTM baseline model for Conc_NOx against the true sequence of NRTC cycle 4. 63
7.3 A prediction made by the CNN baseline model for Conc_NOx against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the difficulties of capturing the highest spikes. . . 63
7.4 A prediction made by the LSTM baseline model for Conc_NOx against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the difficulties of capturing the highest spikes. . . 64
7.5 A prediction made by the CNN baseline model for Flw_FuelDiesel against the true sequence of NRTC cycle 4. 65
7.6 A prediction made by the LSTM baseline model for Flw_FuelDiesel against the true sequence of NRTC cycle 4. 65
7.7 A prediction made by the CNN baseline model for Conc_Soot against the true sequence of NRTC cycle 4. 66
7.8 A prediction made by the LSTM baseline model for Conc_Soot against the true sequence of NRTC cycle 4. 67
7.9 A prediction made by the CNN baseline model for Conc_Soot against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the overall difficulties of capturing the dynamical behaviour of Soot. 67

7.10 A prediction made by the LSTM baseline model for Conc_Soot against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the overall difficulties of capturing the dynamical behaviour of Soot. 68

A.1 The behaviour of the normalised *Exhaust Temperature* signal throughout the test-cycle *NRTC-ww*. I
A.2 The behaviour of the normalised *em_FuelValue* signal throughout the test-cycle *NRTC-ww*. I
A.3 The behaviour of the normalised *Main Injection* signal throughout the test-cycle *NRTC-ww*. II
A.4 The behaviour of the normalised *Post Injection* signal throughout the test-cycle *NRTC-ww*. II
A.5 The behaviour of the normalised *Pre Injection* signal throughout the test-cycle *NRTC-ww*. II
A.6 The behaviour of the normalised *Pre Injection Angle* signal throughout the test-cycle *NRTC-ww*. III
A.7 The behaviour of the normalised *EGR Position* signal throughout the test-cycle *NRTC-ww*. III
A.8 The behaviour of the normalised *Rail Pressure* signal throughout the test-cycle *NRTC-ww*. III
A.9 The behaviour of the normalised *Inlet Position* signal throughout the test-cycle *NRTC-ww*. IV
A.10 The behaviour of the normalised *Throttle Position* signal throughout the test-cycle *NRTC-ww*. IV
A.11 The behaviour of the normalised *Engine Speed* signal throughout the test-cycle *NRTC-ww*. IV
A.12 The behaviour of the normalised *Injection Angle* signal throughout the test-cycle *NRTC-ww*. V
A.13 The behaviour of the normalised *Wastegate Position* signal throughout the test-cycle *NRTC-ww*. V
A.14 The distribution of the normalised *Exhaust Temperature* signal over all input data. VI
A.15 The distribution of the normalised *em_FuelValue* signal over all input data. VI
A.16 The distribution of the normalised *Main Injection* signal over all input data. VII
A.17 The distribution of the normalised *Post Injection* signal over all input data. VII
A.18 The distribution of the normalised *Pre Injection* signal over all input data. VIII
A.19 The distribution of the normalised *EGR Position* signal over all input data. VIII
A.20 The distribution of the normalised *Rail Pressure* signal over all input data. IX

A.21 The distribution of the normalised *Inlet Position* signal over all input data. IX
A.22 The distribution of the normalised *Throttle Position* signal over all input data. X
A.23 The distribution of the normalised *Engine Speed* signal over all input data. X
A.24 The distribution of the normalised *Injection Angle* signal over all input data. XI
A.25 The distribution of the normalised *Wastegate Position* signal over all input data. XI

List of Tables

3.1 Seen is the unit, maximum and mean-values of all 13 signals. 30
3.2 The table shows which scaling method that was chosen for each of the 13 signals. 32

4.1 The three models used for the first grid search. The difference between the models are the number of filters, *receptive fields*, applied in the convolutional layers. 42
4.2 The three models used for the second grid search. The difference between the models is the depth of the convolutional layers. 42
4.3 The three models used for the second grid search. The difference between the models is the depth of the convolutional layers. 43
4.4 The three models used for the grid search over the fully connected neurons. The difference between the models is the depth of the fully connected layers. 45
4.5 The four models used for the grid search over LSTM Cells. The difference between the models is the depth of the LSTM layers. . . . 46
4.6 The three models used for the grid search over the amount of batch normalised layers. The difference between the models are the different learning rates. 47
4.7 Three models trained with and without batch normalisation layers between all other layers. With Batch Normalisation, the models converged faster and provided a lower RPE. 50

7.1 The performance of the two model types (CNN and LSTM) on the measured quantity Concentration of NO_x. Presented are their respective performance on the three metrics presented in Section 4.1. 61
7.2 The performance of the two model types (CNN and LSTM) on the measured quantity Flow Fuel Diesel. Presented are their respective performance on the three metrics presented in Section 4.1. 64
7.3 The performance of the two model types (CNN and LSTM) on the measured quantity Concentration of Soot. Presented are their respective performance on the three metrics presented in Section 4.1. 66
7.4 The performance of the two model types (CNN and LSTM) on the three measured quantities Flow Fuel Diesel, Concentration of NO_x and Concentration of Soot. Presented are their respective performance on the three metrics presented in Section 4.1. 68
7.5 A Table with the final setups of α and β for the robustness part. . . . 69

7.6 A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal NO_x. 69
7.7 A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal Flow Fuel Diesel. 69
7.8 A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal Soot. 70
7.9 The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated for NO_x. Shown is the three metrics described in Section 4.1 on the new Test-set *NRTC 10*-cycle. 70
7.10 The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated on the three metrics described in Section 4.1 on the old Test-set *NRTC 4*-cycle. 71
7.11 The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated on RPE as described in Section 4.1 on the new test-set *NRTC 10*-cycle. 71

1
Introduction

The application of machine learning methods to predict dynamical systems is increasing sharply. By their design, machine learning models offer tremendous flexibility, capable of capturing complex behaviour and patterns, while maintaining a relatively tiny computational footprint. The automotive field is one of the areas in which machine learning is applied with promising results. Occasionally, real sensors may be too expensive or too noisy in real conditions to be used to provide measurements. Instead, machine learning models may be used to predict parameters such as torque, efficiency or emissions. For a given constant power-train system, the models may then provide and maintain a good estimate of the output of the sensors.

1.1 Motivation

Volvo Penta is today a company which produces and delivers top class power-train systems. However, in order to continue to deliver high-quality products an important step is to constantly look into and implement new technologies and solutions. Machine learning as a modelling tool is such a technology. Compared to other modelling methods, models developed through machine learning do not rely on a process being described through equations. Instead, machine learning models learn the connection from the input to the output data and employ a function in order to describe the relationship, this is further explained in Chapter 2. Since engines and powertrains are built upon very complex subsystems and processes, machine learning could prove to be a very efficient tool when it comes to encoding of virtual models of more complex processes. From Volvo Penta's perspective, such a tool is useful not only in the complete product but also during the development process. A virtual model allows for testing even when the physical component is unavailable. The engine could then eventually be tested without being hooked up to a physical test cell: it could instead be tested virtually. Furthermore, small changes to the engine could be implemented and the virtual testing may help to evaluate the effect beforehand. Both of these scenarios would reduce both cost and time for the manufacturer during the development stage. Additionally, virtual models could be of value in the physical engine itself since it could be used to replace an expensive sensor or provide additional information for the engine. This could, in turn, be used to decide how to optimise the engine's power output, exhaust output or fuel consumption. Virtual models could also be used as complementary sensors. Both in order to spot faults within the engine or with the physical sensor itself and in order to get a better sensor estimate in areas where signals with a high-level of noise are

involved.

A crucial part of machine learning is to have access to a large amount of high-quality data. Additionally, the data needs to be labelled in order to perform supervised learning. At Volvo Penta, these criteria are fulfilled, as a big labelled data pool exists. A high quantity of the data has been gathered through running test cycles in the test cells and some through field testing. The gathered data-sets consists of a variety of pre-defined test cycles which aims to test the engine in the majority of its available running conditions.

1.2 Problem description

With regard to Volvo Penta's point of interest, this project will focus on investigating the possibilities of introducing machine learning models as a substitute for physically modelled sensors. The project will focus on three different types of sensors; fuel consumption-, NO_x- and Soot-sensors. The sensors are all measuring physical dynamical parameters. Thus, two types of network structures will be constructed and compared for this task: a Convolutional Neural Network (CNN) and a Long-Short Term Memory (LSTM) network for each of the sensors.

The possibility of developing models with improved robustness will also be investigated. This will cover both the aspects of robustness against signal noise but also robustness in terms of creating models that can give a good sensor estimate for different engine calibrations. Higher robustness in the models could increase the trustworthiness of them and result in a need for fewer models. Additionally, the capabilities of transferring knowledge from a pre-trained model to a new one will be investigated. This could decrease the training time needed for each model and thus allow for new ones to be developed at a lower cost.

At the end of the project, a comparison between the LSTM and CNN structures with respect to all the previous tasks will be performed. This will show the advantages and disadvantages of the different network types with regards to the previously mentioned areas.

1.3 Limitations

The data provided by Volvo Penta contains a large number of engine parameters, obtained from the different tests described in Section 3.2. In the test cell, all parameters are measured and may be directly observed. However, as it is desired to also use the Neural Networks outside of the test environment the models for this project will be developed solely on the signals and parameters given by the Engine Control Module (ECM). These signals are available in an engine outside the test cell and will be available for other engines than the one considered in this project as well.

Due to the limited amount of time, the goal of this project is not to develop neural networks with the highest accuracy, but rather to find structures with sufficient accuracy. An evaluation of the two different types of structures, CNN and LSTM, is the goal of the project.

Additionally, all tuning of the models will be done for the NO_x-sensor, due to the limited amount of time. The structures and setups which are found during this process will be used for the evaluation of the Soot- and fuel consumption-sensor as well.

1.4 Related Work

In the automotive industry, Hardware-in-the-loop, *HiL*, systems are being used in order to verify software and hardware at an early stage in the development process. A HiL-system is a virtual model of a system where some of the subsystems are made up of real physical components. Hence, virtually developed subsystems are mixed with real components to create a complete model. The two main ways for creating the virtual models are through *Theoretical modelling* or *Experimental modelling*. Theoretical modelling focuses on building models through equations and known relationships between signals. However, when dealing with more complex subsystems the relationships can be problematic to describe through equations in a closed form. The second way of modelling, Experimental modelling, is performed through viewing the subsystem as a black-box, considering mainly the input and output of the system. The model then adapts the behaviour of the process by minimising an error measurement between the model and the true process [2]. The models developed during this thesis will fall under the second category as machine learning focus on adapting a function that connects the input and the outputs of the system.

There exists a variety of articles and studies where dynamical systems, or parts of it, are being modelled using machine learning and neural networks. A case closely related to the given project is *Heavy Duty Diesel Engine Modeling with Layered Artificial Neural Network Structures* by Sediako, A.D., Andric, J., Sjöblom, J., and Faghani, E. [3]. The article describes an approach for modelling engine sensors through artificial neural networks. 7 signals from the engine management system (EMS) are used as input for the models in order to predict a total of 30 output signals. The signals were coupled such that some of the more complex output signals used outputs from the other models as additional inputs. This increases the amount of knowledge those specific models get through the inputs. The models were based either on the multilayer perceptron (a trivial example in neural networks), Section 2.1.6, or a recursive structure, Section 2.2. The article shows promising results in using these network types for modelling sensor outputs in the engine. Improvements in terms of prediction accuracy were seen in several signals compared to previous modelling techniques. The exhaust emissions (NO_x and *Soot*) were the most difficult signals to model due to their complex behaviour. However, an improvement was seen in these areas, when making use of the time-dependencies.

A Neural Network type that is commonly used for time-dependent sequences is the Recurrent Neural Network (RNN) [4] since the previous state(s) are remembered. However, as the RNN grows indefinitely when remembering the states, a solution to this is the Long Short-Term Memory network (LSTM), which introduces gates to reset (or forget) the state [5]. For this reason, the LSTM network may be used more efficiently for modelling longer time-dependencies and is used for speech recognition and forecasting [6] (a similar case to observing a state) among others, and therefore

a suitable candidate for capturing the dependencies of an engine.

Convolutional Neural Networks (CNN) are often used to classify objects in images [1], with the beneficial use of the convolutional operator: allowing complex problems to be expressed with fewer parameters. As the LSTM-network may be used to model sequences, the CNN can be used to do the same [7] as well modelling of time series such as in [8]. A dynamical system, with multiple correlated signals over time, can be structured as an image with only one channel and therefore the CNN can be applied to model such system over time, as it has been done in [9].

The engine changes with time as components become worn out and the dynamics differ from its initial behaviour. The issue of this engine (making the neural network models developed for the original dynamics outdated) could be viewed as the problem of training a network for a certain objective and later on fine-tune it for another. This is an issue with multiple solutions such as model initialisation according to Greedy layer-wise pretraining, but more commonly used (especially for convolutional neural networks) is transfer learning. The concept of transfer learning is that low-level features found by a CNN are generic descriptors [10], where the low-level descriptors can be used for different recognition tasks. In [11], it is found that the first layer features are not specific and limited to a certain task or data-set: but rather applicable to multiple data-sets and tasks. These findings could be used to fine-tune a pre-trained neural network on a data-set from another engine or with a different engine calibration. The low-level features such as correlations between signals over time should be able to be transferred to another case, to preserve knowledge.

1.5 Thesis structure

The report is structured in the following manner: in Chapter 2, the necessary theoretical background is presented and summarised. The data-sets, collection, and processing of these are presented in Chapter 3. In Chapter 4, the methodology of creating neural network structures is presented. A methodology for augmenting data and potentially creating more generalised models is presented in Chapter 5. Finally, Chapter 6 presents how transfer learning is applied to re-train models on new data, making them more flexible to new problems. The models are tested and evaluated on unseen data in Chapter 7. Furthermore, the quality in-between the different models is compared and presented. In Chapter 8, the methodology and results are discussed while Chapter 9 presents the conclusions drawn from the results, potential continuations and future work.

2

Background on Neural Networks

Humans are always trying to push technological development further and a big topic in this area is "thinking machines", artificial intelligence. However, in order to create machines that can think for themselves, we need to teach the machines more than just objects and items, we need to teach them how to process what they see and what they measure. If this can be done, the machines can gather the information themselves and make qualified decisions based on the information collected. One approach to this problem is the concept of Machine Learning.

Machine learning is the process of allowing a computer to "learn from examples", i.e. having an algorithm generating an approximated function based on data. By being fed a massive amount of examples to train on, the computer will eventually learn to recognise patterns without explicit instructions on what to focus on. The result from a machine learning algorithm is a mathematical model, built on the data available. These patterns are learned by extracting small features and stacking them, learning more complicated patterns and features by the combination of the smaller ones. When layers of these features are combined and hierarchies exist within the model, the type of machine learning is often called *deep learning*. The most common deep learning models are based upon the technology known as neural networks [1].

2.1 Neural Networks

Multilayer perception is perhaps the most essential example of the deep learning models, certainly of this report. This is also known as a feedforward neural network or Deep feedforward networks. The feedforward network is called as such since information always flows forward in the network, from input to output.

2.1.1 Objective of a neural network

The goal of generating such network is to parameterise and approximate a function, $\hat{f}$, given a labelled data-set. A data-set of inputs, $\mathbf{x}$, and outputs, $\mathbf{y}$, are connected as follows,

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

meaning that the goal of the neural network is to approximate the function,

$$\hat{f}(\mathbf{x}, \theta) \sim f(\mathbf{x}) \tag{2.2}$$

leading to the best possible estimation of f, such that the error, e, is as small as possible:

$$e = \hat{f}(\mathbf{x}, \theta) - f(\mathbf{x}) \tag{2.3}$$

The approximated function, $\hat{f}(\mathbf{x}, \theta)$, acts as a mapping from the input to the output where the goal of the training is to find, "learn", the parameters of the approximated function, θ, yielding the smallest error. The error can be calculated in different ways and exactly how the function is stated differs from case to case. However, the objective of a machine learning algorithm is to produce a function as close as possible to the true one. Thus, whichever representation selected, there is a function which is to be minimised called the *objective*, *loss* or *cost* function. This function, $J(\theta)$, is some variant of Eq. (2.3) and the minimisation of it can be stated as:

$$\theta^* = \arg\min_{\theta} J(\theta) \tag{2.4}$$

with θ being the the set of parameters the function is to learn. There are different types of problems for the neural network to solve, but they all share the idea that the task is to learn from examples, not to learn from exact instructions. However, the type of problem to solve influences the design of both the network as well as the method of learning. The most common tasks are:

- **Classification**: The task of classification is to categorise an item into two or more categories. The approximated function, $\hat{f}$ should in this case take the input $\mathbf{x}$ and produce an output $\mathbf{y}$ such that $\hat{f}(\mathbf{x})$=$\mathbf{y}$: $(1, ..., N)$. A binary classification example problem is given an image of a Dog (input $\mathbf{x}$) to classify whether or not the image is in fact a Dog ($\mathbf{y} = 1$) or not ($\mathbf{y} = 0$).
- **Regression**: Regression is the task of predicting or observing a number, given a set of inputs. The function that is to be approximated is a mapping from N inputs to one output. An example of the regression task is the example of predicting the cost of a house based on one or a number of parameters, such as square-meters and value. The output of such function is a continuous value rather than a discrete number.

The Machine Learning algorithm is evaluated on the performance during training. These measurements are different depending on which type of task is carried out. For instance, for the *classification* task it could make sense to measure the accuracy of the classifier as a metric: the amount of correctly classified examples over the total amount. For a *regression* task, on the other hand, a more suitable metric or measurement could be the mean squared error:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.5}$$

where $\hat{y}$ is the estimated output, y the true output and n the number of samples. In order to be able to approximate a function that produces a prediction, $\hat{y}$, the inner components of a neural network needs to be defined.

2.1.2 Data-set and splits

To train a machine learning algorithm, such as a neural network, a data-set is required. The data-set is of the same type as the one it is supposed to perform on.

For instance, if a classifier such as described in Section 2.1.1 is supposed to learn to properly classify dogs, the data-set needs to be images with positive (images with dogs) and negative (images without dogs) examples. However, all data may not be used in order to teach the model how to identify a dog. That part of the data is called the **training** set. This portion of the data is used to tune the parameters of the network (more on this in the next Section), and is the vast majority of the data-set.

The rest of the data is divided into two parts: **validation** and **test** data-sets. Both sets are held from the training algorithm to make it unbiased from them. The validation set is used to provide an unbiased evaluation of the fit of the model as the training is ongoing. This is also used to tune the hyperparameters of the network. The test data-set, however, is only used when a final model has been obtained. If the validation set is used to give a hint of how the model performs, the test set is how well the model actually performs when training is completed.

Usually, the data-set is split into the three sets with the following proportions:

- **Train:** 70%
- **Validation:** 15%
- **Test:** 15%

2.1.3 Components of the network

A network is said to have a certain depth, represented by the number of layers in between the input and output layers. These layers are called *hidden layers*. Consider the function that is to be approximated, $\hat{f}(\mathbf{x}, \theta)$, the input and the desired output is known, however any calculations in between them are unknown, therefore *hidden*. The purpose of the hidden layers is that each layer can be seen as a different function, propagating the input in a certain way. The network may then approximate the most complex of functions [12].

The most important feature of a neural network is the artificial neuron. Each layer consists of a number of neurons, units that act in parallel. The units are called neurons from the idea that they receive inputs from all neurons in the previous layer and are activated, or fired, in a way that is mimicking the Human Brain. The first neuron in layer l could be described as the weighted sum of its n-inputs, $\mathbf{x}$, as:

$$z_1^{[l]} = \mathbf{w}^{[l]}\mathbf{x}^{[l]} + b_1^{[l]} \tag{2.6}$$

where b is a bias term, added to every neuron, enabling a shift of the function. The weighting of the input to the layer, $\mathbf{w}$, is a vector stacking the weights between neurons $(1 : n)$ in previous the layer and the neuron of interest in the current layer. Thus, the linear summations of a neuron in a hidden layer may be visualised as in Figure 2.1.

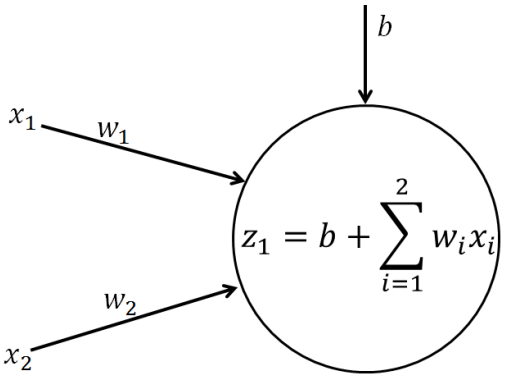


Figure 2.1: The relationship between the inputs, x_i, the weigths, w_i and the bias, b of a single neuron in a hidden layer.

With Eq. (2.6) a linear relationship is obtained between the previous and following layer. In order to extend and enable the network to capture nonlinearities in the system, the neuron is then "activated" by a nonlinear function. The activation, a, of neuron 1 in layer l is described as:

$$a_1^{[l]} = g(z_1^{[l]}) \tag{2.7}$$

where $g(.)$ is the activation function and z defined by Eq. (2.6). There are multiple different activation functions, used for different purposes in different network structures. The activation of a neuron is the last step before any information is released further on in the network, this may be visualised as in Figure 2.2.

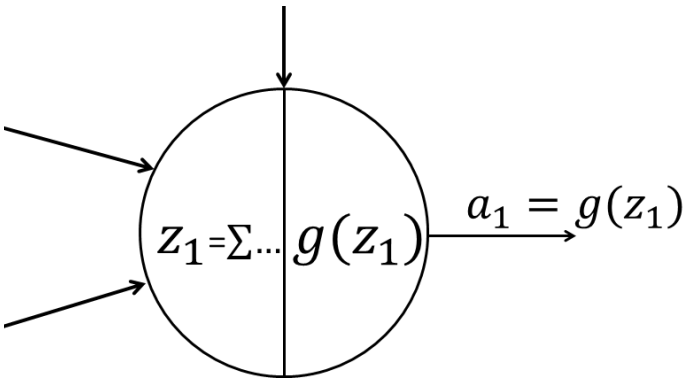


Figure 2.2: The neuron is a linear combination, z_1, of inputs, weights and the bias. Once calculated, the neuron is activated by a function, $g(.)$, such that the output of the neuron in a hidden layer is $a = g(z)$.

The most common, and usually by-default recommended activation function is the *Rectified Linear Unit* (ReLU) [1, 13]. The ReLU-function is stated accordingly:

$$ReLU(z) = g(z) = max(0, z) \tag{2.8}$$

a nonlinearity saturating values below 0. This is a trivial example of how a neuron is "fired" or activated: when the output is below 0, it is considered to be inactive,

while above it is active. There are multiple variations of the ReLU-function, one of them being *elu* which usually produces more accurate results and converges the cost-function to zero faster [14]. *elu* is defined as:

$$elu(z) = \begin{cases} z, & z \geq 0 \\ \alpha(e^z - 1), & z < 0 \end{cases} \tag{2.9}$$

where z is defined by Eq. (2.6), e is the exponential function and the constant $\alpha \in (0, 1)$ (usually set to 1) which turn *elu* to *ReLU* by $\alpha = 0$. A neuron's output is described by Eq. (2.7) and stacked together forms the input to the next layer as:

$$x^{[l+1]} = \begin{bmatrix} a_1^{[l]} \\ a_2^{[l]} \\ \vdots \\ a_n^{[l]} \end{bmatrix} \tag{2.10}$$

The weights, $\mathbf{w}$, and biases, b, of the network are learned by training. Together they form the learnable parameters, θ. Learning the parameters is the goal of training the network, but where does the training begin?

2.1.4 Model Initialisation

A Neural Network, or model, consists of multiple layers with weights and biases, as explained in Section 2.1.3. These weights and biases, θ, are what the network learns from the algorithm. However, for the neural network to propagate through the examples and update the parameters (more on this in Section 2.1.5), it requires some sort of initial point, an initialisation of the weights and biases. According to [1], modern initialisation techniques are simple: using more sophisticated ones is hard since the optimisation of the network is not fully comprehended. The point of initiation affects the rate and possibility of the convergence of the network, and different initialisation strategies may be more beneficial to different types of problems. There are however heuristic choices of initialisation techniques which aim to initialise the parameters θ sampling from a normalised distribution. One way, which may be seen in Eq. (2.11), sample the weights, $\mathbf{w}$, from a uniform distribution based on the number of inputs, m, to the layer,

$$w_{i,j} \sim U(-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}) \tag{2.11}$$

Another way, more commonly used, is the one proposed in [15]. Here the weights are initialised by, *normalised initialisation*, which may be seen in Eq. (2.12).

$$w_{i,j} \sim U(-\sqrt{\frac{1}{m+n}}, \sqrt{\frac{1}{m+n}}) \tag{2.12}$$

where m is the number of inputs to the layer and n is the number of outputs. This technique assumes, among other things, that the network acts as a chain of matrix-multiplications. However, it disregards the possibility of nonlinearities which, as defined in Eq. (2.7), clearly does not hold. This strategy is by many the default option for initialisation and provides a satisfying result [1].

2.1.5 Propagation and learning

Eq. (2.6) - (2.7) describe how a single neuron is provided with information, the outputs of previous layers, and passes it forward in the network. This is called *forward propagation*; propagating the initial input $\mathbf{x}$ through all hidden layers to produce the final output, or estimate, $\hat{y}$. This is what the function approximated by the network is currently producing given a certain input. However, with labelled data the true output, y, is known from a given input. The loss, or cost, may then be calculated according to the selected cost function, $J(\theta)$. This is the objective function sought to minimise, and given that the weights and biases of the network may be gathered in a vector, θ, the following is to be found:

$$\theta^* = \arg\min_{\theta} J(\theta) \tag{2.13}$$

The optimal parameter configuration for a network might be a hard, if not an impossible, task to solve in a straight forward manner. The parameter vector, θ, may, and often does, contain millions of parameters and is often solved using *gradient descent*. The minimisation of the cost function will look as follows:

$$\theta^* = \arg\min_{\theta} J(\theta) = \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} L(f(x_i; \theta), y_i) \tag{2.14}$$

with L being the loss between a hypothesis or estimated function and the true value, for one training example, and m the number of examples considered for an update. Thus the cost represents an average over the *batch* of training examples, denoted m. With the representation of the cost function as is, *gradient descent* is often used as an iterative optimising method by "taking steps" in the direction of the negative gradient of the cost function. The gradient is thus computed

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(f(x_i; \theta), y_i) \tag{2.15}$$

and the update rule for the parameter vector may be formulated as follows:

$$\theta \leftarrow \theta - \alpha \nabla_\theta J(\theta) \tag{2.16}$$

where α is the step-size of the learning algorithm, called *the learning rate*. The actual computation of the gradient is known as *backpropagation* [16] within the field of machine learning. The idea is to calculate the impact of a weight, w_i, on the loss, L. This is done by propagating the loss "backwards" in the network, as follows:

$$\frac{\partial L}{\partial w_i} = \sum_j \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_i} = \nabla_{w_i} L \tag{2.17}$$

where a_j are the neurons in between the loss function and the weight, w_i. By calculating these gradients, it is assumed that all weights are being updated alone. When using the gradient descent algorithm, the gradient is calculated as in Eq. (2.15) and the update is done according to Eq. (2.16) for each weight. In Figure 2.3 the *backpropagation* is seen with the gradients calculated as according to the Chain Rule of Differentiation.

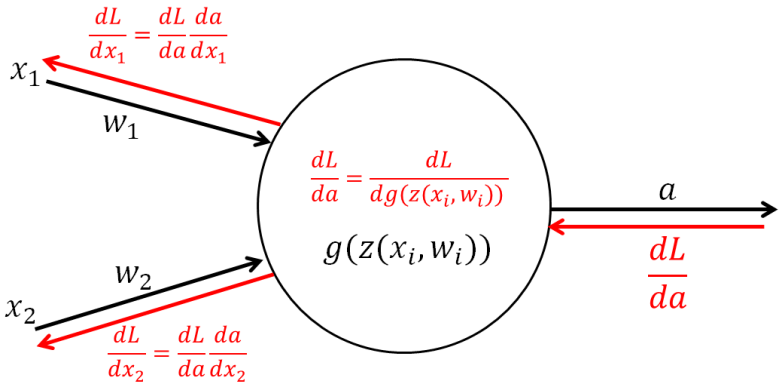


Figure 2.3: The forward propagation, $a = g(z(x_i, w_i))$, according to Eq. (2.6) and backward propagation, $\frac{dL}{da}$, according to Eq. (2.17). The propagation may be seen with respect to a single neuron. The loss is followed through the operations and finally calculated for each input, with respect to each weight and bias.

However, as both the parameter vector and the training set often are very large, the computations become highly computationally expensive. Due to this *mini-batches* are often used instead of entire data-sets for computations, using Eq. (2.15) with m being instead the size of the mini-batch, iterating through the entire set. Different optimisers may be used for different purposes in regards to different problems, but the idea is similar to performing the update according to Eq. (2.16). An optimiser closely related to the gradient descent algorithm is *Stochastic Gradient Descent* (SGD), where instead of calculating the gradient of the batch, the gradient is calculated by the loss after every example [1]. It results in a less smooth path and close to the optima it is worse at minimising the error. It, however, converges faster than gradient descent.

Since SGD and gradient descent first became famous a lot of new optimisers have been developed. The ones that are used the most are all based on SGD but with additional features added to them in order to handle specific flaws with regular SGD. The ones that are mentioned in this project is RMSprop, Adam [17] and Nadam [18]. The complexity for each of these optimisers increases with RMSprop being the least and Nadam the most complex.

2.1.6 Multi-Layer Perceptron

By connecting the different components that have been described previously, a basic Multi-Layer Feed-forward network can be composed, as explained in [19]. A Multi-Layer Feed-forward network is a network type that consists of an input, output and at least one hidden layer in between. In Figure 2.4 such a network can be seen. In this case, the network consists of an input layer, an output layer, and two hidden layers. The input layer consists of two neurons while the output layer consists of only one neuron. The hidden layers consist of three neurons in the first hidden layer and two neurons in the second hidden layer. All the layers are connected through the weights, $\mathbf{w}^1$, $\mathbf{w}^2$ and $\mathbf{w}^3$. It can be seen that each neuron is connected to all neurons in the next layer, hence all the layers are *fully connected*. Above each layer the bias vector for that layer is denoted, $\mathbf{b}^1$, $\mathbf{b}^2$ and $\mathbf{b}^3$.

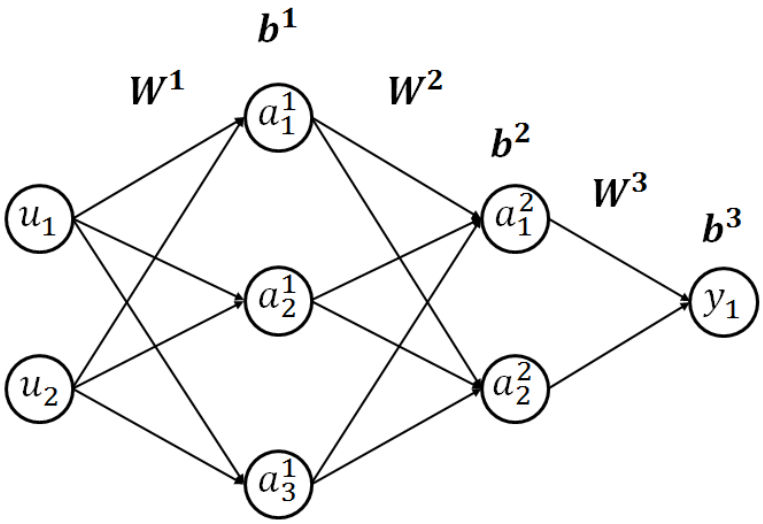


Figure 2.4: An example of a multi-layer perceptron with one input layer, two hidden layers, and one output layer.

In the report each individual weight will be denoted as $w^{[l]}_{n,m}$, where l denotes which layer the weight connects to, n denotes which neuron in the previous layer that the weight is connected to and m denotes which neuron the weight is connected to. For the network given in Figure 2.4 this would result in the following weight matrices:

$$\mathbf{w}^1 = \begin{bmatrix} w^{[1]}_{1,1} & w^{[1]}_{2,1} \\ w^{[1]}_{1,2} & w^{[1]}_{2,2} \\ w^{[1]}_{1,3} & w^{[1]}_{2,3} \end{bmatrix}, \mathbf{w}^2 = \begin{bmatrix} w^{[2]}_{1,1} & w^{[2]}_{2,1} & w^{[2]}_{3,1} \\ w^{[2]}_{1,2} & w^{[2]}_{2,2} & w^{[2]}_{3,2} \end{bmatrix}, \mathbf{w}^{[3]} = \begin{bmatrix} w^{[3]}_{1,1} & w^{[3]}_{2,1} \end{bmatrix} \tag{2.18}$$

Similar notations will be used for the biases. In the example above this would be given as:

$$\mathbf{b}^1 = \begin{bmatrix} b^{[1]}_1 \\ b^{[1]}_2 \\ b^{[1]}_3 \end{bmatrix}, \mathbf{b}^2 = \begin{bmatrix} b^{[2]}_1 \\ b^{[2]}_2 \end{bmatrix}, \mathbf{b}^3 = \begin{bmatrix} b^{[3]}_1 \end{bmatrix} \tag{2.19}$$

In the area of machine learning forward- and backward-propagation is frequently used in order to train and use the network as mentioned earlier in the report. A simple example of this with respect to the basic network in Figure 2.4 will now be given in order to show just how many tune-able parameters that exist in the network. The output of the network would be given by forward propagating the inputs as follows:

$$\begin{aligned} y_1 &= g_3(w^3_{1,1}a^2_1 + w^3_{1,2}a^2_2 + b_3) \\ a^2_m &= g_2(w^2_{m,1}a^1_1 + w^2_{m,2}a^1_2 + w^2_{m,3}a^1_3 + b^2_m) \\ a^1_m &= g_1(w^1_{1,m}u_1 + w^1_{2,m}u_2 + b^1_m) \end{aligned} \tag{2.20}$$

The output of the network, y_1, can thus be seen as a combination of functions from the inputs:

$$y_1 = f_3(f_2(f_1(\mathbf{u}))) \tag{2.21}$$

which displays the complexity possible to capture in Neural Networks. In Figure 2.4 only two hidden layers are used, however as Eq. (2.20) hints, the combination of linearities and nonlinearities provides a powerful function-approximation. In many cases, even deeper networks are used, making the possibilities of network greater and more complex.

2.1.7 Feature Scaling

A neural network may have multiple inputs, from 1 up to N, with all N-signals capturing different behaviours. More importantly, the signals may measure things of a varying scale; for instance, one signal could stretch the region {0,10} while another {-100,1000}. For machine learning, the approximation of a function is based on patterns and the relative value of the signal (rather than the absolute value) is what is to be considered. The different signals have different scales even though they are representing comparable objects. Each signal has a different type of behaviour when studied over time and should, therefore, be scaled with respect to it. The transformation, or scaling, of the data can be of different sorts but defined as follows,

$$y_i = f(x_i)$$

with f being the function scaling data-point x_i into y_i. If the values, x_i, of a signal, s_n, are equally distributed from the minimum, s_{min}, to the maximum, s_{max}, a reasonable technique is the *Normalisation* method described in [20], where f is defined as follows

$$y_i = f(x_i) = \frac{x_i - s_{min}}{s_{max} - s_{min}} \tag{2.22}$$

linearly scaling the values x_i of signal s_n to a unit range {0,1}. If the values of s_n instead have a more heavy distribution, or much of the data concentrated in a specific region, the *Standardisation* method is more suitable. In this case, f is described as:

$$y_i = f(x_i) = \frac{x_i - \mu}{\sigma} \tag{2.23}$$

with μ being the mean and σ the standard deviation of signal s_n. The signal s_n is then transformed to a distribution with zero-mean and unit-variance.

2.1.8 Regularisation

If a machine learning network is trained for an extensive amount of time there is a chance that the network learns all the features of the training data, including noise. This is known as overfitting. When the network becomes overfitted, it starts to learn the features of the noise in the data and the network's ability to generalise decreases. It would then perform better on the training data itself, but worse on previously unseen data. Since the data in a real-time situation is previously unseen for the network, this behaviour is undesirable.

A way to handle the overfitting problem is to use regularisation, where one of the most common methods is the *early stopping* method. During the training of a neural network, the training loss usually decreases, while the validation loss

eventually plateaus, and sometimes starts to increase. This is due to the network becoming overfitted and is learning features specific to the training data. *Early stopping* would stop the training when the validation loss does not improve over x-epochs. Hence, early stopping helps with avoiding overfitting through regulating the number of epochs that the network will use during the training process [1].

Regularisation methods are designed to lower the generalisation error and this is done in different ways. Some focus on adding terms to the loss-function in order to regularise the weights, such as the L^2- and L^1-*regularisation* techniques. Other methods, such as *Dropout*, eventually increase generalisation of the network by forcing each neuron to become more individually dependent. This is done through zeroing out a number of weights each epoch which means that the neurons will not be able to rely on other neurons being active. There exist a big variety of regularisation methods that are more or less useful in generalising a model depending on the specific case and task [1].

2.1.9 Batch Normalisation

Batch Normalisation is an optimisation technique designed to improve the stability and reduce the training time of a network [21]. The complexity and time it takes to train a network are increased by *internal covariate shift*, to which the authors of [21] introduce a method to decrease the complexity of the training. A hidden layer's input, $\mathbf{x}_i$ is affected by the values of the parameters in the previous layers $(\mathbf{h}_{i-1}, \mathbf{h}_{i-2}, ...)$ meaning that a small change in an earlier stage is increased further on. The distribution of the input to a layer, $\mathbf{h}_i$, changes as the parameters of the previous layers do; this continuous adaption required by the layer is known as the internal covariate shift. This slows down the training and makes saturated nonlinearities, such as the common *ReLU*, especially hard to deal with. Proposed by [21], Batch Normalisation is a way to handle this, by whitening the inputs. For a given batch, the data is normalised as follows,

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{2.24}$$

where μ and σ are the mean and standard deviation of the batch, while ϵ is a small positive number to avoid division by zero. x_i here represents the output of the layer in one dimension, as $x_i = w\ u + b$, where w, b and u are the weight, bias and input. Choosing to normalise the output of the layer, before any saturating activation function is applied, it is more likely to be distributed non-sparse and symmetric, behaving more Gaussian. Finally, the output is scaled and shifted

$$y_i = \gamma \hat{x}_i + \beta \tag{2.25}$$

The "expressive" power of the layer, in the sense of what a layer represents, may be reduced when normalising each of the inputs. In Eq. (2.25), the introduced variables γ and β have to be learned by the network to allow the output to express any mean or standard deviation. This parameterization, rather than just using $\mu = 0$ and $\sigma = 1$, makes the network more expressible while it is still easily learned in backpropagation.

Batch Normalisation may also reduce generalisation error and might make other regularisation methods (see Section 2.1.8) redundant [1]. This is due to the noise introduced in the estimation of the parameters in Eq. (2.24) - (2.25).

2.1.10 Learning Rate

The learning rate of a neural network is the parameter which specifies the step-size used during the back-propagation, i.e. how much the weights of the network will be updated based on the gradient of the objective function. The learning rate is seen denoted as α in Eq. (2.16). The equation shows that the update of the weights is simply scaled by the size of the learning rate and is therefore set to a number between 0 and 1. However, the process of finding a suitable learning rate for the network can be tedious. Choosing a large learning rate will speed up the training process but may lead to the algorithm missing an optimum by overshooting it. This could lead to a network that does not converge to a local or global minimum but instead fluctuates around it or simply diverges. On the other hand, choosing a small learning rate could lead to a higher training time [18].

Since a high learning rate speeds up the training but is less precise, the combination of a high learning rate initially and a lower learning rate later on could be very useful. The algorithm would quickly move towards a local or global minimum and as it gets closer the learning rate is decreased to slow down the process and converge towards the minimum. This concept is what adaptive learning rates are based on. Adaptive learning rates are methods that alter the learning rate during the training process. The simplest adaptive learning rate method is to reduce the learning rate when the loss function stops decreasing. The adaptive learning rate methods typically include a "learning scheduler", with the task of decreasing the learning rate according to specific criteria, for instance when the loss function is stable along several epochs. This process will then continue during the entire training phase [1].

A very specific but important case for this project is learning rates during transfer learning. When performing transfer learning as explained in Section 1.4 the network close to convergence since the weights are already pre-trained. Hence, a low learning rate is to prefer for transfer learning in order to avoid overfitting of the network [10].

2.2 Recurrent Neural Network

The Recurrent Neural Network, RNN, is a type of neural network that very much resembles the multi-layer perceptron networks, Section 2.1.6. The major difference is that the RNN does not only contain dependencies within the current states of the network but also with states in previous time steps. This essentially allows RNNs to map the history of previous inputs to each output which makes RNNs optimal for dynamical systems which tends to have dependencies on previous time steps.

The time dependency makes RNN a bit more complex to work with compared to other networks. This is due to the fact that the weight updates need to be based on not only the state of the neurons at the current time step but also on the state

of neurons in previous time steps. In order to visualise this, RNNs are typically unfolded over time. Figure 2.5 shows a simple network structure unfolded over time and provides a simplistic view of how the weights' update depends on previous time steps.

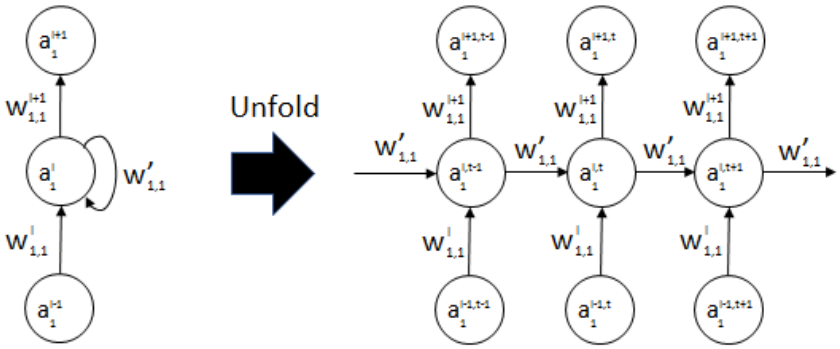


Figure 2.5: A Recurrent Neural Network unfolded over time in order to visualise the time dependencies in the network and how to work with them.

The forward propagation in an RNN is performed through considering both the external inputs to the activation function from neurons in previous layers but also by looking at previous outputs from the activation function at the current neuron. Hence, the first part of the forward propagation is performed as for a normal feedforward network but with an additional part that handles the time dependency from previous time steps. For the i:th neuron in the l:th layer at time step t this update can be explained as:

$$z_i^{l,t} = \sum_{n=1}^{N} w_{n,i}^{l} a_n^{l-1,t} + \sum_{h=1}^{H} w'_{h,i} a_h^{l,t-1} \tag{2.26}$$

$$a_i^{l,t} = g(z_i^{l,t}) \tag{2.27}$$

Here w^l denotes the weights that connect the output of layer $l-1$ to layer l and w' denotes the weights that connect the previous time step to the current time step, [4].

For the backward propagation, we consider the most common type of algorithm for Recurrent Neural Networks; Backward Propagation Through Time (BPTT), [22]. Since the output of the activation function affects not only the next hidden layer but also the same layer in the upcoming time step the resulting effect on the loss function can be written as:

$$\delta_i^{l,t} = \frac{\partial \mathcal{L}}{\partial z_i^{l,t}} \tag{2.28}$$

$$\delta_i^{l,t} = g'\left(z_i^{l+1,t}\right)\left(\sum_{k=1}^{K} \delta_k^{l,t} w_{ik}^{l+1} + \sum_{h=1}^{H} \delta_h^{l,t+1} w'_{ih}\right) \tag{2.29}$$

The complete weight update is then achieved as a sum over the whole time sequence from time step 1 up until the current time step T:

$$\frac{\partial \mathcal{L}}{\partial w_{ik}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial z_k^{l,t}} \frac{\partial z_k^{l,t}}{\partial w_{ik}} = \sum_{t=1}^{T} \delta_k^{l,t} a_i^{l,t} \tag{2.30}$$

However, today long term time dependencies in Recurrent Neural Networks are known to be problematic. When the time sequence T grows large it will cause the gradient to become very small or very large, this is known as the Vanishing Gradient Problem. The problem was discovered in the early 1990s and revolves around the fact that during back-propagation the gradient grows exponentially larger or smaller not only with each added layer but also with each recurring time step. Hence, the gradient can become very small or large even when only a few layers are being used, [23]. A solution to the Vanishing Gradient Problem was introduced with the Long Short-Term Memory networks.

2.3 Long Short-Term Memory

Many attempts were made to solve the vanishing gradient problem and a well-known solution is the Long Short-Term Memory, LSTM, approach. LSTM cells are using gate units in order to control how much of the old memory that should be passed on through the network. There are in total three different gates that are used in an LSTM cell and these are the input gate, output gate and the forget gate. The input and output gates are multiplication factors of the in- and output while the forget gate is multiplied with the cell's previous states. The concept with gated units was introduced in 1997 by Sepp Hochreiter and Jürgen Schmidthauber, [5]. The gates allow the network to change how much previous time steps will affect the current gradient through the forget gate. This means that the network can scale and zero out the effect from previous time steps, or with other words forget parts of its memory. By setting the network to forget parts of its memory before the gradient becomes too small or too large the vanishing gradient problem can be solved. This can be viewed as the time sequence T being truncated to only maintain a time dependency on a shorter window of time instead of all the previous time steps. The updating equations for an LSTM network is more complex than for a normal RNN. In Figure 2.6 we can see an LSTM cell which contains the input, input gate, forget gate, output gate, and the output. All of the gates are typically activated through a sigmoid function and the equations for each gate depends on the states of the neurons in the previous layer but also on the state of the neurons in the previous time step. The equation for the forget gate and the input gate is as follows:

$$f_i^{l,t} = \sigma\left(b_i^f + \sum_n U_{i,n}^f a_n^{l-1,t} + \sum_j W_{i,j}^f a_j^{l,t-1}\right) \tag{2.31}$$

$$g_i^{l,t} = \sigma\left(b_i^g + \sum_n U_{i,n}^g a_n^{l-1,t} + \sum_j W_{i,j}^g a_j^{l,t-1}\right) \tag{2.32}$$

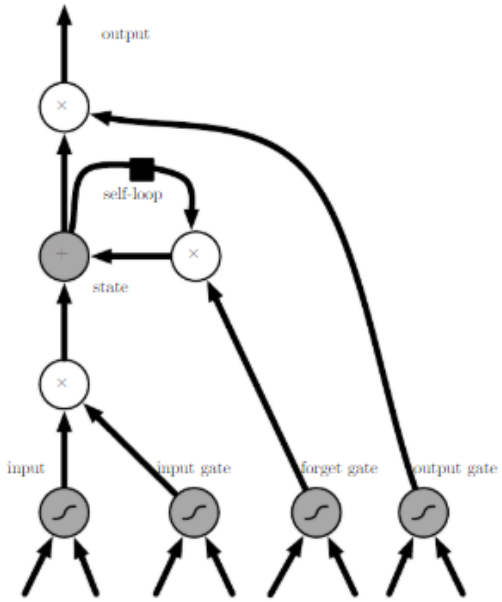


Figure 2.6: An LSTM Cell with the recurrent output from the previous time step and the output of the previous layer as input. Also seen is the input gate, forget gate and the output gate. The picture is copied from the Deep Learning textbook by Ian Goodfellow, Yoshua Bengio and Aaron Courville [1].

In order to perform the update $f_i^{l,t}$ is then multiplied with the state of the cell in the previous time step while $g_i^{l,t}$ is multiplied with the current input:

$$s_i^{l,t} = f_i^{l,t} s_i^{l,t-1} + g_i^{l,t} \sigma \left(b_i + \sum_n U_{i,n} a_n^{l-1,t} + \sum_j W_{i,j} a_j^{l,t-1} \right) \tag{2.33}$$

Finally, the output of the cell, $a_i^{l,t}$, is given by:

$$a_i^{l,t} = \tanh \left(s_i^{l,t} \right) q_i^{l,t} \tag{2.34}$$

Where $q_i^{l,t}$ is the value of the output gate which is calculated as:

$$q_i^{l,t} = \sigma \left(b_i^o + \sum_n U_{i,n}^o a_n^{l-1,t} + \sum_j W_{i,j}^o a_j^{l,t-1} \right) \tag{2.35}$$

Information and equations found in the *Deep learning textbook*, [1] and Alex Graves book *Supervised Sequence Labelling with Recurrent Neural Networks*, [4].

2.4 Convolutional Neural Network

A convolutional neural network, CNN, is a type of the deep neural network class which is commonly used in applications that form some sort of grid [8]. Much like the classical feedforward network, the Convolutional networks consist of neurons with

weights and biases, as described in Section 2.1.6. However, unlike the previous, the most common example is the use of CNN for analysis and recognition of objects in images, where the image is of a 2D-grid. The network is designed to see and look for features in the image (or whichever other type of input sequence), in order to determine whether or not a feature learned by the network is appearing or not. The network is *shift invariant*, meaning that features may present anywhere in the image and the network should still be able to identify them.

Again, the aim of the network is, given a certain input, to predict or observe a state so as the normal MLP 2.1.6 However, as the input to a CNN is a grid (Image, Time-series in 2D), an MLP network would require a lot of parameters to be able to express the system. For instance, an image with three channels (RGB) and of the size (100×100) would require in the first hidden layer, $100 \times 100 \times 3 = 30\,000$ weights for each neuron in the following layer. As the amount of pixels in height and width increases so does the number of parameters, rapidly. The Convolutional Neural Networks avoid this massive amount of parameters and the risk of overfitting by sharing weights in between different neurons. As seen in Figure 2.7, the input vector x is multiplied by the weight vector w which then finally is summarised in the hidden layer, h, as z.

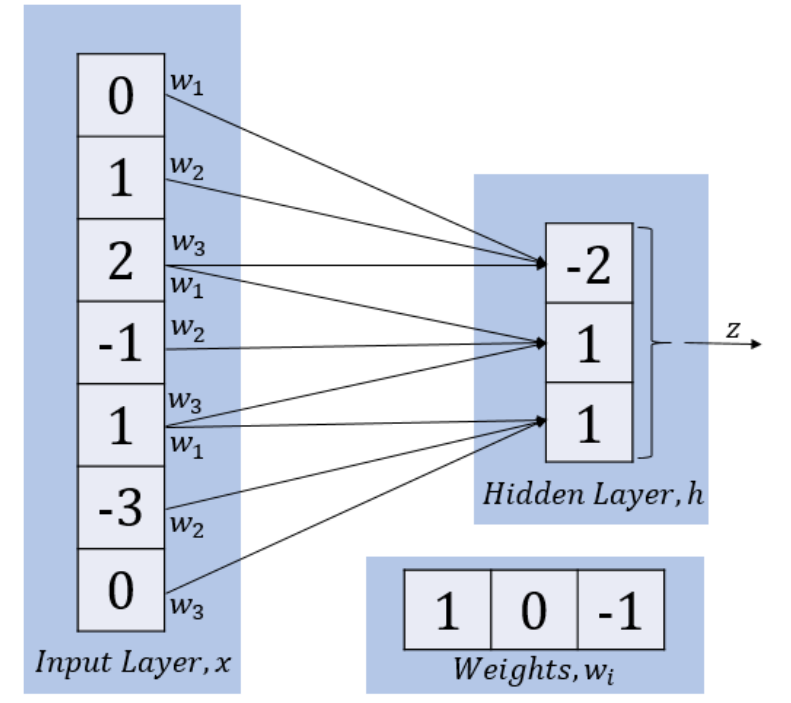


Figure 2.7: How convolutional networks make use of shared weights. Here only 3 weight are in use, when for a fully connected layer it would require 7 times the weights.

The weight-vector now consists of only 3 weights (selected by design), while a fully-connected layer would instead require 7×3 weights in the same layer. Before how parameter sharing works is explained further, the most fundamental function of the CNN will be accounted for.

2.4.1 The Convolution Operator

The name, Convolutional Neural Networks, comes from the fact that, at least, one layer of the network applies the mathematical operation sharing the same name. Convolution is a mathematical operation on two functions which produces a third one. It is an operation commonly used in signal processing for applying filters/kernels, and is defined as follows:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.36}$$

with $*$ being the sign of the convolution operator. In Eq. (2.36), the function *f(.)* is the **input** to the convolution, with *g(.)* being called the **filter** or **kernel**. The output produced by the operation is referred to as the *feature map* in Machine Learning, which is explained further on. The function *g(.)* differ from application to application, but it may be said that in general it is a function designed to give a response of how one function affects and correlates with another.

In Eq. (2.36) the case considered is continuous, however the case with machine learning at hand is the discrete case, rather, as is defined as:

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau) \tag{2.37}$$

When considering the input to a convolutional neural network is a spatial, grid-like, object, the convolution is often defined, and performed, over multiple axes at the same time. A 2D-image I with dimensions (m, n) is convoluted with a filter K of size (i, j) over both axis's at the same time as an extension of Eq. (2.37), providing the output, O, as:

$$O(I, K) = (I * K)(i, j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m, n) \tag{2.38}$$

2.4.1.1 Convolutional Filters and activation maps

The learning of Convolutional Neural Networks is as always learning the weights and biases. However, the learnable parameters are now stacked into filters (or kernels). The filters are "slid" across the input image during the forward propagation. This is actually the convolution operator as described in Eq. (2.38). The responses, the functions generated from the convolutions, are referred to as *feature maps*. An example of how convolution could look for an image of size (6×6) and a filter of size (3×3) can be seen in Figure 2.8.

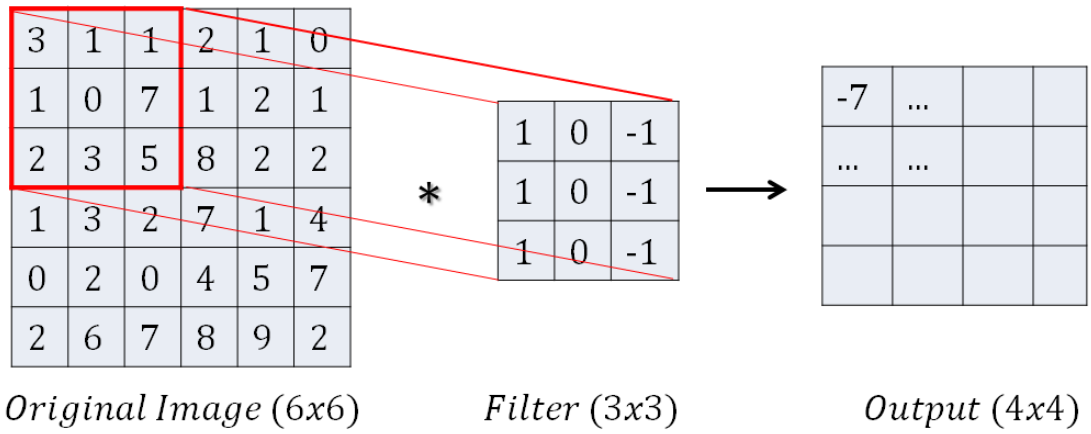


Figure 2.8: Convolution between an image and filter, providing an output.

In Figure 2.8, the filter is "slided" across the image with a **stride** set to 1. This means the filter is moved one pixel, or step, for every convolution over the image. This could be seen in Figure 2.9 where the filter is shifted one pixel to the right. This operation is continued in downwards direction as well until the entire image has been convoluted.

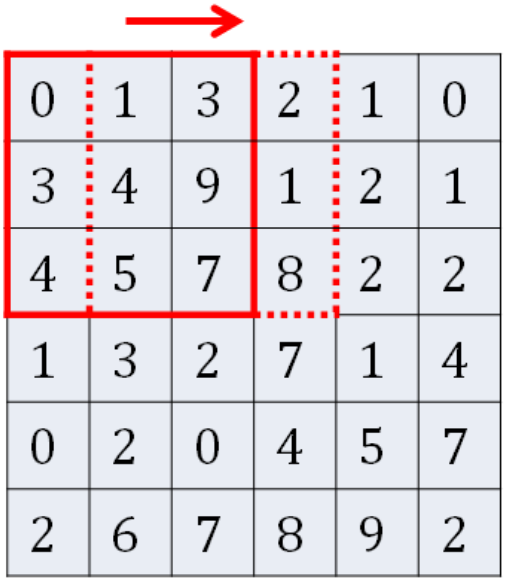


Figure 2.9: The "sliding" of a filter over the input to the convolutional layer. Here with a stride set to 1.

The result of the convolution, the output, can sometimes be called *activation maps*. This is because, when fully trained, the filters (or kernels) are *feature detectors* tuned to a specific feature. The "response" of a filter applied to an image is an *activation map* which detects whether or not a certain feature is visible in the image. The amount and size of the filters, which sometimes is referred to as the **receptive**

field of the network, are parameters of design. The receptive field of the network determines how much of an image or sequence the network "sees" at once. For instance, if the input to the network is a time-sequence with multiple signals then the receptive field (filter size $n \times n$) tries to correlate $n-$signals over $n-$seconds.

2.5 Hyperparameter Iteration Techniques

There are multiple hyperparameters that could be tuned for a neural network: from number neurons and filter sizes in each layer to the depth of the network as well as which optimiser to use. Yet, there is no ground truth of what values to set for these hyperparameters. Instead, different iteration techniques are often used to find suitable values for the task at hand: in this project Grid Search and Random search will be presented in Sections 2.5.1-2.5.2. Often, these methods can be combined by finding trends in the Grid Search and fine-tuning these with the Random Search.

2.5.1 Grid Search

Grid search is a structured way to iterate over hyperparameters and other decision variables for a network. Taking a number of hyperparameters, often three or fewer [1], and iterating over different values and combinations of these. This concept is visualised in Figure 2.10, where a 2D-problem (two hyperparameters) is formulated.

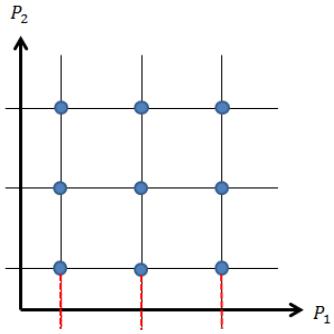


Figure 2.10: The concept of grid search considering two parameters, one on each axis.

A discrete, finite, number of values for each parameter, P_1 and P_2, is decided and explored. The algorithm trains a model for each combination of the parameters, P_1 and P_2. The impact and combination of different hyperparameters are tested by evaluating the models on the validation test-set, hopefully displaying trends and optimal choices for the parameters. Not only does this provide a better choice of parameters, but it also displays the effect or impact a certain parameter has on the performance of the network. It may be found that some parameters matter more than others: In Figure 2.10, two parameters are tested. It could be so that no matter the choice of parameter P_1 an increased value of parameter P_2 always yields a better result. In this example parameter P_2 is more important, meaning that iterating over the parameter P_1 could be a waste of time. This is one of the drawbacks of the

algorithm: since it iterates over a fixed number of values for the parameters, much of the time may be spent finding the optimal value for a parameter that does not have a big impact on the performance of the network. Also, the number of iterations increases exponentially as more hyperparameters are introduced. For this reason, an alternative method may be used.

2.5.2 Random Search

Compared to the grid search a random search is a less fixed way of iterating over parameter values. Instead of setting up specific values for parameters 1 and 2 to iterate through a random search sets the parameter values randomly for each iteration. This means that you no longer have a specific combination of the parameters that you iterate through. The process is visualised in Figure 2.11, what can be noted is that with the same number of combinations as in the grid search in Figure 2.10 the random search here covers a lot more individual values for each parameter (the red lines).

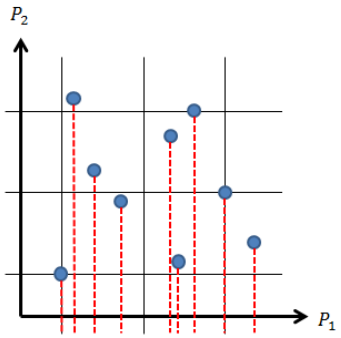


Figure 2.11: The concept of random search considering two parameters, one on each axis.

3
Data Processing

In order to create models of the sensors, a better understanding of the physical model is needed. The different signals of the system need to be evaluated to be able to choose the suitable ones to use as inputs to the neural networks. Additionally, a crucial step when working with machine learning is to have data that is structured and processed. Hence, the gathered data needs to be properly investigated and processed.

3.1 The Physical Model

As mentioned in Section 1.2 the focus of this project is to create virtual models of physical sensors. The sensors are subsystems of a complete engine, the physical model of the project. The engine is a 6-cylinder 8 litres diesel engine with a top power of 250kW. It is equipped with a turbocharger and common rail fuel injection system. The engine fulfils the stage 5 emission standards for off-road and it is currently being used in a variety of vehicles where the main area surrounds construction vehicles.

An engine is a dynamical system consisting of a wide range of subsystems. The complexity of the different subsystems and their signals varies: some can easily be described through equations and formulas where others cannot. Especially complex are the signals which rely on the internal combustion process of the engine.

The internal combustion system itself is can be divided into four different steps. The first step is the intake. This is where the air is pushed into the combustion chamber and is mixed with vaporised fuel. This step ends when the piston reaches its lowest position and the second step begins; compression. Here the mixture is compressed when the piston is going back up until it reaches its highest position. The third step is where the mixture ignites, this is the internal combustion step which was mentioned before. It is a chaotic process where temperatures and pressures spike rapidly when the fuel-air mixture is ignited. It is also here emissions such as NO_x and Soot are created through a chaotic chemical transformation. The exhaust gases are then pushed out of the combustion chamber in the fourth and final step. These four steps are performed over two rotations of the crankshaft in the engine. Considering an engine that runs at 2000 RPM it means that the complete process is performed more than 10 times per second, creating the highly transient behaviour in the engine, [24].

Hence, developing models that properly describe this process is of high difficulty and thus creating models dependent on the process can be equally difficult. Two of the three sensors considered in this project measure the pollutants of the

engine. These pollutants are created during the internal combustion process and are therefore complex to model with analytical modelling techniques.

At Volvo Penta, *Hardware-in-the-Loop* (*HiL*) systems are being used to test and verify physical components and software in the engine. The models of the subsystems which are developed in this project will be a part of developing and improving the HiL-systems at Volvo Penta. Additionally, such models could eventually be implemented in a real-time engine. This would allow the ECU of the engine to make decisions based on the predictions of the models, such as if acceleration or deceleration should be done in order to keep the emissions at a certain level.

3.2 Data-set and test cycles

The data is gathered from the engine described in Section 3.1 and will be evaluated and organised into a structure that allows for easy handling when training. This means that one repository for the LSTM Network and one for the CNN will be created. In order to develop a model that can be used in a real-time engine, the data will be sorted so that only signals from the Engine Control Module (ECM) will be a part of the input.

The complete data-set consists of 15 test cycles provided by Volvo Penta, where each cycle contains $10\,000 - 100\,000$ samples in a sequence. A sample is built from 80 signals and represents a discrete step in continuous time. The sample-time is $10Hz$. The structure of each test cycle may be seen in Figure 3.1. All 80 signals are either measured from a vehicle, in the test-cells or physically modelled by test-engineers at Volvo Penta.

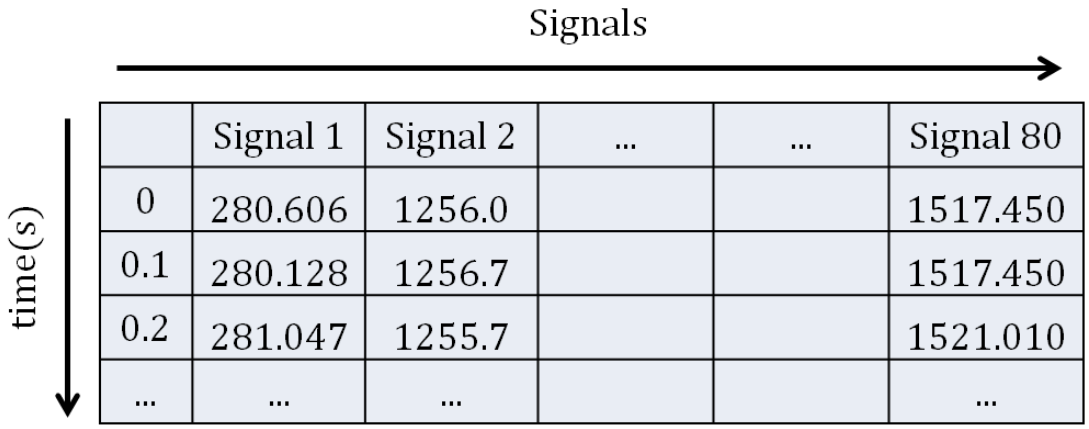


Figure 3.1: The structure of the data, as provided from the test-cycles. Each cycle consists of 80 signals and is sampled with $10Hz$

The test cycles used in this project are a subset of all different test cycles run at Volvo Penta. The various cycles aim to represent how the engine performs in a variety of its running conditions and the complete setup of data-sets consists of the following test cycles:

- **Non-Road Transient Cycle** - The Non-Road Transient Cycle, NRTC, is a transient cycle, meaning the data is gathered continuously and captures the behaviour of the variables as the speed and load of the engine changes. The NRTC cycle is performed in two parts where the engine is first run with a cold start (from room temperature), soaked and cooled for twenty minutes before the second sequence begins. This is the cold part of the cycle. For the second run, heat remains from the previous sequence [25], this is the warm part of the cycle. Additionally, two types of NRTC cycles will be used during the project. The one explained above will be denoted *NRTC-cw*, cold-warm. The other type, *NRTC-ww* (warm-warm), is run so that both of its sequences are warm from the start.
- **Part Load Map** - The Part Load Map, PLM, is used only in the development stages of the engine. The data-set consists of steady-state samples that try to cover the complete speed and load rate of the engine. Steady-state implies that the samples only contain data where the engine is fixed to a certain speed and load, with slight deviation due to the fact that it is a physical system.
- **Load Response** - The load response cycle aims to test the engine at maximum torque for different engine speeds. It is performed through running the engine at specific engine speeds with low torque. The torque is then pushed from a low value all the way to maximum torque while measurements are being taken. The procedure then continues for the next engine speed.
- **J2** - The J2 cycle is developed by the Volvo Penta PEMS (Portable emissions measurement system) team. It is designed to imitate an engine that is run very poorly. The cycle is performed with a very low load compared to the NRTC cycle.

3.3 Target signals

As mentioned in Section 1.2, three output signals will be modelled and evaluated. The signals differ from each other both in how they are measured but also in the complexity of each signal's behaviour. The following signals are being considered:

- **Flw_FuelDiesel** $[g/s]$ - The total amount of fuel injected per second. The signal is considered to be easy to model since it has a direct relationship to the input signals. This signal will be denoted as Flw_FuelDiesel or Flow Fuel Diesel in the report.
- **Conc_NOx** $[ppm]$ - The amount of NO_x emissions in the exhaust gases. This signal is a pollutant and as described in Section 3.1 will, therefore, be a more complex signal to model. This signal will be denoted as Conc_NOx or Concentration of NO_x in the report.
- **Conc_Soot** $[mg/kg]$ - the amount of Soot in the exhaust gases. Just as the NO_x-signal this signal measures a pollutant and will, therefore, be complex to model. Currently, due to the unpredictable behaviour of Soot, a physical model does not exist at Volvo Penta. This signal will be denoted as Conc_Soot or Concentration of Soot in the report.

3.4 Selection of input-signals

As described in the first parts of Chapter 3, the provided data-sets consists of 80 signals which are either measured or modelled. The modelled parameters can be either an approximate value set by the ECM, or modelled as a combination of measured signals. Creating a model that consists of 80 input-signals is both time-consuming (when training the neural networks) and all signals are not always available. In order to ensure that the model is based on available signals, a first step is to narrow down the data-set and remove the signals which do not exist in the ECM. Among the 80 provided signals are also the output target signals, i.e. the signals to be modelled in this project. Also, these are excluded from the data-set, and placed as targets, or desired outputs, with respect to the input signals. The signals are then further reduced based on their relation to each given output signal. This step could be performed by letting the network decide which input signals are redundant during the training step. However, since expertise about the relationship between the input and output signals is available at the company this is done by hand. This saves time when training the network, as fewer weights are needed to cope with the inputs, see Section 2.1.6 for a feedforward network or 2.4 for the increased complexity of more signals for a convolutional neural network.

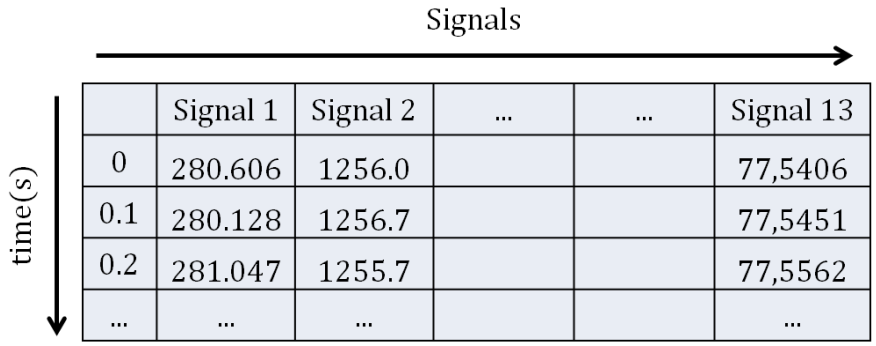
Finally, the following 13 signals are selected as the input to the Neural Network. These are the ones deemed to have a physical relation to the output signals, while still being part of the ECM:

- **Engine Speed** [rpm] - The speed of the engine. Measured by a sensor.
- **Fuel Value** [mg/str] - The amount (mg) of fuel consumed by each stroke (str) of a piston. A modelled parameter.
- **Injection Angle** [deg] - The position of the crankshaft when the fuel injection occurs, specified by the number of degrees before the piston reaches its highest point. Modelled parameter.
- **Rail Pressure** [bar] - The pressure in the fuel rail (before the fuel splits into smaller rails leading to the cylinders). Modelled parameter.
- **Wastegate Position** [%] - A percentage to set how much the wastegate should be open. Modelled parameter.
- **EGR Position** [%] - A percentage to set the position of the EGR valve. Modelled parameter.
- **Exhaust Temperature** [**°C**] Temperature in the exhaust gases. Measured parameter.
- **Main Injection** [mg/str] - The amount (mg) of fuel injected in the main-injection per stroke (str). A modelled parameter.
- **Post Injection** [mg/str] - The amount (mg) of fuel injected in the post-injection per stroke (str). Modelled parameter.
- **Pre Injection** [mg/str] - The amount (mg) of fuel injected in the pre-injection per stroke (str). Modelled parameter.
- **Inlet Pressure** [kPa] - Pressure at the inlet of the cylinder (after compressor). Measured parameter.
- **Pre Injection Angle** [deg] - The position of the crankshaft when the pre-injection occurs, specified by the number of degrees before the piston reaches

it's highest point. A modelled parameter.

- **Throttle Position** [%] - A percentage that indicates how open the throttle is. 100% - fully open and 0% - fully closed. Measured parameter.

This reduces each cycle to a sheet containing a sequence of $10\,000 - 100\,000$ samples of each of the 13 signals. The sample-frequency is still set to $10Hz$. The new structure of each test cycle may be seen in Figure 3.2.



	Signal 1	Signal 2	...	...	Signal 13
0	280.606	1256.0			77,5406
0.1	280.128	1256.7			77,5451
0.2	281.047	1255.7			77,5562
...	...	...			...

Figure 3.2: The structure of the data, when sorting out the 13 most significant signals. Each cycle consists of 13 signals and is sampled at $10Hz$

The 13 signals selected are used as the input to the model. In Figures 3.3-3.4, the behaviour of two selected signals may be seen during one of the 7 test-cycles provided. The test-cycle seen here is of the sort *NRTC-ww* as described in Section 3.2. These signals have different units, magnitude and behaviour throughout the cycle. The rest of the signals' behaviour may be seen in Appendix A.1.

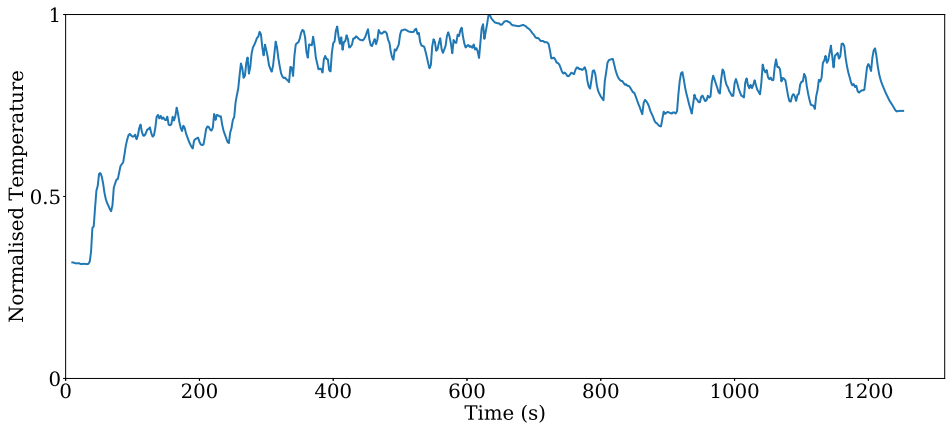


Figure 3.3: The behaviour of the *Exhaust Temperature* signal throughout the test-cycle *NRTC-ww*.

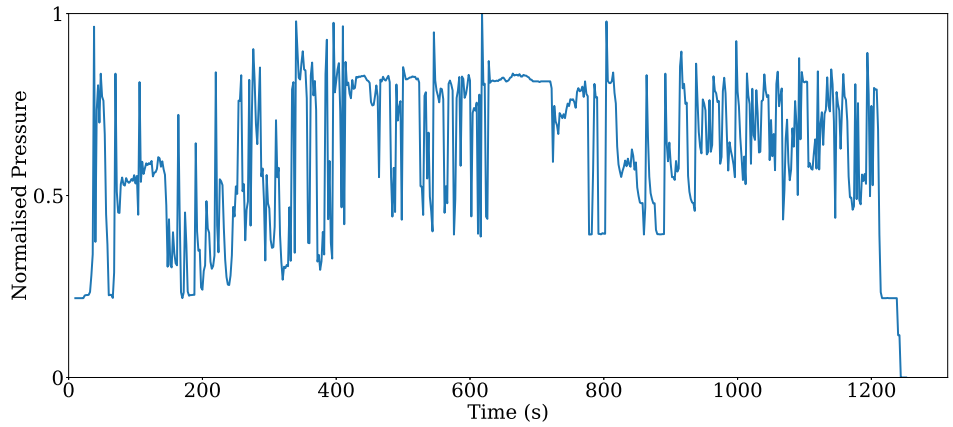


Figure 3.4: The behaviour of the *Rail Pressure* signal throughout the test-cycle *NRTC-ww*.

3.5 Feature Scaling and processing

As described in Section 2.1.7 signals might be of different scales, which may be a problem for the neural network to cope with. Of the 13 signals selected as inputs to the Neural Network, the scales are varying in between them quite a lot, meaning that feature scaling is necessary. Seen in Table 3.1 are the maximum and mean values of the 13 input signals. As seen, the signals are of different magnitudes, while, as described in Section 2.1.7, it is pointed out that the relative value of the signal is the important part.

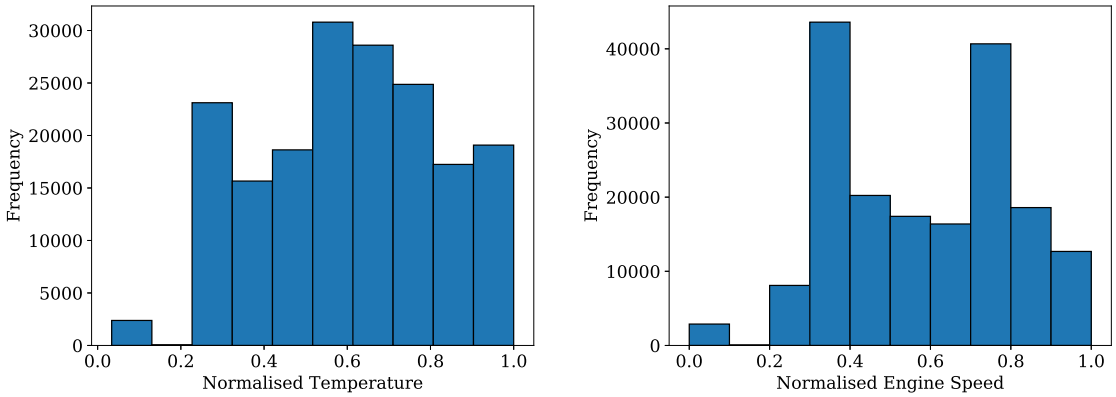
	unit	max	μ
Engine Speed	rpm	2407.29	1399.17
Fuel Value	mg/str	159.73	54.62
Injection Angle	deg	10.08	3.64
Rail Pressure	bar	1800	866.03
Wastegate Position	%	95	58.07
EGR Position	%	95.02	45.23
Exhaust Temperature	°C	505.5	309.72
Main Injection	mg/str	156.23	51.84
Post Injection	mg/str	6.6	0.99
Pre Injection	mg/str	17	1.96
Inlet Position	kPa	297.90	151.97
Pre Injection Angle	deg	17.21	8.38
Throttle Position	%	81.18	54.02

Table 3.1: Seen is the unit, maximum and mean-values of all 13 signals.

Seen above in Table 3.1, the scales of the input signals are of different units, meaning the minimum and maximum values vary in order of magnitude. The maximum values for, for instance, *Engine Speed* is 2407.29 *rpm* while the corresponding

number for *Pre Injection* 17 mg/str. While it would be possible to train a model under this premise, it would substantially slow down the process if the solution even converges. For that reason, the signals are scaled, either by normalising them by their minimum and maximum values as in Eq. (2.22) or by normalising based on the distribution as in Eq. (2.23).

As described in Section 2.1.7, the features, the input-data, are scaled differently depending on whether or not a clear distribution may be found. Therefore, each signal is analysed based on its distribution in the form of a histogram. Seen in Figure 3.5 are the histograms of *Exhaust Temperature* and *Engine Speed*. These histograms are calculated over the entire provided data-set, unlike the plots displayed in Figures 3.3-3.4. This is done to capture the "true" behaviour of the signals, not only the behaviour of a certain test-cycle.

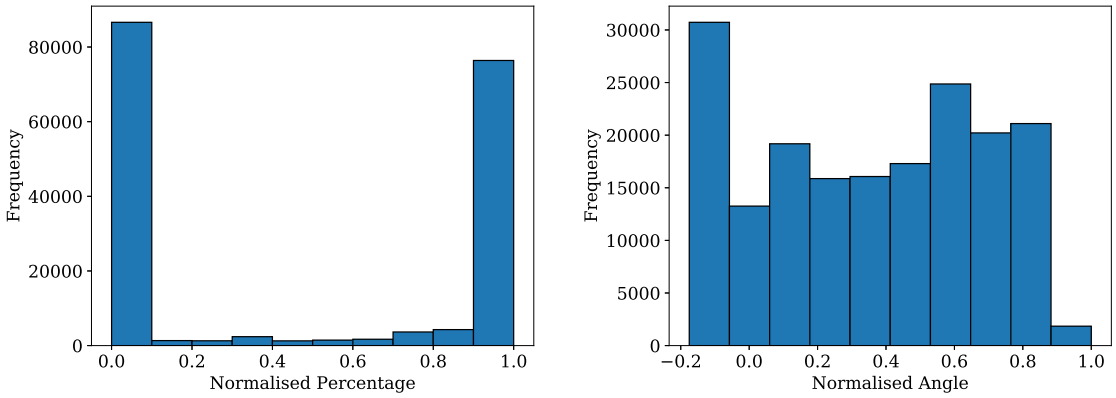


(a) Exhaust Temperature **(b)** Engine Speed

Figure 3.5: The distribution of two signals with different units and magnitude throughout the test-cycle *NRTC-ww*.

Both signals show clear distribution around certain values, which according to Section 2.1.7 should be scaled according to Eq. (2.23). The mean is calculated and subtracted from each sample, while dividing it by the standard deviation of the entire signal. The means and standard deviations are calculated over all the provided test-data, so that a biased test-cycle does not influence these numbers too much.

While the examples of the distribution in Figure 3.5 show clear distribution around certain values, other signals show different behaviour. In Figure 3.6, the distribution of signals *EGR Position* and *Injection Angle* are seen. The rest of the distribution Figures may be seen in Appendix A.2.



(a) EGR Position **(b)** Injection Angle

Figure 3.6: The behaviour of two signals with different units and magnitude throughout the test-cycle *NRTC-ww*.

Comparing the four signals presented in Figure 3.5 and Figure 3.6 one can see that a difference in the distribution of the signals exists. For *EGR Position*, the distribution is heavily concentrated in two pillars, in each end of the value-spectra. The mean of the distribution is however 45.23%, as displayed in Table 3.1, meaning a distribution around this mean does not make sense. For the signal *Injection Angle* the values are somewhat evenly distributed over the spectra. Signals such as these are instead normalised with the *min-max*-scaling described in Eq. (2.22).

The signals were scaled according to Table 3.2, where most signals fall under the normalisation, min-max, scaling. Most signals have distributions similar to the ones in Figure 3.6

	Scaling
Engine Speed	min-max
Fuel Value	min-max
Injection Angle	min-max
Rail Pressure	min-max
Wastegate Position	min-max
EGR Position	min-max
Exhaust Temperature	min-max
Main Injection	min-max
Post Injection	standardisation
Pre Injection	standardisation
Inlet Position	standardisation
Pre Injection Angle	min-max
Throttle Position	min-max

Table 3.2: The table shows which scaling method that was chosen for each of the 13 signals.

With the signals scaled according to Table 3.2, corrupted data needs to be

handled. Occasionally, the sensors provide bad readings with incomplete data-gatherings. This appears in the form of *NaN*-entries in the data-sheets, entries that are not values. Often, *NaN*-entries come in a sequence so that a couple of samples are unreadable. However, the most common case is that *NaN*-entries appear at the beginning of a test-cycle, as the sensor readings have not stabilised from initiation. The initiation-sequence of those cycles is thus excluded. The frequency of *NaN*-entries in the rest of the test-cycle is often $1-10$ entries per 10 000 samples. In cases of missing values the sequence is interpolated to fill the missing point with the average of the previous and following values. With $x(t_k) = NaN$, a new value is calculated according to Eq. (3.1).

$$x(t_k) = \frac{x(t_{k+1}) + x(t_{k-1})}{2} \tag{3.1}$$

Eq. (3.1) is used linearly on sequences of *NaN*-entries. The sequence is then processed to the one of Figure 3.7.

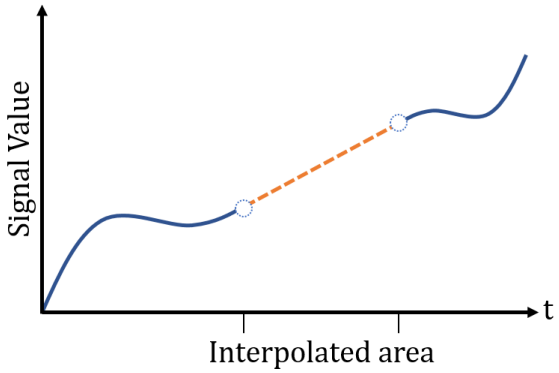


Figure 3.7: A signal processed by using the interpolation described in Eq. (3.1). The orange dotted line is the synthetic values calculated.

3.6 Reshaping the data

Once the data is scaled and processed in accordance with Section 3.5, the data is structured and split to suit the training procedure described in Section 2.1.5. The Neural Networks investigated are sequence-based so that each input is a sequence over a certain time. Each example propagated through the network is then a sequence of 13 signals and n time-steps, forming an array of size $(n \times 13)$. Since the Neural Network to be designed is an observer, it estimates the value of the output at time t_n by propagating the sequence of inputs from time t_0 to t_n.

The data from the test-cycles is reshaped to suit this demand. Each input sequence is paired with an output, so that each input, $x(t_0 : t_n)$, has a target value, $y(t_n)$. In Figure 3.8 it can be seen how n samples of the input sequence are matched with the output, target value, at time t_n.

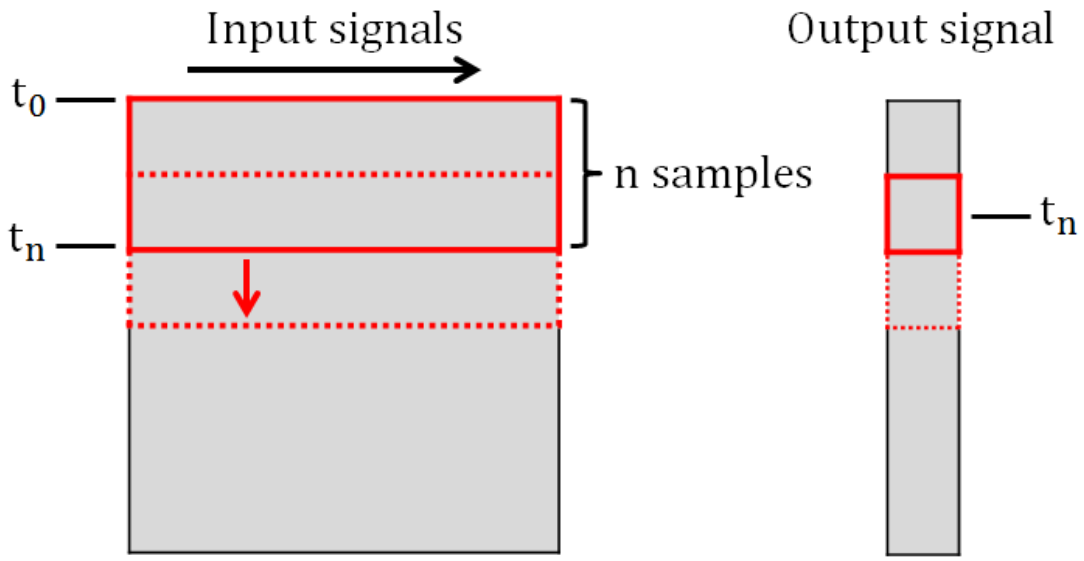


Figure 3.8: The reshaping of the test-cycles creates an input-array of size $(n \times 13)$ with a target value of (1×1).

The data is conformed and saved as an example, as illustrated in Figure 3.9, and stacked in an example-array.

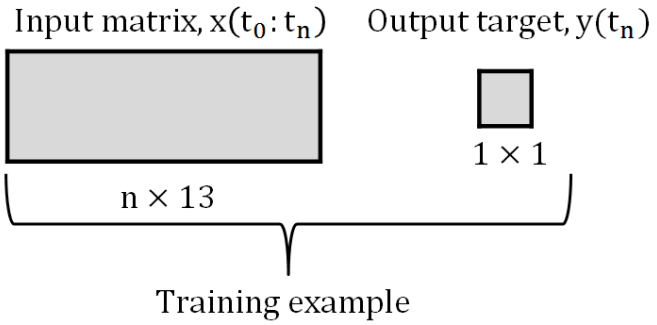


Figure 3.9: The reshaped input is saved along with the target value as a training example.

As every sequence consists of n samples, a test-cycle of K samples would be broken into a sequences of inputs $X(x)$ as

$$X(x) = [x(t_0 : t_n), \quad x(t_1 : t_{n+1}), \quad ..., \quad x(t_{K-n} : t_K)] \tag{3.2}$$

with target values as a sequence $Y(y)$ as

$$Y(y) = [y(t_n), \quad y(t_{n+1}), \quad ..., \quad y(t_K)] \tag{3.3}$$

For each test-cycle all target values before t_n will be discarded, as there are not enough inputs to be reshaped into a $n-$long sequence.

Finally, when the entire data-set is reshaped, it is stacked as an array of examples, each with an *input sequence* and a *target output*. The stacking may be seen in Figure 3.10.

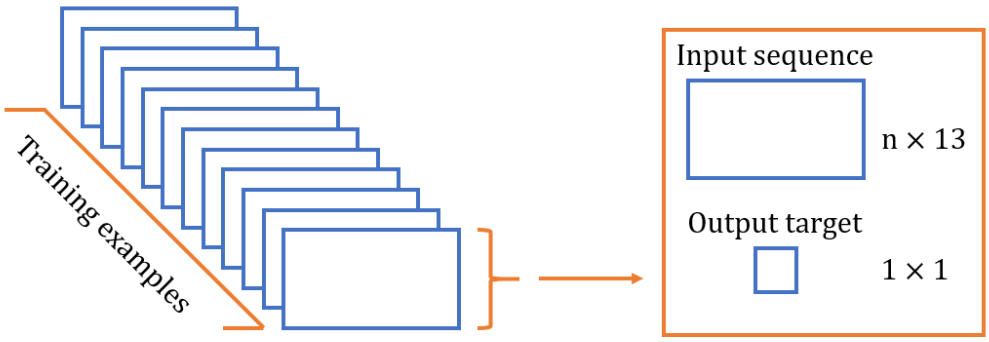


Figure 3.10: The reshaped input sequence and target data is saved as a training example and stacked in an array.

Once the entire data-set is arranged according to Figure 3.10, the data is divided into three types of sets: training, validation and test sets. This is done according to the theory described in Section 2.1.2. Instead of taking 15% of the data from an entire sequence, random examples are extracted for both the **validation** and **test** sets from all seven provided test-cycles. This is done so that a portion of all types of behaviour may be captured within all three data-sets.

3.7 Data-set split

The data-set consists of 15 cycles which types are described in Section 3.2. 12 of the cycles are of type *NRTC* while one *PLM*-, *LR*- and *J2*-cycle are provided. Two of the *NRTC*-cycles (one cold-warm and one warm-warm) and the *PLM*-, *LR*- and *J2*-cycles are from the same calibration: this is the **base-set**. The rest of the NRTC-cycles are cycles where the engine shows a different behaviour compared to the base-set.

The exact use of the cycles is described in the following Section.

Baseline models

The Baseline-models, developed in Chapter 4, will use the **base-set** for training, validation and test data. This data is reshaped into sequences according to Section 3.6. From this, 70% of the sequences will be used for training the networks and 15% to validate while training. The remaining 15% is used to test the fully trained models and to tune the hyperparameters of the networks. Additionally, the final performance and result of the baseline models are tested on one of the ten previously unseen NRTC-cycles (denoted as *NRTC 4*).

Below in Figure 3.11 the engine speed and torque of each sample in the training data and test data can be seen. The figure shows that the test data is within the

region of the training data. However, it should be noted that the samples of the test data come from a completely separate run compared to the training data as described above.

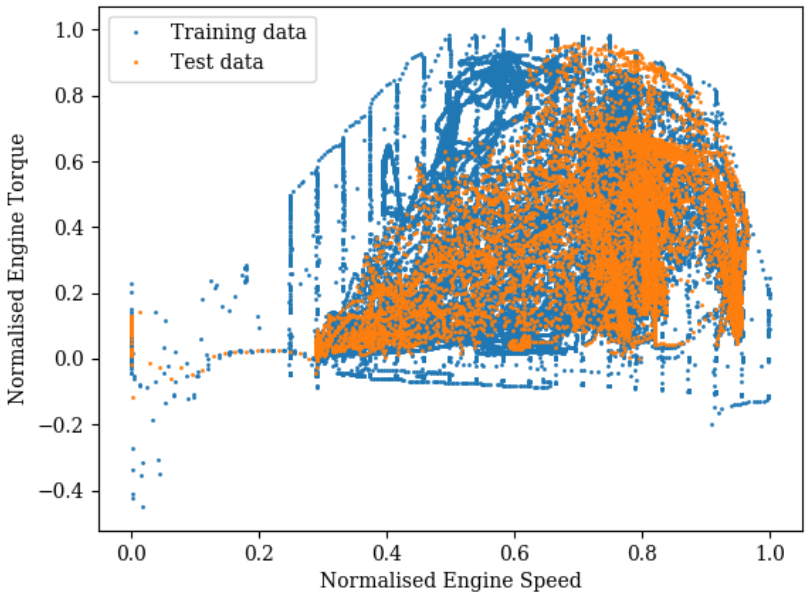


Figure 3.11: The region where the training sequence of the base-set and the NRTC 4 data operates.

Robust models

The Robust models, developed in Chapter 5, will also use the **base-set** for training, validation and test data. The tuning will again be done against the test-set. Additionally, the final performance and result of the Robust models are tested on eight of the unseen NRTC-cycles (denoted as *NRTC 1-8*).

The training and test data which are used for the robust models are operating around the same points as seen in Figure 3.11 above. The difference here compared to the baseline models is that additional samples will be created by adding noise to the original samples, as described in Chapter 5.

Transfer learning models

The Transfer Learning Models, developed in Chapter 6, will, however, be trained and validated on *NRTC 9*, where the split is 80% and 20%. They are tested on the unseen *NRTC 10*-cycle. However, as the retraining is done on new dynamics, there is a possibility of overfitting, making the performance on other cycles worse. Thus the models will be evaluated on the **baseline** test cycle (*NRTC*-4) and compared to the initial result.

Below in Figure 3.12 the operating points for the training and test data of the transfer learning can be seen. Both of the sets are fairly close to each other but are

still two separate test runs.

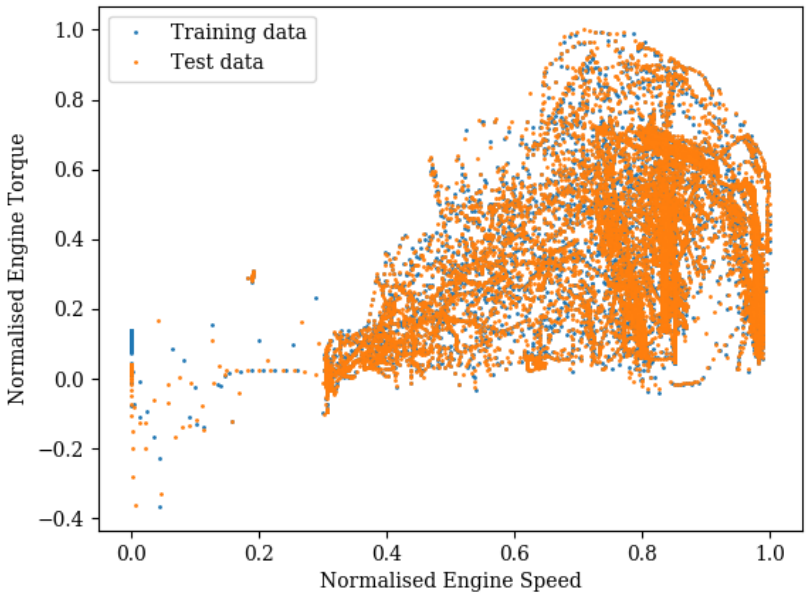


Figure 3.12: The region where the NRTC 9 (training data) and the NRTC 10 (test data) operates.

4
Network structures

Once the pre-processing of the data has been performed properly and reshaped to fit the different network structures, the models can be developed. The first step is to decide upon which loss function and which regression metrics to use, i.e. which error function the network will try to minimise and which metrics to use in order to compare the performance of the networks. Once this is done the base model for each of the structures needs to be decided upon. For the CNN, the web-page of Keras [26] provides multiple examples, of which inspiration is taken. For the LSTM model, a very basic structure of two LSTM-layers and two fully connected layers was chosen as an initial setup. The base models can then be used as a starting point for an iterative process where one hyperparameter is changed while the other remains the same in order to find trends on which parameters that affect the network the most. The different depths and amount of neurons are also explored in this iterative process. The process is performed through two different methods; grid and random search. After these search methods are performed, a selection of the hyperparameters is done based on the obtained results.

4.1 Loss Functions and Regression Metrics

As described in Section 2.1.1 there are different metrics and loss functions that are used for different tasks. For a regression task Mean Squared Error (MSE), as displayed in Eq. (2.5), is commonly used both as loss function and metric. The loss function used for this project, however, was the Root Mean Squared Error (RMSE) and the MSE as a metric. RMSE is defined as the root of the MSE, as in Eq. (4.1).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{4.1}$$

RMSE is used as the loss-function for the reason that the target data is of a high magnitude. This means that using MSE, predictions that be in the right direction might still be punished high, even though the result may be satisfactory. It means the model might overestimate how poorly it performs, especially when the data is noisy. Since part of this thesis is training with a high level of noise, using MSE as a loss function might be problematic. For this reason, RMSE is used as the loss function for the algorithm while MSE is used as a metric for model evaluation.

However, as three different quantities are investigated the MSE provides a metric of different magnitudes, which is both hard to compare and understand. For instance a model with an $MSE_{soot} = 5$ is a considered poor performance, while a

better one for NO_x could be in the region of $MSE_{NO_x} = 1000$. A metric based on the absolute error is introduced as a complement to these deterministic metrics. A metric that could be used is the mean absolute percentage error (MAPE), which is defined in Eq. (4.2).

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{y_i} \tag{4.2}$$

However, there is a disadvantage to using MAPE: it calculates the relative error (in percentage) in every single point and takes the average over the set. If the true signal, y_i, is close to 0, a high contribution is added to the relative error. Thus, this metric can be very sensitive even in cases where the absolute error is small. Instead, a closely related error is introduced: the Relative Percentage Error (RPE). In Eq. (4.3) this new metric is defined.

$$\text{RPE} = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{\sum_{i=1}^{n} y_i} = \frac{e_{sum}}{y_{sum}} \tag{4.3}$$

From Eq. (4.3) it is seen that the sum of error, e_{sum}, is calculated by the absolute error in every observation divided by the sum of the quantity over an entire cycle, y_{sum}. This way, the performance of a model is based on its ability to observe the output over an entire cycle, rather than its ability to observe the output in one instance. In Figure 4.1, the quantities e_{sum} and y_{sum} are shown.

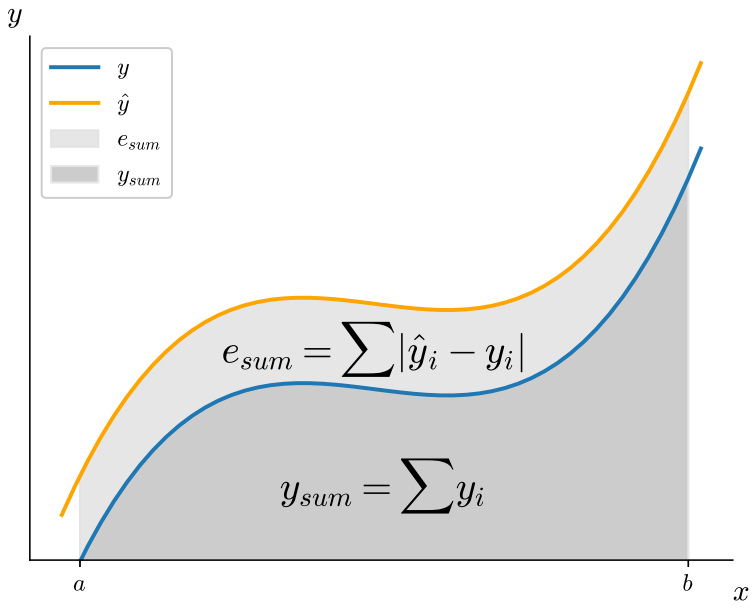


Figure 4.1: The value observed by the network, $\hat{y}$, and the true sequence, y, against time. On the closed set between a and b, the sum of the true sequence is highlighted in dark-grey while the sum of the error (the sum of the difference between $\hat{y}$ and y) is highlighted in bright-grey.

According to [3] it is also interesting for the manufacturer of an engine to be

able to observe the total emission on an entire cycle. To be able to measure this, a metric called *cycle total* is also introduced and defined in Eq. (4.4).

$$cycle\_total = 100(1 - \frac{\sum_{i=0}^{n} \hat{y}_i - \sum_{i=0}^{n} y_{true,i}}{\sum_{i=0}^{n} y_{true,i}}) \tag{4.4}$$

where $\hat{y}$ is the observed signal over the entire cycle and y_{true} is the true sequence of the observed variable.

Another metric used in [3] is the R-squared (or R^2) metric [27]. It is the coefficient of determination and measures the linear relationship between simulated and experimental data [3]. This is defined as,

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{4.5}$$

where $\bar{y}$ is the mean of the observed data y. Observing Eq. (4.5), it is obvious that the maximum value may be 1, meaning that the Neural Network predicts exactly the observed value, y_i, at time i. If R^2 is 0, it means the network predicts the average at every time; negative values mean the model is worse than predicting the average of the cycle. This metric, as well as *cycle total*, is mainly being used to compare the results of this thesis to the article of [3].

4.2 Grid Search

In order to decide upon the structures of the networks a grid search, Section 2.5.1, was first performed over the hyperparameters as well as the depth and amount of neurons in the networks. By doing so trends could be found for the different variables and variables which had a lower impact on the network performance could be identified.

4.2.1 Grid Search: CNN

For the Convolutional Neural network, a natural place to begin the hyperparameter search is at the heart of convolution: the number of convolutional layers. To test this, three models were implemented and trained. The models are identical except for the number of filters in the convolutional layer(s). The models are defined as in Table 4.1. The input sequence selected for these models consists of all 13 signals over 3 seconds (samples with a frequency of 10 Hz), leading to an input of (30×13).

Each model has been trained with one, two and three convolutional layers, with the same specifications as in Table 4.1. In Figure 4.2 these models are plotted with respect to RPE when trained to observe Conc_NOx. The trend (the *Average* of the models) is seen in red.

	Input Layer	Convolutional Layer	Fully Connected Layer	Output Layer
Model 1	(30 × 13)	16 Filters, *relu*	50 Neurons, *relu*	*relu*
Model 2	(30 × 13)	32 Filters, *relu*	50 Neurons, *relu*	*relu*
Model 3	(30 × 13)	64 Filters, *relu*	50 Neurons, *relu*	*relu*

Table 4.1: The three models used for the first grid search. The difference between the models are the number of filters, *receptive fields*, applied in the convolutional layers.

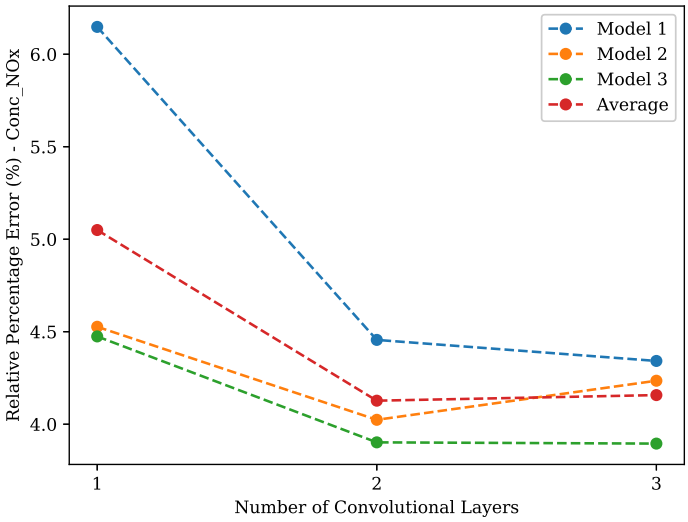


Figure 4.2: How the RPE is affected by the number of (depth) of convolutional layers for three models with varying number of filters in each layer. The average (trend) displays how the RPE is decreased as the depth of the neural network increases.

From Figure 4.2, the most crucial conclusion is to choose at least two-depth convolution. It may also be seen that the number of filters (16 for Model 1, 32 for Model 2 and 64 for Model 3) is important as well. More filters appear to lead to lower (better) RPE, which is further explored by three new models, seen in Table 4.2. The difference in these models is now the number of convolutional layers between the input and fully connected layer.

	Input Layer	Convolutional Layer(s)	Fully Connected Layer	Output Layer
Model 1	(30 × 13)	1 Layer, *relu*	50 Neurons, *relu*	*relu*
Model 2	(30 × 13)	2 Layer, *relu*	50 Neurons, *relu*	*relu*
Model 3	(30 × 13)	3 Layer, *relu*	50 Neurons, *relu*	*relu*

Table 4.2: The three models used for the second grid search. The difference between the models is the depth of the convolutional layers.

The models are trained with four specifications: *16*, *32*, *64* and *128* filters in each layer, with the input sequence being (30 × 13).

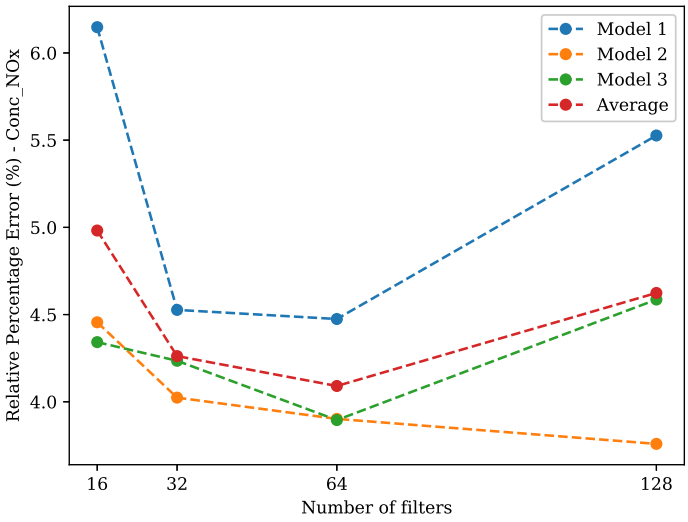


Figure 4.3: How the RPE is affected by the number of filters in each convolutional layer, for three models with different depth in regards to the convolutional layers. The average between the models (the trend) may be seen plotted in red.

As the number of filters in each convolutional layer is increased the RPE decreases to a certain point. It appears beneficial to use at least 64 filters for all models, however, more than that the RPE starts to increase. As more parameters are introduced, the risk of overfitting increases, which may be what is seen in Figure 4.3. However, as the different models display different behaviour with different settings, this is not entirely conclusive.

As a final grid search over the hyperparameters of CNN, the Kernel/Filter Size (or *receptive field*) is iterated. The models defined and trained in this grid search-iteration are presented in Table 4.3. The difference between the models is the same as for testing the number of filters: one, two and three convolutional layers.

	Input Layer	**Convolutional Layer(s)**	**Fully Connected Layer**	**Output Layer**
Model 1	(30 × 13)	1 Layer, *relu*	50 Neurons, *relu*	*relu*
Model 2	(30 × 13)	2 Layer, *relu*	50 Neurons, *relu*	*relu*
Model 3	(30 × 13)	3 Layer, *relu*	50 Neurons, *relu*	*relu*

Table 4.3: The three models used for the second grid search. The difference between the models is the depth of the convolutional layers.

In Figure 4.4, the results of these models may be seen. In difference to the previous grid searches, the results are less linear: the trend RPE is linearly decreasing as the kernel size increases, however, for a single model, this is not as obvious. The RPE is going both up and down for all models, except for the model with one layer

convolution which is worse for all configurations. Depending on the model, different Kernel Sizes should be applied accordingly for the best result.

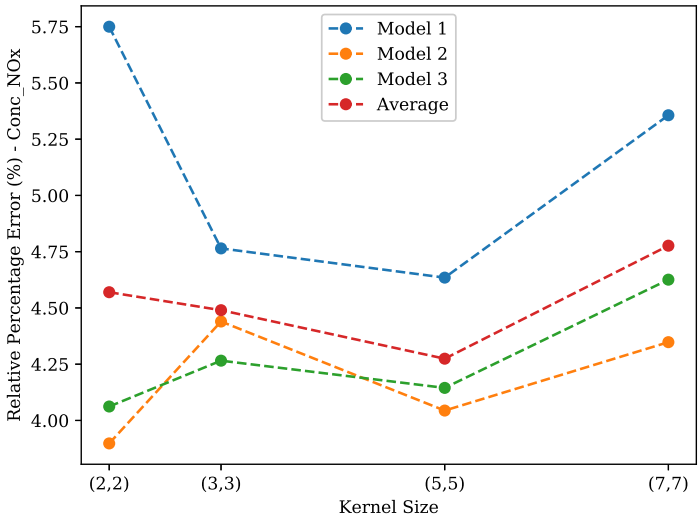


Figure 4.4: How the RPE is affected by the kernel size of the filters in each model. Seen is Convolutional models with 1-, 2- and 3-layer depth. Also plotted is the average (trend), in red.

The grid search for the Convolutional models shows that the number of convolutional layers is important: increased depth decreases the RPE. The number of filters shows a similar trend as a better (lower) RPE is found when the number of filters is increased in the convolutional layers. As for the kernel size, overall an increased *perceptive field* decreases the RPE, but this needs more attention. More important, a slightly more complicated model tends to provide a lower RPE, which may be seen in Figures 4.2, 4.3, 4.4, where "*model 1*" is performing worse than the other models.

When performing the grid search, the number of filters, kernel size as well as the number of neurons is kept constant through the models. Different configurations, such as an increased number of filters in each layer as the network becomes deeper (counting convolutional layers), may provide lower RPE for the test-set. This is found by performing a less strict and time-consuming search: the random search.

4.2.2 Grid Search: LSTM

For the LSTM models, all of the grid searches were performed while evaluating the RPE for the Conc_NOx data. The first grid search was performed over the amount of fully connected layers as well as the number of neurons in each layer. A starting model with 2 LSTM layers which contained 25 cells in each layer was used as seen in Table 4.4. The input data consists of samples of size (30×13) as for the CNN models since the same amount of time steps and signals are being used.

	Input Layer	LSTM Layers	Fully Connected Layer(s)	Output Layer
Model 1	(30 × 13)	2 Layers, 25 Cells, *elu*	1 Layer, *elu*	*relu*
Model 2	(30 × 13)	2 Layers, 25 Cells, *elu*	2 Layers, *elu*	*relu*
Model 3	(30 × 13)	2 Layers, 25 Cells, *elu*	3 Layers, *elu*	*relu*

Table 4.4: The three models used for the grid search over the fully connected neurons. The difference between the models is the depth of the fully connected layers.

Further, the result of the first grid search can be seen in Figure 4.5. There, each of the models which are specified in Table 4.4 is evaluated against the RPE. Additionally, the average of the models is plotted in red in order to show the overall trend.

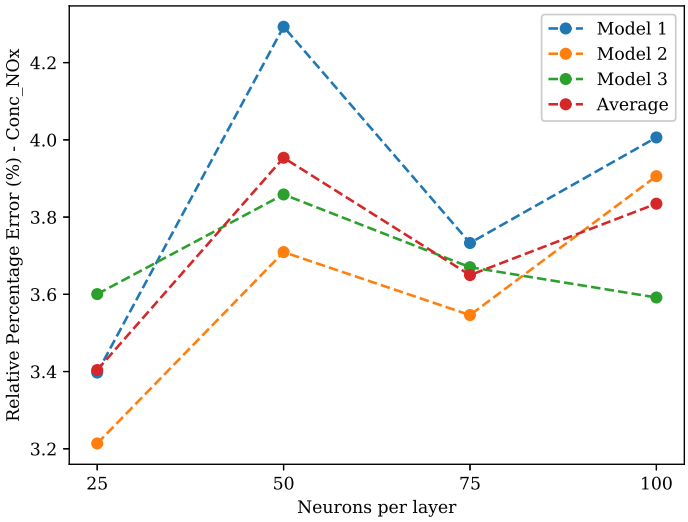


Figure 4.5: How the RPE is affected by the number of neurons in each fully connected layer, for three models with varying amount of fully connected layers. The average between the models (the trend) may be seen plotted in red.

The grid search over the number of neurons, Figure 4.5, showed that 25 neurons in each fully connected layer gave the best performance. Additionally, model 2 showed a strong performance overall and therefore a total number of 2 fully connected layers were decided upon.

The second grid search of the LSTM models was performed over the number of cells in the LSTM layers while also sweeping over the amount of LSTM layers. The models which are shown in Table 4.5 consisted of 1-4 LSTM layers along with 2 fully connected layers with 25 neurons in each.

The performance of the models described in Table 4.5 are shown in Figure 4.6. Each model, as well as the trend (the average of the models), is plotted against RPE.

	Input Layer	LSTM Layer(s)	Fully Connected Layers	Output Layer
Model 1	(30 × 13)	1 Layer, *elu*	2 Layers, 25 Neurons, *elu*	*relu*
Model 2	(30 × 13)	2 Layer, *elu*	2 Layers, 25 Neurons, *elu*	*relu*
Model 3	(30 × 13)	3 Layer, *elu*	2 Layers, 25 Neurons, *elu*	*relu*
Model 4	(30 × 13)	4 Layer, *elu*	2 Layers, 25 Neurons, *elu*	*relu*

Table 4.5: The four models used for the grid search over LSTM Cells. The difference between the models is the depth of the LSTM layers.

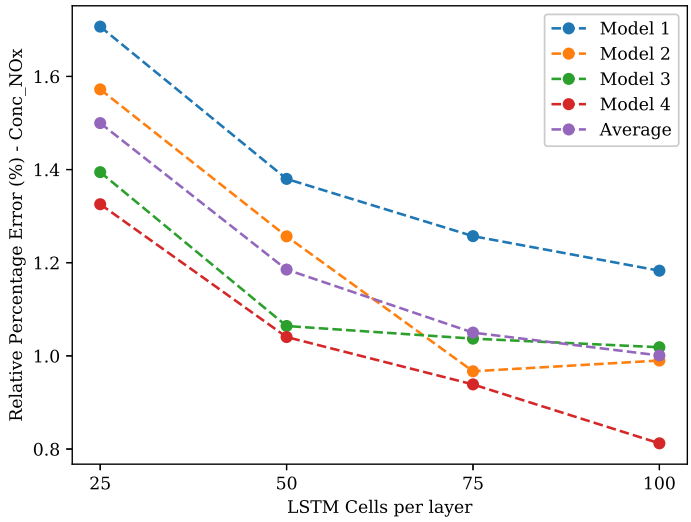


Figure 4.6: How the RPE is affected by the number of cells in each LSTM layer, for four models with varying amount of LSTM layers. The average between the models (the trend) may be seen plotted in purple.

From Figure 4.6 it can be seen that as the complexity of the LSTM layers (number of cells and layers) increases, the RPE decreases. An even lower RPE might be possible to achieve through increasing the number of cells and LSTM layers further. However, due to the fact that the computational complexity increases with a more complex model, a structure with 4 LSTM layers and 100 cells in each layer was decided upon.

Through early on trials it was noted that batch normalisation on the LSTM layers showed a much better performance in RPE as well as a higher consistency during the training process, i.e. before the batch normalisation was added a lot of the models tended to diverge during the training process. Hence, a grid search evaluated against the RPE performance was done while iterating over the amount of batch normalised layers. In Table 4.6 three models are specified which are identical except for the different learning rates which will be used during their training process.

The three different models which are displayed in Table 4.6 are further evaluated in Figure 4.7. The average of all the models is plotted in order to show the overall trend.

	Input Layer	LSTM Layers	Fully Connected Layers	Output Layer	Learning Rate
Model 1	(30 × 13)	4 Layers, *elu*	2 Layers, *elu*	*relu*	0.001
Model 2	(30 × 13)	4 Layers, *elu*	2 Layers, *elu*	*relu*	0.005
Model 3	(30 × 13)	4 Layers, *elu*	2 Layers, *elu*	*relu*	0.01

Table 4.6: The three models used for the grid search over the amount of batch normalised layers. The difference between the models are the different learning rates.

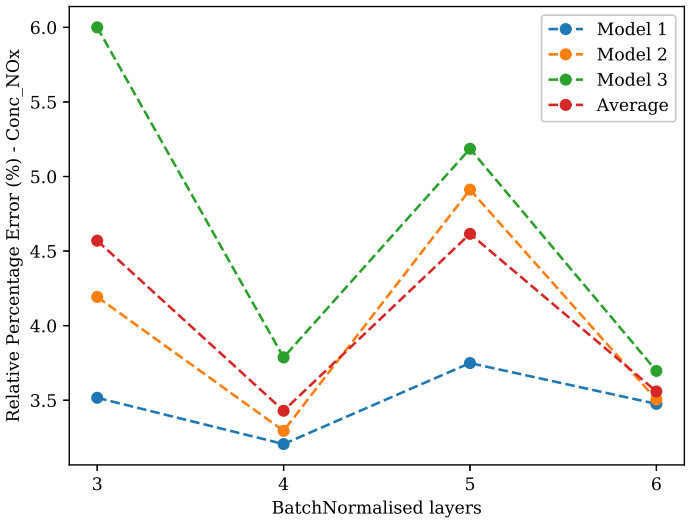


Figure 4.7: How the RPE is affected by the amount of Batch Normalised layers, for three models with varying learning rate. The average between the models (the trend) may be seen plotted in red.

The grid search performed over the amount of batch normalised models which is displayed in Figure 4.7 showed that batch normalisation on the four first layers gave an overall best performance for the LSTM models. This means that batch normalisation will be added to the four first layers which are the LSTM layers and that the fully connected layers will not contain any batch normalisation. Another interesting point which the grid search in Figure 4.7 shows is that with a lower learning rate a lower RPE value can be achieved. This follows the theory explained in Section 2.1.10 very well. Where a lower learning rate gets us closer to the local or global minimum. Additionally, the training times for the models which used a higher learning rate was much lower than the models trained with a lower learning rate which also follows the theory explained in Section 2.1.10. Essentially it comes down to a balancing of performance versus training time where a decision was made to continue with a learning rate of 0.001 which showed the best performance in RPE.

4.3 Random Search

Found in Section 4.2, the Convolutional-type networks appear to provide the best results when the number of convolutional layers is more rather than less. For this reason, the Networks to be continued are either three or four convolutional layers deep. The number of filters has previously been kept constant, with 64 filters in each layer providing the lowest RPE, from Eq. (4.3). Ambiguity exists in the size of the Kernels: no linearity between RPE and the size of these is seen. For these reasons, the kernel sizes and number of filters in each layer will be semi-randomly searched.

Since the input to the network is quite small, between one and six seconds times the number of input signals: (10×13)-(60×13), an increased Kernel Size quickly decreases the passed on feature maps. The Kernels are therefore randomised differently in different layers: the first layer has the most degree of freedom and the kernel is drawn randomly in the range of $((1:30) \times (1:13))$, while the following layers may be drawn with respect to the output of the first one. The number of filters is randomly selected in between $1-150$ for each layer. Also, the number of fully connected layers following the Convolutional layers is randomly selected between $1-4$.

In Figure 4.8, 89 models are evaluated and plotted against the test data-set. Since there is an overflow of models, only the significant ones will be presented.

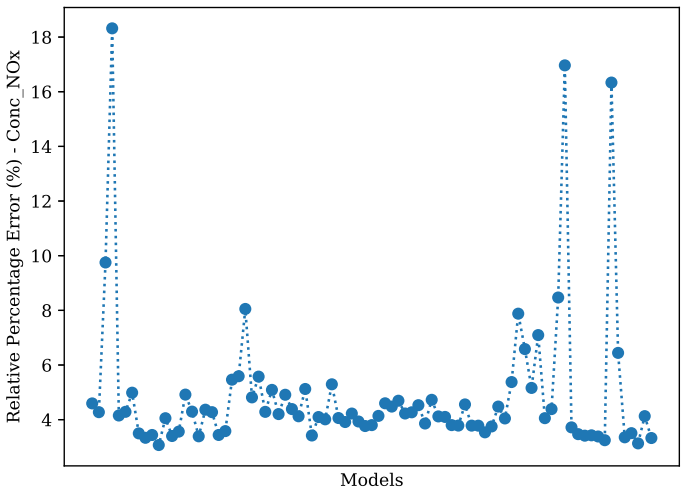


Figure 4.8: 89 models, generates by randomly selected values for the hyperparameters Kernel Size, Number of Convolutional Filters and number of fully connected layers, evaluated against the test-set.

As may be seen, the vast majority of the models provide a similar RPE-value, meaning that most values for the hyperparameters provide a reasonable result. The ones that stand out, i.e. the ones with an RPE over 5, however, have in common that the number of filters either is quite small (around 10-20 in most layers) or with

large kernel sizes (over (15×7)). The models with the lowest RPE also share the number of fully connected layers, that is 3, which appears to provide the lowest RPE.

4.4 Selection of Hyperparameters and functions

In this Section, the final selection of hyperparameters will be explained and motivated, with the background of the findings in Sections 4.2-4.3. In some cases a more complex model is preferred in terms of performance against RPE 4.3, however, the complexity of training such model might not be beneficial for the purposes of this project and so this will be motivated in the following Sections.

4.4.1 Convolutional hyperparameters

As the first layer of a convolutional network is used to pick out basic features, such as edges and other generic shapes [10], the need for a massive amount of filters in redundant at the beginning of the model. Instead, the model could be built like an inverted pyramid: fewer filters in the more shallow layers of the network, working up to a higher amount in the deepest of convolutional layers. Since it was found in Sections 4.2-4.3, that a number of filters between $64 - 128$ is providing the best result, these are set in the final layers.

The Kernel Size has been iterated through both by grid and random search, and it has been shown that a perceptive field of (3×3) as well as sizes such as (15×7) for the first layer provide similar results, with the larger one containing about three times the amount of parameters. This increases the risk of overfitting while it takes a significant amount of time to train. For this reason, the Kernel Size is selected to (3×3), throughout the Convolutional Layers.

The number of Fully connected Layers is selected in accordance with the findings of Section 4.3, where 3 fully connected layers gave the lowest values of RPE.

4.4.2 General Parameters and functions

Different activation-functions were used and tested in the different models. Mainly the Rectified Linear Unit (ReLU) and exponential linear unit (elu) were explored. In all tests $ReLU$ was outperformed, or performed similarly, to *elu*, for which reason the latter is used for all layers except the last. The last layer uses $ReLU$ since neither emissions nor fuel flow can be negative from a physical point of view.

No extensive search over the multiple optimisers was done, however, according to [1] reasonable optimiser functions are the SGD and Adam. Both of these along with RMSprop and Nadam were tested and evaluated. Out of the four optimisers *Adam* provided slightly better results than Nadam and was therefore used for the training of the final models.

As mentioned in Section 2.1.8, regularisation methods can, and should, be applied to the networks. One of these is Dropout which should improve the network's ability to generalise. However, all models (even when using a small dropout-rate) diverged when this was applied and Dropout was deemed to be unsuitable for this

project. Instead, Batch Normalisation was used. As according to Section 2.1.9, using Batch Normalisation improves stability and might also reduce training time. In Section 4.2.2 it was mentioned that this was found to be correct which can be seen in Table 4.7, where three identically models were trained and evaluated with and without Batch Normalisation. In all three cases, it proved beneficial to use batch normalisation, while the number of epochs it took to train these were lower than without.

	RPE (%) Without Batch Normalisation	**RPE (%) With Batch Normalisation**
Model 1	4.595	3.337
Model 2	5.813	4.866
Model 3	5.822	4.645

Table 4.7: Three models trained with and without batch normalisation layers between all other layers. With Batch Normalisation, the models converged faster and provided a lower RPE.

As batch normalisation improves the overall stability of training the networks, a higher learning rate may be used, as such is the case for the Convolutional Network where a learning rate of 0.05 was used, rather than the by-default recommended 0.001.

Another regularisation technique that was implemented was the *early stopping*-algorithm which is described in Section 2.1.8. The algorithm is designed to prevent overfitting, which it successfully has done.

4.5 Final Network Structures

The final structure which will be used as the baseline model for LSTM consisted of 4 LSTM layers with together with 2 fully connected layers. All layers used *elu* as activation function except for the output-layer which was implemented with a *ReLU* as the activation function. Furthermore, batch normalisation was added after each of the LSTM layers but not after the fully connected layers since no real improvement in performance was seen through adding batch normalisation after the fully connected layers. The number of cells in each LSTM layer was set to 100 and the number of neurons in each fully connected layer was set to 25. The Learning rate for the LSTM model was set to 0.001 and the most suitable optimiser was found to be Adam. The final structure is shown in Figure 4.9.

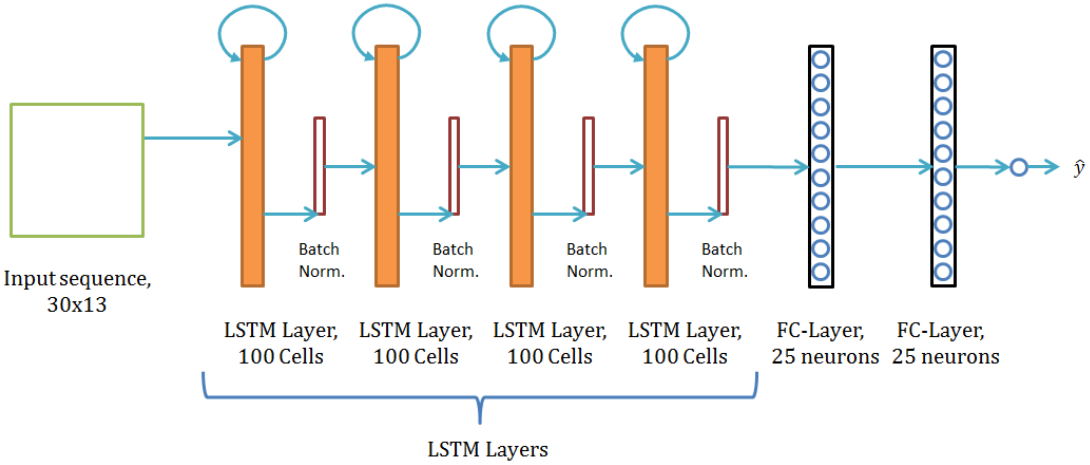


Figure 4.9: A Figure over the structure of the baseline model for LSTM.

For the Convolutional Neural Network, the structure to be used as the baseline model is presented in Figure 4.10. The model consists of three convolutional layers and two fully connected ones before the output layer of the network. The convolutional layers are structured as a pyramid with (3×3)-filters; the first layer has 32 filters, the second 64 and the final one 128. The fully connected layers contain 50 neurons which are activated by *elu*. The same activation function is also used for the convolutional layers, however the output layer is activated by $ReLU$, as per explained in Section 4.4.2. In between all layers and their respective activation function batch normalisation is added.

For the Convolutional Neural Network, the learning rate used for the optimisation function, Adam, was 0.05 (to make use of the benefits of batch normalisation, see Section 2.1.9).

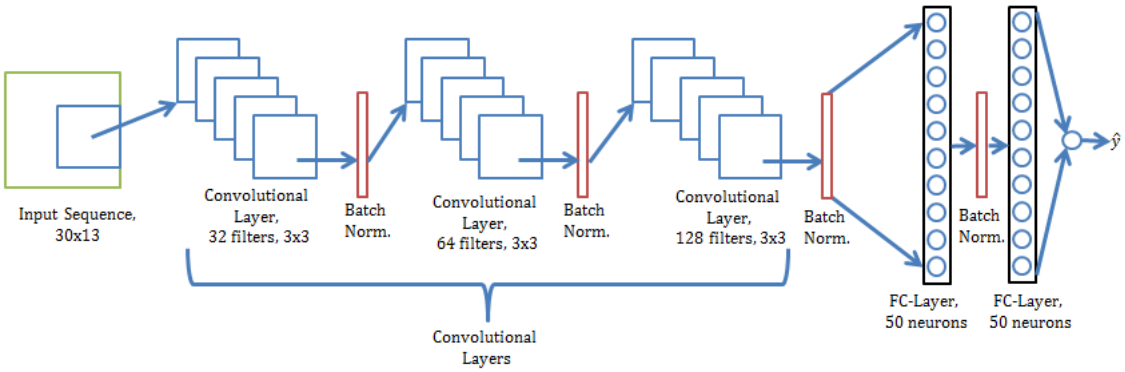


Figure 4.10: A Figure over the structure of the baseline model for CNN.

The Network structures presented in the text above and Figures 4.9-4.10 are used for all quantities to estimate (Flw_FuelDiesel, Conc_NOx and Conc_Soot). The difference for these quantities is the input sequence: for Flw_FuelDiesel an input sequence of size (10×13), i.e. considering a time-dependency of one second

sampled with a frequency of 10, is used. For Conc_NOx an input sequence of size (30×13) and for Conc_Soot (50×13) are used.

5
Robustness

A model which is trained on data from a specific engine with a certain calibration can become very accurate in predicting the output of a sensor. However, the high accuracy of the model will be very limited to the specific engine and calibration. Furthermore, the model may be very sensitive to input noise (such as a low-accuracy sensor) and requires similar environmental conditions as when the training data was obtained. In order to create a more robust and general model, a known method is to augment the data by adding noise to it [28]. This method has shown promise in other areas which handle time-dependent systems and is, therefore, a very viable method to investigate in this scenario.

The data will be augmented with noise through two different methods which will be explained more in depth in the upcoming Section. For each method, the amount and size of the noise will then be iterated through in a similar fashion as with the hyperparameters in Section 4.2. In order to simplify this process, the only sensor which will be used in the iterations is the NO_x-sensor. The optimal amount and size of the noise that is found is then assumed to be representative for all three sensors.

The models will be trained on the initial data but evaluated on eight previously unseen NRTC-cycles which shows a different behaviour compared to the base-set.

5.1 Introducing noise

In order to find a suitable size on the noise each of the signals were examined independently. From consultation with the engineers at Volvo Penta, it was clear that specifications on exactly which sensor that has been used for each of the signals cannot be given. Thus, the size of the noise is roughly estimated percentages given from the engineers at Volvo Penta or found online for sensors which normally are used for measuring the specific signal. All the percentages are then gathered in a matrix, P, which has the error percentage of each signal along the diagonal:

$$P = \begin{bmatrix} p_{1,1} & 0 & \cdots & 0 \\ 0 & p_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & p_{13,13} \end{bmatrix}$$

The reason for creating a diagonal matrix is to simplify the multiplication with the input samples in order to create the deviation of the noise for each sample. As described in Section 3.6 the input data samples come in the shape of a matrix. This

matrix is $(n \times 13)$ in size where 13 comes from the number of signals and n is the number of time steps which the network will consider. For the robustness part, the time steps will always be set to 30 and therefore the input matrix will constantly be a (30×13) sized matrix.

The noise is introduced as,

$$x_i' = x_i + \alpha W \tag{5.1}$$

where α is a factor to scale the noise and W is the noise. The added noise, W, is drawn from a uniform distribution such as,

$$W \sim U(-D, D) \tag{5.2}$$

where D is a vector consisting of the amount of maximum noise possible for each of the 13 input signals at each time-step. This deviation, D, is calculated at every time-step as,

$$D = x_i P \tag{5.3}$$

The noise is therefore created individually at every time-step by each of the signals potential deviation. The target value of each noisy input, x_i', is

$$y_i' = y_i \tag{5.4}$$

The noise augmentation is done for both the **training** and **validation** set.

The number of samples to augment with noise is selected by the parameter β. A list of indexes, I_β, is uniformly drawn, β-long, from the data-set (with m-examples) and augmented according to Eq. (5.1). The full algorithm for augmenting data is stated as:

Algorithm 1: How the iterative augmentation of data is done, based on the scaling α and number of examples to augment, β.

Data: $x_{training}$, $x_{validation}$

set α;
set β;
$I_\beta \sim U(0, m)$;
for i in I_β **do**
    $D = x_i P$;
    $W \sim U(-D, D)$;
    $x_i' = x_i + \alpha W$;
    $y_i' = y_i$;
end
$x_{training} = [x, x']$;
$y_{training} = [y, y']$;

As seen in Algorithm 1, new examples are added, not substituting the original example, meaning the number of training data is increased by the factor β.

5.2 Grid search with noise

In order to evaluate the effectiveness of adding noise through the method described in Section 5.1, an iterative process is started, much like the one in Section 4.2.

Through training the models over a grid consisting of different α- and β-values the effect of the magnitude and amount of noisy samples can be studied. The results of this iterative process are evaluated against NRTC 8, which is a cycle where the engine showed fairly similar behaviour to the training data but was still deviating a bit. The results of this grid search can be seen in Figure 5.1.

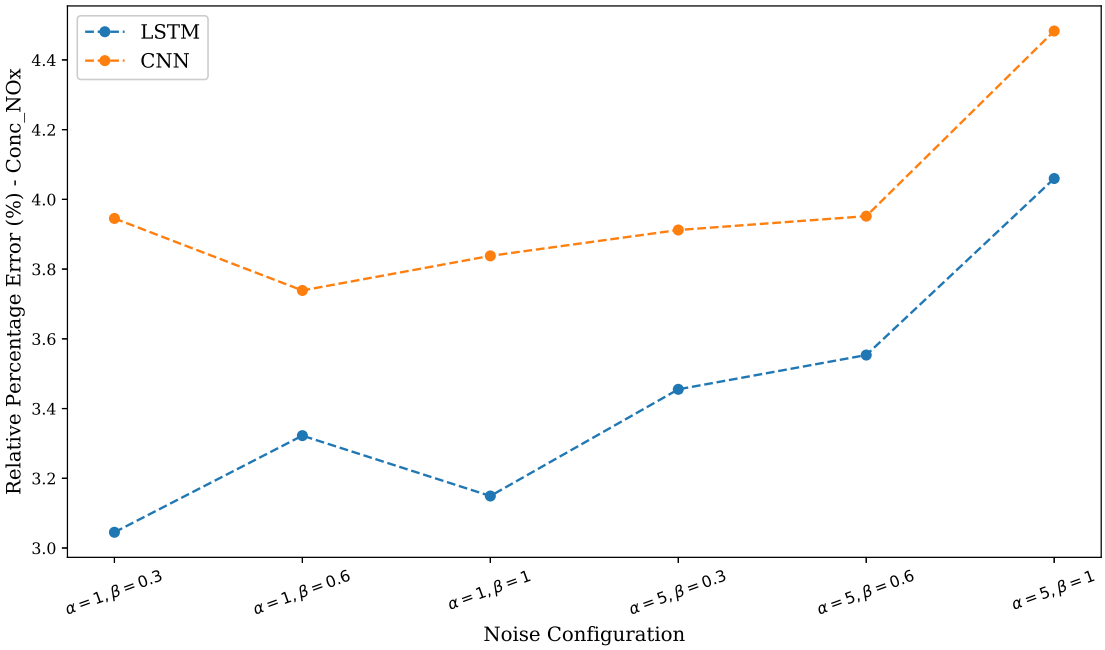


Figure 5.1: A grid search performed where different setup of α and β was iterated over and evaluated against RPE.

From the graph shown in Figure 5.1 it could be seen that both CNN and LSTM showed a top performance for robustness training at $\alpha = 1$. Additionally, an overall lowest value was found at $\beta = 0.3$ and $\alpha = 1$ for the LSTM but for the CNN a setup with $\beta = 0.6$ and $\alpha = 1$ showed the best performance. Hence, the LSTM will be evaluated with the first setup while the CNN will be evaluated with the second one.

6
Transfer Learning

Much like the methodology of Chapter 5, which is designed to improve the performance of a model on a data-set far from the training- and validation-data, transfer learning [10, 11] is another approach to this type of problem. However, unlike training with noise, transfer learning requires additional data which is from the new task. A robust model (such as the ones described in Chapter 5) may perform better on unseen data than a model that has been trained on a specific data-set; a model fine-tuned on a data-set of a different engine calibration should cope with the changing dynamics even better. The fully trained models of Chapter 4 are reused as the baseline model trained with additional data for this Section.

The models will be trained on the new data, by different application: retraining only the last fully connected output layer, fine-tuning all fully connected layers and finally use the previous model's weights as initialising points such as described in Section 2.1.4.

The models developed in this Section will be trained on a NRTC-cycle which has not previously been seen containing 10158 training examples and 2540 validation examples. The models will be tested and evaluated on yet another unseen NRTC-cycle with 12697 examples. Both cycles with different calibrations than the initial data-set.

6.1 Grid search with transfer learning

Such as in [11], the layers to retrain will be iterated over for the CNN and LSTM models. The layers that will not be trained may be seen as "frozen" i.e. layers which parameters, weights and biases, are locked into the state which acquired by previous training. Training only the output layer should then use all feature extraction acquired from previous training in all layers except the last. On the contrary, retraining all layers could be viewed as training a completely new model, but with a previous model's weights as initial weights, potentially leading to faster convergence.

As it is desired to not train the model too much (and achieve overfitting for the new task), the number of epochs for each model to be trained as well as the learning rates will be limited. In [11], the same type of iteration is done, with substantially more data. Using the same ratio between training epochs and amount of data, the number of epochs to retrain the models of this project is set to 12 and the learning rate to 0.01, as in [11].

The baseline CNN model has three convolutional layers and two fully connected

ones (see Section 4.5, Figures 4.9-4.10) while the LSTM consists of four LSTM-layers and two fully connected before the output layers. These models will be retrained, fine-tuned, iterating over the layers backwards: first retraining only the output layer, then the fully connected ones and finally the base layers (or feature extraction layers for the CNN [10, 11]). This may be seen in Figure 6.1, where three different strategies are shown.

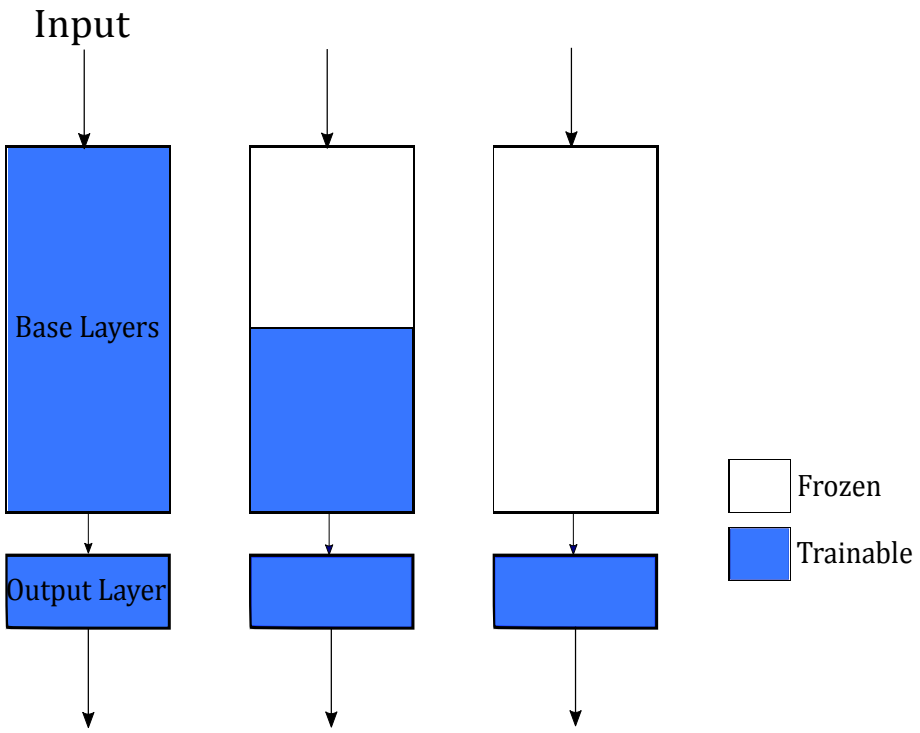


Figure 6.1: Strategies of using transfer learning by freezing different amount of layers. In the left staple all layers are trainable, while the middle stable only has some of the base layers (Convolutional or LSTM) trainable and to the right is an example of when only the output layers and the fully connected layers are trainable, keeping the base layers untouched.

The grid search is performed by retraining the models' layers, from the output to the input. In the case of the Convolutional Neural Network there are six layers, or configurations, to be tried: the weights to the output layer, two fully connected layers and three convolutional layers. For the LSTM network, there are seven: the output layer, two fully connected layers and four LSTM layers. In Figure 6.2 the retrained Convolutional Neural Network models' results are plotted as RPE for *NRTC-10* on the y-axis, against the number of trainable layers on the x-axis.

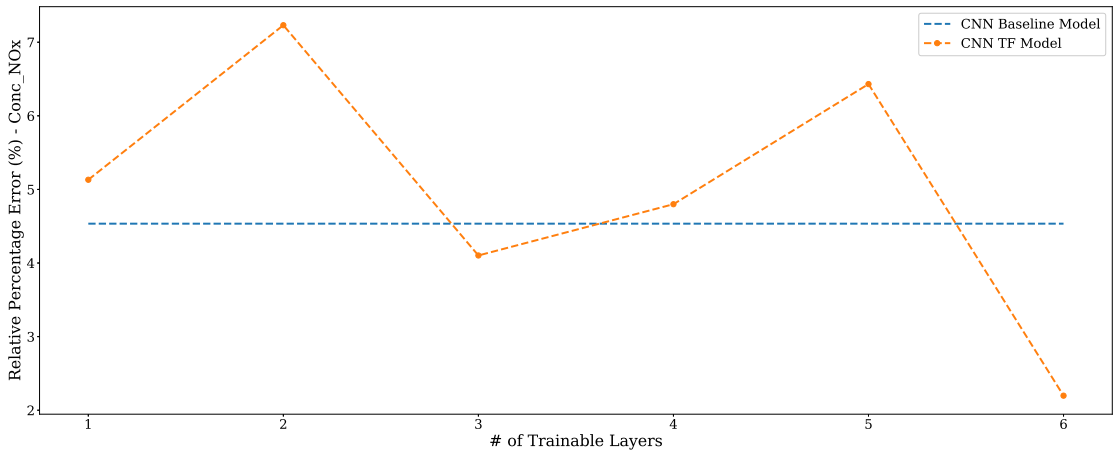


Figure 6.2: The performance of the CNN model's configurations on the test cycle (*NRTC-10*), presented in RPE on the y-axis against the number of trainable layers on the x-axis.

In Figure 6.2, it may be seen that for all configurations except two the baseline model performs better on the *NRTC-10* cycle. The exception is at three and six trainable layers. Three layers is when only the convolutional layers are frozen, but all fully connected as well as the output layers are retrainable. Six layers is when all layers of the model are retrainable, thus every weight and bias is changed.

In Figure 6.3, the retrained LSTM models' results are plotted as RPE for *NRTC-10* on the y-axis, against the number of trainable layers on the x-axis.

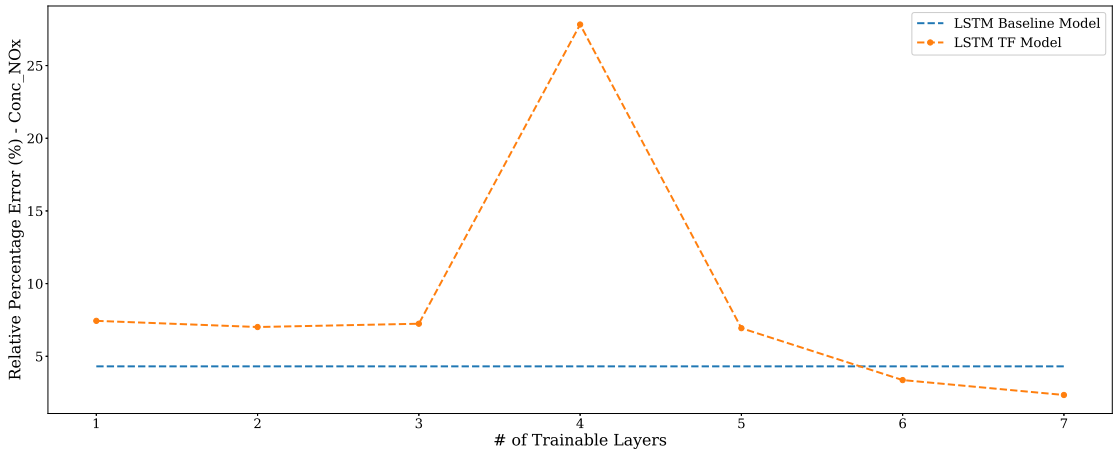


Figure 6.3: The performance of the LSTM model's configurations on the test cycle (*NRTC-10*), presented in RPE on the y-axis against the number of trainable layers on the x-axis.

In the Figure, it may be seen that for all configurations except two the baseline model performs better on the *NRTC-10* cycle. Unlike the case of the Convolutional models, the two models which perform best are those with no frozen layers or with only the first LSTM layer being frozen. All other configurations provide worse results than the baseline model.

In both the case of the Convolutional Neural Network and the LSTM network, retraining all layers according to the configurations aforementioned yield the best result and are therefore selected from the grid search of the transfer learning.

7
Results

The results have been divided into three different Sections which aims to follow up and evaluate the models and configurations which were developed in Chapter 4-6. Where Chapter 4 will be evaluated against NRTC 4, Chapter 5 will be evaluated against NRTC 1-8 and Chapter 6 which is trained on NRTC 9 will be evaluated on NRTC 10. The models will be evaluated based on RPE, Cycle total and R^2-value. Additionally, the latency of the models and training time will be taken into consideration.

7.1 Baseline model performance

In Chapter 4 structures for the baseline models were produced and then presented in Section 4.5. These produced models will be evaluated on the metrics described in Section 4.1 and tested on one of the *NRTC* cycles. This cycle (NRTC 4) is of an engine with a calibration similar to the one used for the training data-sets, and consists of 14 000 observations. The tests will be performed and evaluated for all six models; CNN and LSTM for all three output signals.

The evaluation is done by propagating the input-sequences of the data, split according to Section 3.6, through the network, obtaining predictions made by the models at every time-step, i. Once the entire cycle is obtained, the performance can be evaluated on the metrics RPE, cycle total and R^2; Equations 4.3, 4.4 and 4.5.

Concentration of NO_x

In Table 7.1, the models evaluated on the metrics may be seen for the quantity Concentration of NO_x.

	Metrics		
	RPE	*Cycle Total*	R^2
CNN	3.41	98.64	0.993
LSTM	2.80	99.00	0.995

Table 7.1: The performance of the two model types (CNN and LSTM) on the measured quantity Concentration of NO_x. Presented are their respective performance on the three metrics presented in Section 4.1.

It may be seen that the LSTM model is outperforming the CNN model. In fact, it provides better results in all three metrics, ever so slightly. The Relative

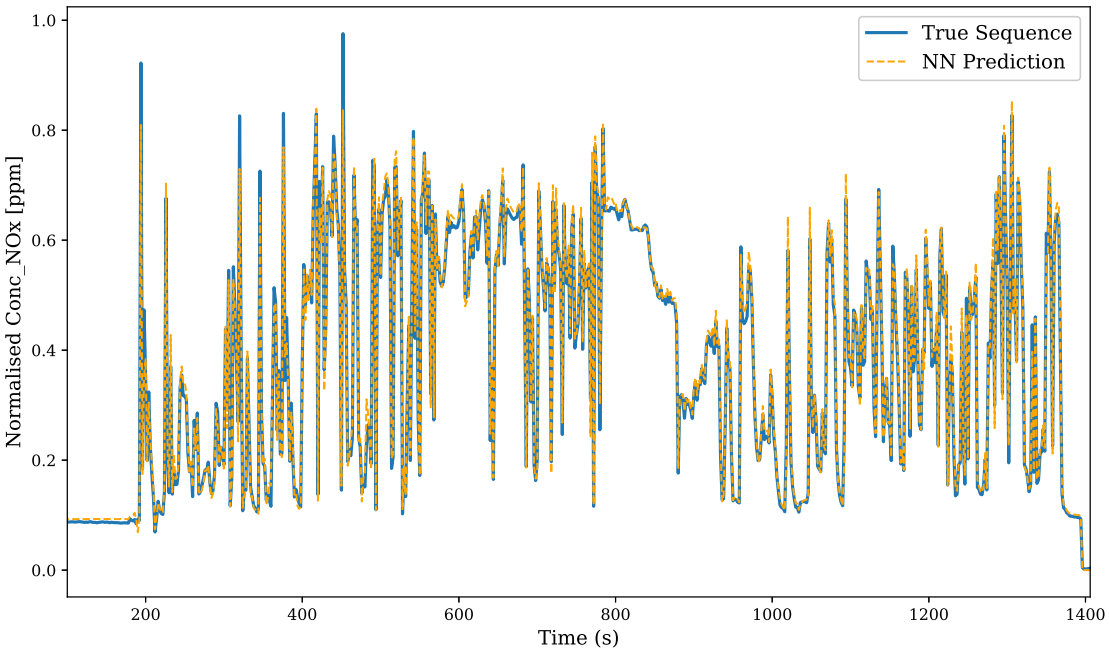


Figure 7.1: A prediction made by the CNN baseline model for Conc_NOx against the true sequence of NRTC cycle 4.

Percentage Error (RPE), as described in Eq. (4.3) in Section 4.1, is the main metric; it is used when the models are trained and evaluated. It may be seen from Table 7.1 that the magnitude of the RPE is in the same order (slightly higher for the LSTM network) as the RPE of the training (seen in Figure 4.7), but now for data from an entire and unseen cycle. The values of the other metrics, *cycle total* and R^2, are satisfactory since the respective maximum values are 100 % and 1. This means that, in terms of *cycle total*, the models capture $98,64\%$ and $99,05\%$ of the entire emission over a cycle of 14 000 samples (1 400 seconds).

In Figures 7.1-7.2 we depict the predictions made by the networks against the true sequence for Concentration of NO_x of the Convolutional Neural Network and LSTM-networks respectively. As the metrics in Table 7.1 suggests, both models are able to capture the overall behaviour of the NO_x; they follow the trend while capturing most of the spikes in the cycle.

However, it may be seen that both models have some difficulty in predicting the highest values in the region $300 - 500$. The difficulties of capturing the spikes with the highest magnitude may be seen in Figures 7.3-7.4 respectively for the CNN and LSTM. These spikes, being of high magnitude, contribute substantially to the error in the metrics.

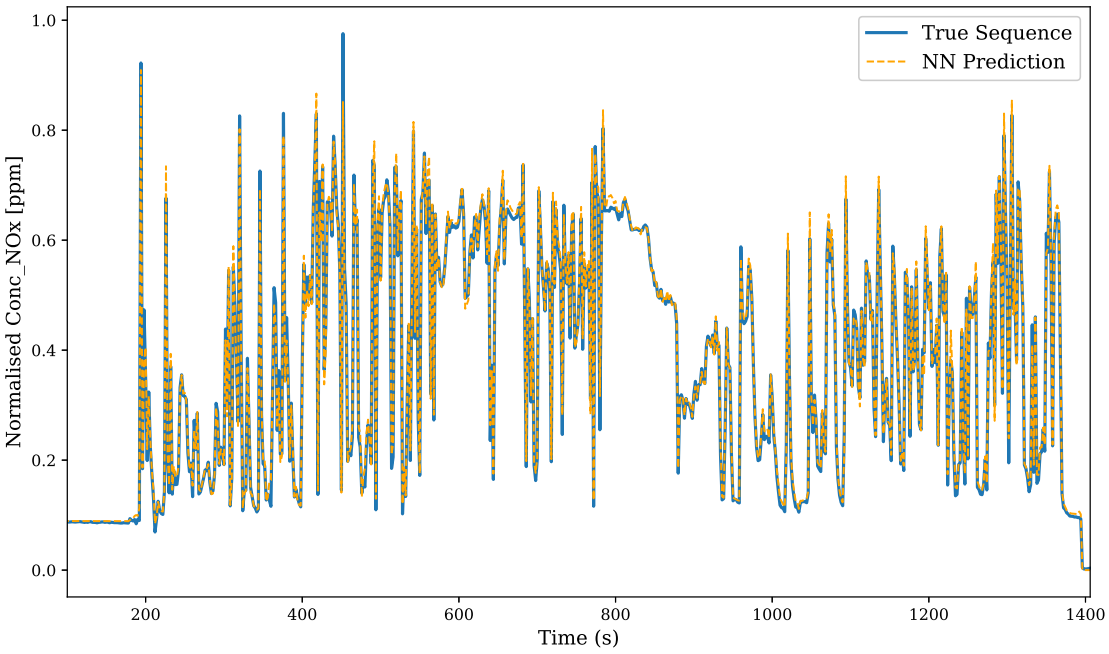


Figure 7.2: A prediction made by the LSTM baseline model for Conc_NOx against the true sequence of NRTC cycle 4.

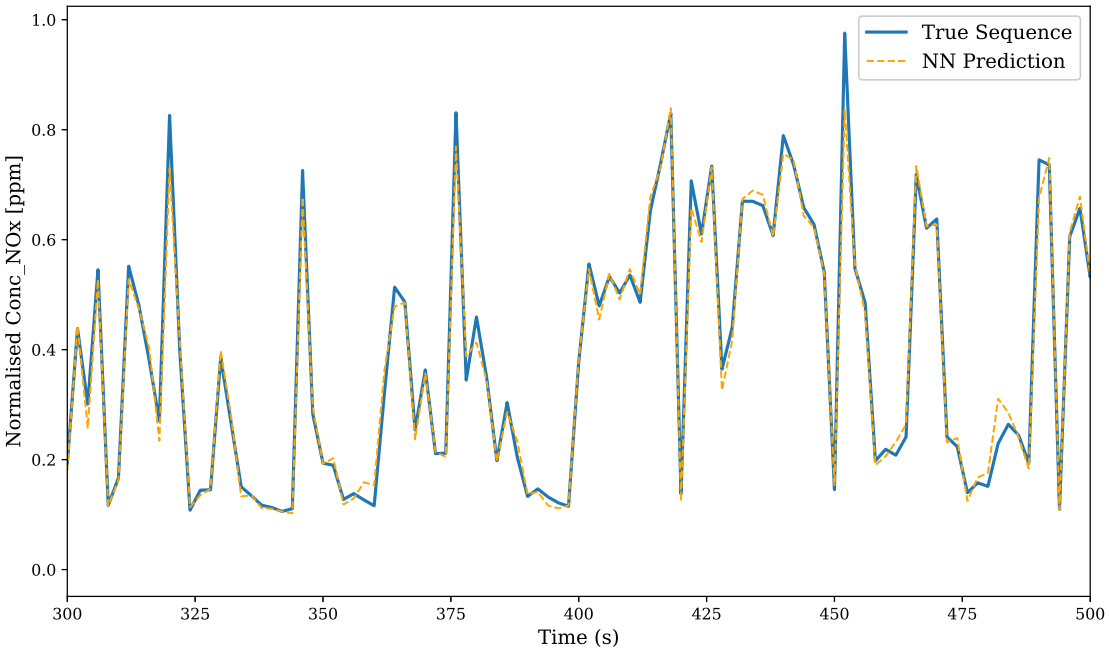


Figure 7.3: A prediction made by the CNN baseline model for Conc_NOx against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the difficulties of capturing the highest spikes.

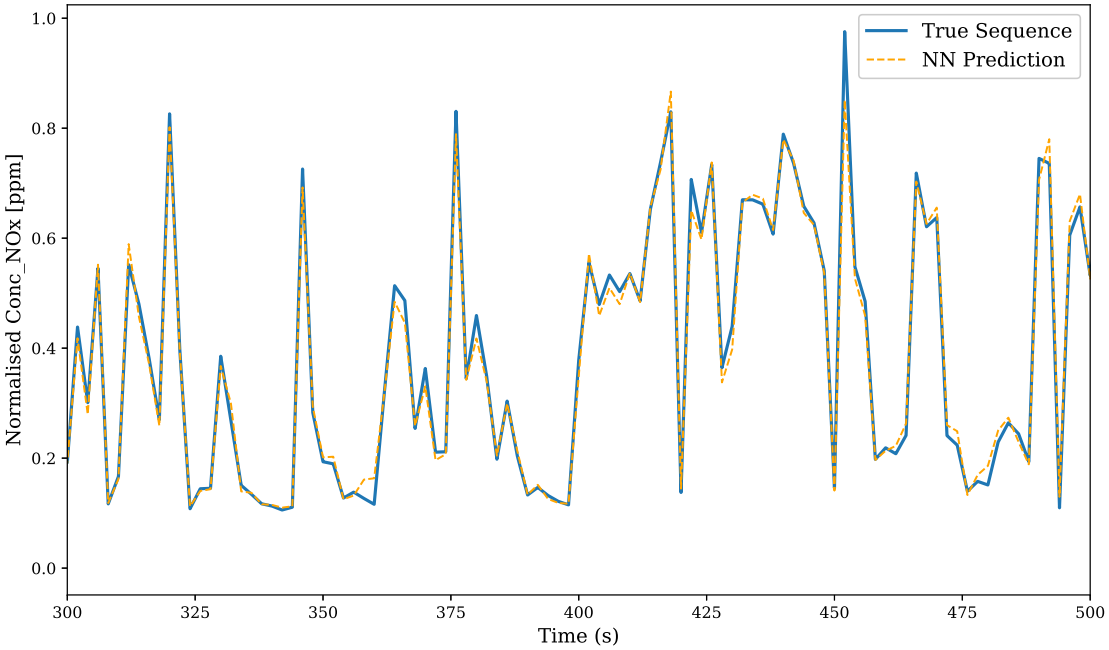


Figure 7.4: A prediction made by the LSTM baseline model for Conc_NOx against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the difficulties of capturing the highest spikes.

Flow Fuel Diesel

In Table 7.2, the models evaluated on the metrics may be seen for the quantity Flow Fuel Diesel.

	Metrics		
	RPE	*Cycle Total*	R^2
CNN	2.44	99.74	0.998
LSTM	2.40	99.83	0.997

Table 7.2: The performance of the two model types (CNN and LSTM) on the measured quantity Flow Fuel Diesel. Presented are their respective performance on the three metrics presented in Section 4.1.

It may be seen again (as for the quantity NO_x) that the LSTM model is outperforming the CNN model. Only the RPE, however, is substantially better than previous. The other metrics, *cycle total* and R^2 are both close in size. The Relative Percentage Error is again close in value to the ones obtained when testing the models during training, but now for an entire cycle. The models capture *cycle total* and R^2 better for Flow Fuel Diesel than for Concentration of NO_x, as there is almost no mistake in these metrics (recall that maximum values for these are 100 % and 1).

In Figures 7.5-7.6, the prediction made by the networks are seen against the true sequence of Flow Fuel Diesel for the CNN and LSTM-models respectively.

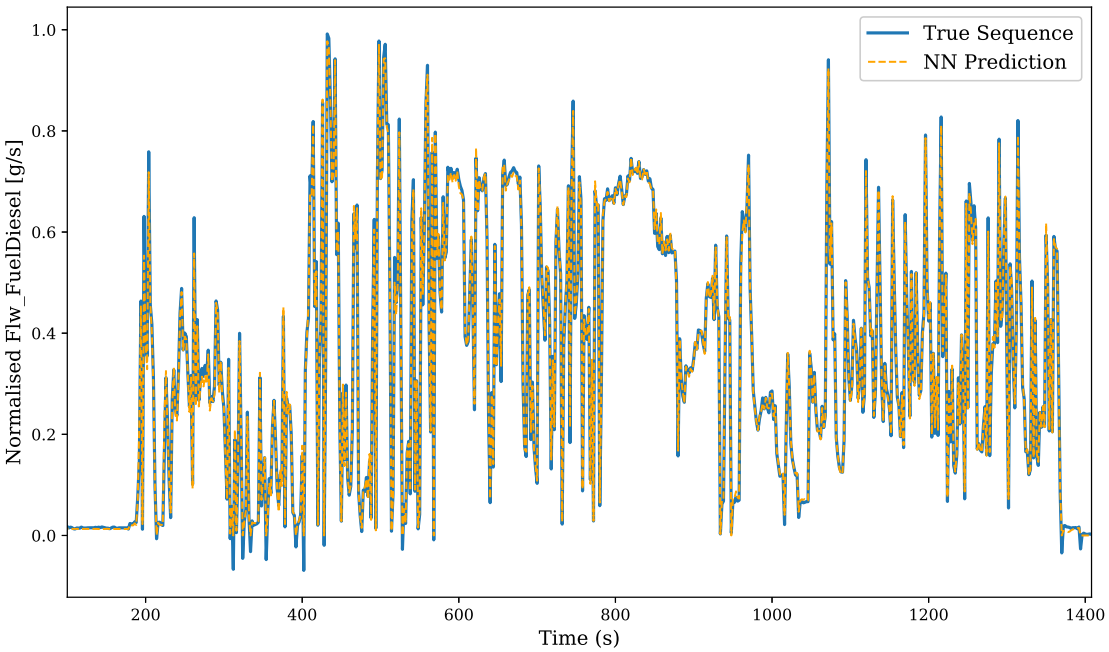


Figure 7.5: A prediction made by the CNN baseline model for Flw_FuelDiesel against the true sequence of NRTC cycle 4.

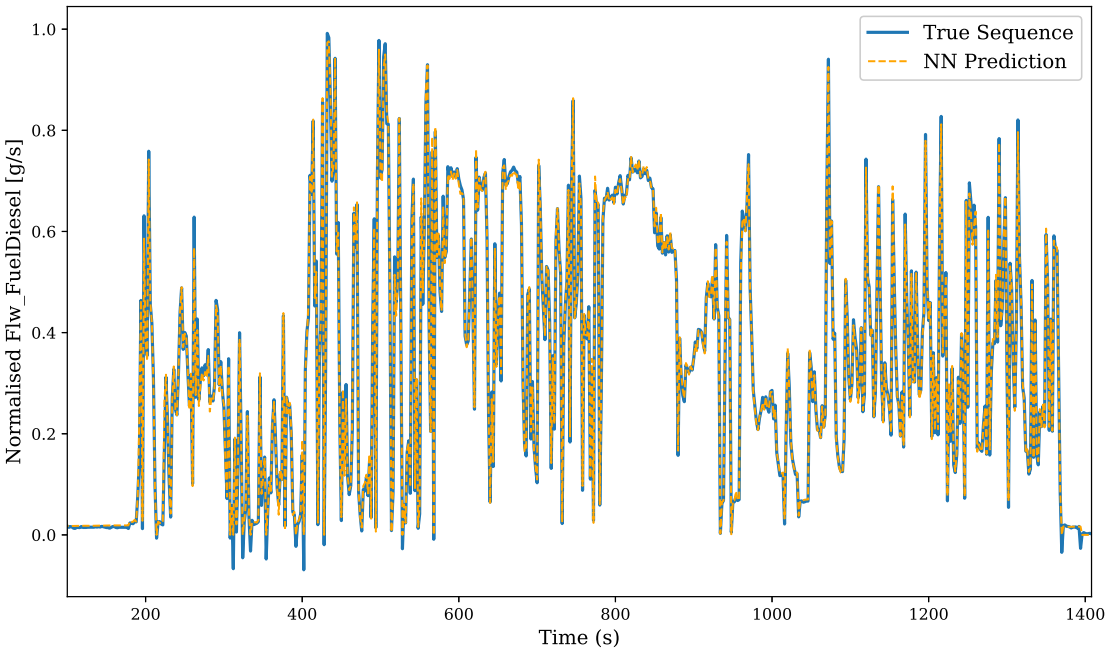


Figure 7.6: A prediction made by the LSTM baseline model for Flw_FuelDiesel against the true sequence of NRTC cycle 4.

Concentration of Soot

In Table 7.3, the models evaluated on the metrics may be seen for the quantity Concentration of Soot.

	Metrics		
	RPE	*Cycle Total*	R^2
CNN	18.95	98.60	0.838
LSTM	18.19	98.05	0.834

Table 7.3: The performance of the two model types (CNN and LSTM) on the measured quantity Concentration of Soot. Presented are their respective performance on the three metrics presented in Section 4.1.

As shown in Table 7.3, the performance of the networks on Soot is worse than on other quantities. For once, the Convolutional Neural Network is outperforming the LSTM network on one of the metrics: R^2. RPE is as high as 18.95% and 18.19% for the networks respectively, while the *cycle total* metric is comparable to those of the two other quantities. In Figures 7.7-7.8, the prediction made by the networks are seen against the true sequence for the CNN and LSTM-models respectively. Compared to Figures 7.1- 7.2 and Figures 7.5- 7.6, it may be seen that the behaviour of the Soot-emissions is more fluctuating and the network thereafter. Most of the spikes are off and very few time instances capture the magnitude. This is somewhat expected since the quantity Concentration of Soot is harder to model and has more complex dynamics.

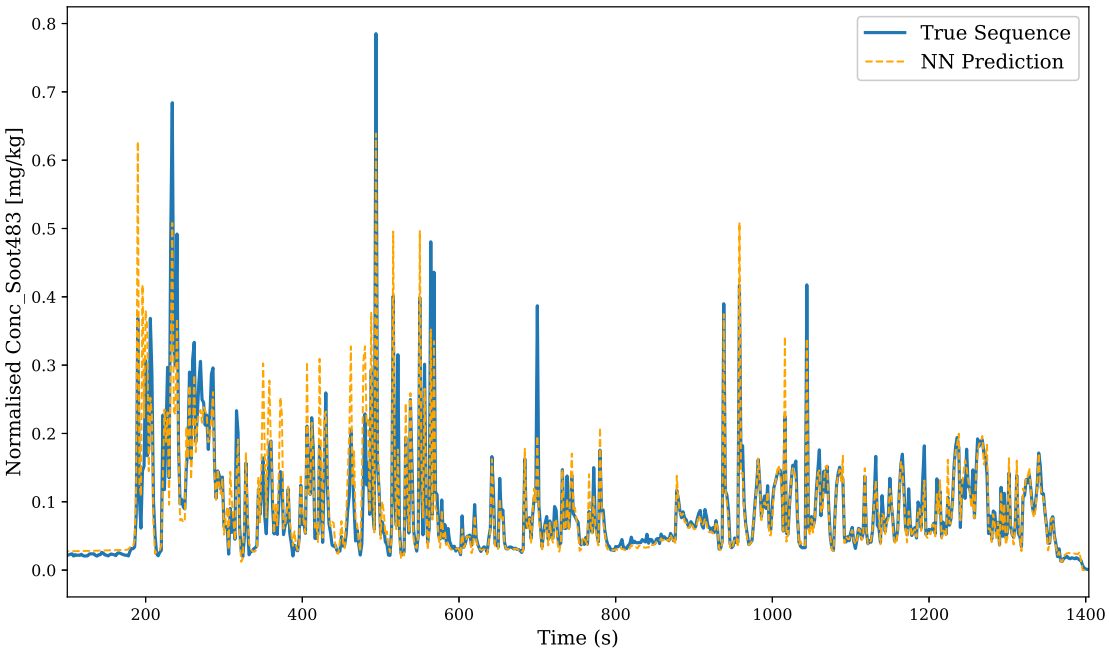


Figure 7.7: A prediction made by the CNN baseline model for Conc_Soot against the true sequence of NRTC cycle 4.

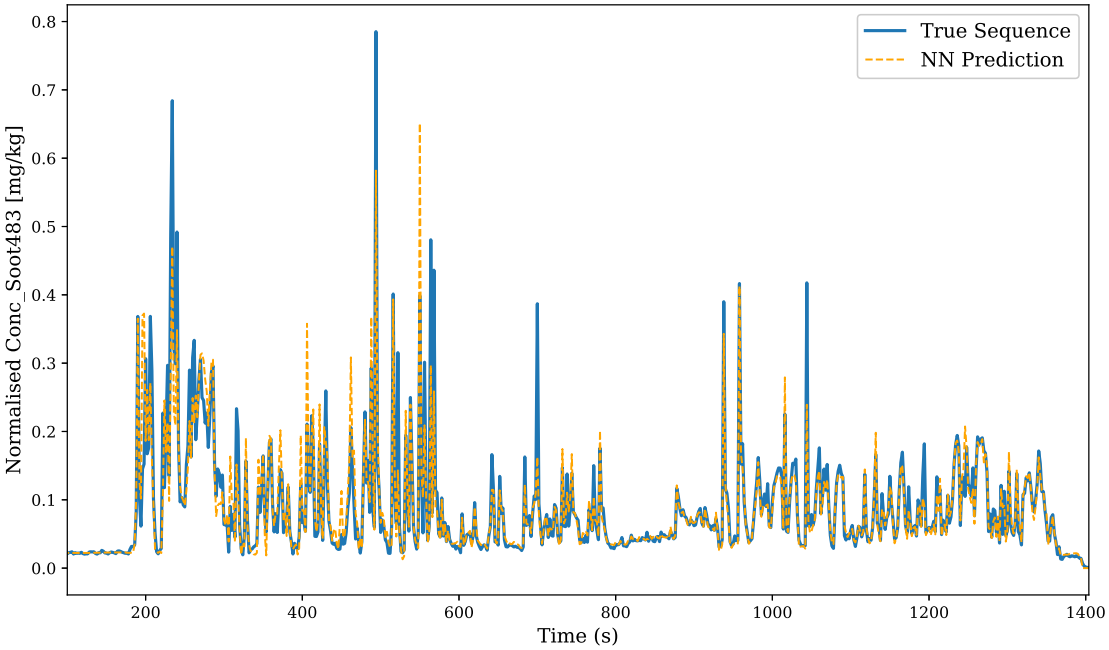


Figure 7.8: A prediction made by the LSTM baseline model for Conc_Soot against the true sequence of NRTC cycle 4.

For Concentration of NO_x, the region $3\ 00 - 5\ 00$ was zoomed in in Figures 7.3-7.4 to display how well it captured some of the more fluctuating parts of the cycle. The same region may be seen for Concentration of Soot, in Figures 7.9-7.10.

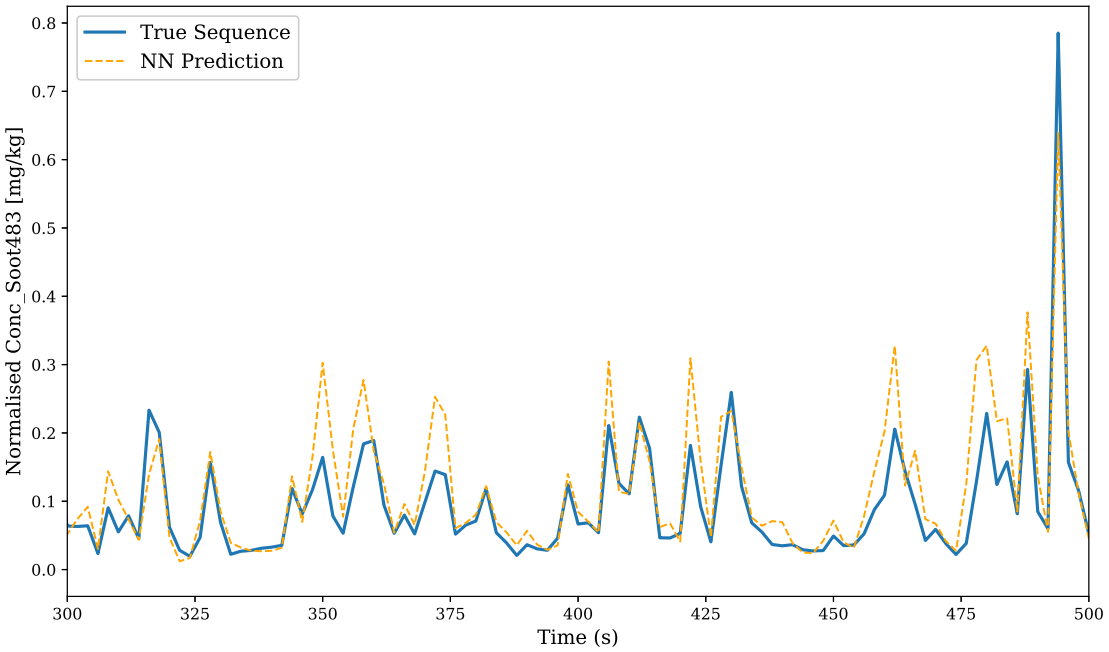


Figure 7.9: A prediction made by the CNN baseline model for Conc_Soot against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the overall difficulties of capturing the dynamical behaviour of Soot.

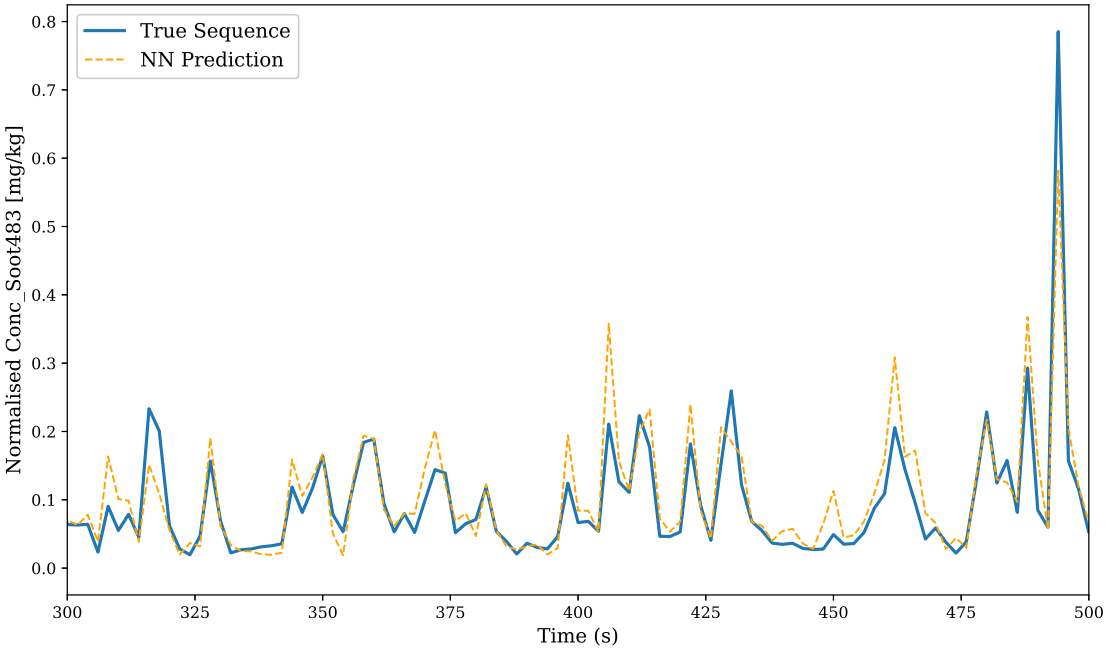


Figure 7.10: A prediction made by the LSTM baseline model for Conc_Soot against the true sequence of NRTC cycle 4. Zoomed in in the region of 300-500 to demonstrate the overall difficulties of capturing the dynamical behaviour of Soot.

Summary of the baseline models

In Table 7.4, a summary of the baseline model's performance with respect to the different quantities and metrics is seen.

	CNN			**LSTM**		
Quantity	Flw_Fuel	NO_x	Soot	Flw_Fuel	NO_x	Soot
RPE	2.44	3.41	18.95	2.40	2.80	18.19
Cycle Total	99.74	98.64	98.60	99.83	99.00	98.05
$\mathbf{R^2}$	0.998	0.993	0.838	0.997	0.995	0.834

Table 7.4: The performance of the two model types (CNN and LSTM) on the three measured quantities Flow Fuel Diesel, Concentration of NO_x and Concentration of Soot. Presented are their respective performance on the three metrics presented in Section 4.1.

7.2 Robustness

For the robustness Section, a grid search over the α- and β-parameters of the noise configuration was performed in Chapter 5. Based on that grid search a final noise configuration for CNN and LSTM independently could be decided upon. The following α and β values showed the best performance for each network structure:

Network Type	α	β
LSTM	1	0.3
CNN	1	0.6

Table 7.5: A Table with the final setups of α and β for the robustness part.

The models trained with these noise configurations were evaluated on *NRTC* 1-8. The performance may be seen in Table 7.6 for the target signal Conc_NOx.

NO_x	**NRTC** (RPE, %)							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
LSTM	4.24	4.50	5.23	2.80	3.88	3.89	4.32	3.47
LSTM Noise	4.37	4.93	5.50	3.01	3.43	3.83	4.54	3.05
CNN	5.29	5.27	5.74	3.41	3.79	3.77	5.10	3.64
CNN Noise	5.57	5.45	5.59	3.31	3.83	3.87	4.93	3.74

Table 7.6: A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal NO_x.

From Table 7.6 it can be seen that the models do not improve on all cycles, compared to the **baseline** models. Instead, a similar behaviour for both the CNN and LSTM models is found: they improve on the cycles where the baseline models had the most difficulties with. The performance was also decreased on the cycles the baseline models performed best on. This is seen more on the LSTM network which appears to benefit more from the noise training than its CNN counterpart.

In Table 7.7, the performance for the models trained on target signal Flw_FuelDiesel may be seen.

Flow fuel	**NRTC** (RPE, %)							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
LSTM	3.61	6.24	6.61	2.40	3.65	3.73	3.19	3.92
LSTM Noise	2.91	5.65	5.94	2.55	3.45	3.80	3.31	4.12
CNN	3.04	5.37	5.84	2.44	3.91	4.17	3.09	3.93
CNN Noise	3.06	5.18	5.77	3.57	4.86	4.89	3.31	3.94

Table 7.7: A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal Flow Fuel Diesel.

Like the results for the NO_x models, the performance of the Flow Fuel Diesel models was overall increased on the cycles the initial model had difficulties with, and decreased on the cycles it performed well on. This behaviour is especially clear for the LSTM model.

In Table 7.8, the performance for the models trained on target signal Conc_Soot may be seen. The results from the Soot robustness were different from the others.

Soot	**NRTC** (RPE, %)				
	1	**4**	**5**	**7**	**8**
LSTM	26.46	18.19	21.70	22.08	28.86
LSTM Noise	27.00	17.74	20.26	21.82	25.92
CNN	26.90	18.95	19.13	24.31	27.00
CNN Noise	25.71	19.78	19.96	24.46	27.52

Table 7.8: A Table of the noise-trained models together with the baseline models. Shown is the RPE value of each NRTC cycle for each model for the target signal Soot.

As seen in the Table, and mentioned before, the results of the Soot robustness models were different from the previous ones. For the LSTM model, all *NRTC* cycles improved, ever so slightly, with the introduced noise, except for *NRTC* 1. For the CNN model, however, the performance was decreased for all cycles except the first. The difference between the robust and **baseline** models, however, is very small compared to the difference for the other target signals.

7.3 Transfer Learning

In Chapter 6, it was described how the baseline models were retrained by freezing certain layers in the network structure while training others. The grid search of that Chapter showed that retraining all layers of both network structures provided the lowest RPE.

The results of the networks are presented in Table 7.9, where the respective performance on *RPE*, *Cycle Total* and R^2 are shown for the **baseline** models and retrained models. The retrained models are denoted as **TF** and the evaluated target signal is NO_x.

NO_x	**NRTC 10**		
Metric	*RPE*	*Cycle Total*	R^2
LSTM Baseline	4.31	99.26	0.985
LSTM TF	2.34	99.30	0.995
CNN Baseline	4.53	98.27	0.984
CNN TF	2.20	99.95	0.996

Table 7.9: The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated for NO_x. Shown is the three metrics described in Section 4.1 on the new Test-set *NRTC 10*-cycle.

As it may be seen in the Table, both models improve their performance by $45 - 50\%$, in RPE while improving the other two metrics ever so slightly. It may be

seen that before transfer learning was applied, the LSTM model outperformed the CNN on the cycle. However, after transfer learning was used, the CNN model has better values for all metrics.

As the transfer learning is applied to the baseline models, which were trained for the initial case, the weight and biases are tuned in a new direction. To make sure the models are not overfitted to the new data-set, they are evaluated against the test-set for the **baseline** models, *NRTC-4*. Seen in Table 7.10 is the performance of the **baseline** and **TF** models on the original test-set.

NO_x	**NRTC 4**		
Metric	*RPE*	*Cycle Total*	R^2
LSTM Baseline	2.80	99.00	0.995
LSTM TF	3.32	99.02	0.994
CNN Baseline	3.41	98.64	0.993
CNN TF	4.07	99.36	0.989

Table 7.10: The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated on the three metrics described in Section 4.1 on the old Test-set *NRTC 4*-cycle.

It may be seen in Table 7.10 that the performance of the transfer learned models is decreased on the original test-data, as the weights and biases are shifted to suit the new data-set. However, the decreased performance on *NRTC-4* is lower than the increased counterpart on the new test-set *NRTC-10*.

Additionally, the same configuration was used in order to create transfer learned models for the Flow Fuel Diesel and Concentration of Soot. The results of these models are shown in Table 7.11.

	NRTC 10 (RPE, %)	
	Flow Fuel	**Soot**
LSTM Baseline	4.43	31.06
LSTM TF	4.51	32.67
CNN Baseline	4.42	30.22
CNN TF	4.54	13.55

Table 7.11: The performance of the two types, CNN and LSTM, **baseline** and **TF** models evaluated on RPE as described in Section 4.1 on the new test-set *NRTC 10*-cycle.

Table 7.11 shows that for Flow Fuel Diesel neither the LSTM nor the CNN model shows any improvement, instead they remain at a fairly similar RPE-value, slightly higher than before. The LSTM model's performance for Soot does not benefit from transfer learning either and instead, it shows a slightly higher RPE-value after the transfer learning. The CNN model, however, improves its performance for Soot by 50% (decreasing the error RPE) as compared to the baseline model.

8
Discussion

In this Section, the results and possible factors which affect the results are discussed. This concerns data-handling, methodology and processes which could have been performed differently during the project. Additionally, it covers possible areas which could be investigated further in the future.

Input signals and feature scaling

When it comes to machine learning, there are multiple factors affecting the results of a neural network. Due to the nature of data-driven modelling, one of the more prominent factors is the way input data is handled. In this project 13 signals were selected from the 80 available, to speed up the training. However, all 80 could be used, allowing the network to "zeroing" out the redundant signals, ensuring no correlations are disregarded. Additionally, as the Convolutional Neural Network finds correlation between signals over time, the order of the signals matters. A more thorough study on how the signals correlate could improve the performance.

The feature scaling is another possible factor for the results: in the project normalisation and standardisation (described in Section 3.5) scaling techniques are used where there might be more reasonable selections for the scaling (e.g. the *EGR Position* signal could be scaled with the hyperbolic tangent, see Figure 3.6). As mentioned in Section 3.5, the NaN-values are handled through interpolation. Instead, there could be argued that a more reasonable way is to zero-out those part and consider it as noise, or entirely exclude these from the data.

The noise added to the data for the robustness is estimated. The accuracy for each sensor is not known to the decimal while the uncertainties of the modelled parameters are hard to estimate. The realistic amount of noise could be found through an extensive search, however, for this thesis they are estimated through expertise at the company, or found for similar sensors online.

Choice of hyperparameters

Another crucial factor affecting the performance of a neural network is the way the hyperparameters are selected. Since the goal of the project was not to generate the perfect model (see Sections 1.2-1.3), but rather to demonstrate that neural networks can be an efficient solution for this application, the best possible option for models was not aimed at. For instance, the number of convolutional layers, their kernel size and the optimal number of neurons in a layer were selected once the results were

sufficient and satisfactory. An even deeper network, more than three convolutional layers and different kernel sizes in each layer could possibly generate an even more accurate model. As shown in Figure 4.5, the least number of neurons tried was 25 for the LSTM while it appears that even fewer could provide a better result. In Figure 4.2 it may be seen that more layers could possibly improve the quality of the network, but they had at that point provided satisfactory results.

The same may be said about the loss function: RMSE is a widely used loss function for regression tasks and was successfully used, even though there exists a variety of other functions that may provide similar or even better results. The choice of optimiser, *Adam*, could be further investigated as there are plenty of viable options. However, it proved to be efficient, both in training time-wise and accuracy-wise.

The choice of time-steps for the networks (recall 10 for Flow Fuel Diesel, 30 for NO_x and 50 for Soot) was selected balancing the training time against the accuracy. For the more complex Soot, a higher time-step such as $60-80$ could possibly increase the ability of the network. However, the number of parameters would increase a lot for the Convolutional Neural Network and extensive training would be needed.

Also, all tuning of the models: baseline, robustness and transfer learning grid searches are done against one of the target signals, NO_x. This was done since tuning against all three target signals would have taken too much time. However, tuning against one of the other two signals could have yielded a very different result. For instance, the input size would have changed, leading to another configuration and possibly fewer/more layers, depending on the signal.

As the network structures for the robustness were the same as the baseline models, no real tuning of its hyperparameters was done. Instead, the grid search was performed with two configurations of noise: scaled by a factor $\alpha = (1,\ 5)$ and the amount of data given by $\beta = (0.3,\ 0.6,\ 1.0)$. Iterating through other values, testing, for instance, random search methods could provide a better configuration for the robustness part.

The configuration found for transfer learning was mainly based on the methodology of [11]. The grid search was performed by iterating through the frozen layers, while other possibilities exist. A relatively small amount of time was spent on this and more investigation could be done in this area. However, satisfactory results were found for transfer learning as well.

Results

The results for all of the models presented in Section 7.1 shows a reliable performance. The evaluation cycles are all of type *NRTC*, previously unseen for the models but the training-set, however, consists partly of *NRTC*. It is possible the samples used for evaluation are fairly close to some of the samples in the training data which could make the evaluation slightly biased. Looking at the distribution of the training and test data (in the space of *engine speed* and *torque*) in Figure 3.11 it can be seen that *NRTC 4* lies within the region of the training data. It is important to note that the cycle is a completely separate run, where the inputs and outputs differ, compared to the *NRTC* cycles in the base-set. However, with the specific nature

of time-dependency in the networks (especially the LSTM), the overlap between the training and test cycles is necessary. It would, however, be very interesting to evaluate on a completely different type of cycle, other than the *NRTC*, solely for evaluation purposes.

When training the models with added noise, as explained in Chapter 5, a slight improvement in the generalisation can be seen for some of the models. The results in Section 7.2 shows that most of the models perform slightly better at cycles where a high RPE was seen before and slightly worse where a low RPE was seen. Additionally, the averaged error over the cycles for each LSTM model is decreased with noise added. This could indicate that the models become better to handle the differences in between the cycles, which is the aim of training with noise. One reason for only seeing small changes in the generalisation could be that a large amount of initial data has been used. The baseline models are, thus, fairly generalised to begin with. Hence, it could also be argued that the behaviour of the NRTC cycles used for evaluation might not be different enough for the data-augmentation to show any large-scale differences. It can also be argued that the test cycles do not include any significant amount of signal noise, meaning that the robustness is hard to evaluate with these cycles. It would be of interest to evaluate on cycles with different behaviour: for instance, cycles where the engine have aged, cycles where the calibration is vastly different or customer-data where noisy signals are more prominent.

The results of the transfer learning which can be seen in Section 7.3 shows very promising results for the NO_x-sensor where both the LSTM and the CNN lower their RPE-value down to about 50%. For the other sensors the only model that showed an improvement in RPE was the CNN model for Soot. However, the evaluation is once again done against an *NRTC*. To properly test the "transferability" of the virtual sensor models, more extensive test could be done on a different cycle, with a different engine, hopefully ensuring that the same results could be found there. As the re-training is performed for only 12 epochs on a limited new training-set, it is not unlikely that this procedure should provide similar results for another engine and cycle. An additional point worth mentioning is the behaviour during the grid search, in Section 6.1, where the CNN models' performance is increased only when all fully connected layers or all convolutional layers are re-trained. Training one fully connected or all fully connected and one or two convolutional layers decrease the performance.

On-vehicle implementation

Additionally, the models generated in this thesis have been implemented on the hardware-platforms *Raspberry Pi 3* and *FPGA* in a thesis running parallel to this [29]. The thesis explores the possibility of implementing LSTM and CNN models in terms of memory-usage, computational time and accuracy. The implementation was done successfully which means that the work of our thesis could be applied in the virtual test system at Volvo Penta.

More specifically the thesis shows that, implemented on a Raspberry Pi 3, the LSTM model uses up $184MB$ of memory and has a computational time of $50ms$. The

CNN, on the other hand, shows a memory-usage of $150.12MB$ and a computational time of $20ms$ per sample [29]. The CNN model, even with a higher number of parameters than the LSTM, can produce predictions faster due to the layout of the 2D grid. Predictions are calculated in parallel, speeding up the calculations, while the LSTM models require sequential computations, taking more time and using more memory.

9
Conclusion

The aim of this thesis was to investigate the possibilities of using machine learning models as a substitute for physically modelled sensors. Two network structures were proposed, one LSTM and one CNN, and evaluated. We found that reliable models for Flow Fuel Diesel, NO_x and Soot could be developed using the methodology of this thesis. The best performance was found from the LSTM model with 2.40%, 2.80% and 18.19% error for the sensors respectively. The performance of the CNN model showed a slightly higher error.

In *Heavy Duty Diesel Engine Modeling with Layered Artificial Neural Network Structures* by Sediako, A.D., Andric, J., Sjöblom, J., and Faghani, E. [3] a similar study was done. The emission sensors, NO_x and Soot, were modelled and provided 98% and 86% for cycle total and 0.91 and 0.1 for R^2 respectively. The models of this thesis exceeded these values by providing a result of 99.0% and 98.05% (cycle total) and 0.995 and 0.834 (R^2). However, it is important to note that different cycles were used both for training and testing, in between the two studies.

The robustness against noise and different calibrations provided ambiguous results. We saw that augmenting the data with noise increased the ability to generalise for the LSTM models (except for the NO_x sensor) while the CNN model does not appear to benefit from the augmentation at all. The increased performance of the LSTM were moderate at best (decreased error by 4% on average) and we conclude that there could be benefits from augmenting the data but that, as mentioned in the discussion, the amount of extra data and noise levels needs to be carefully selected.

The transfer learning results show more promise than the ones of the robustness, but there is ambiguity here as well. For the CNN models, transfer learning overall provides a satisfactory result as the error (RPE) of NO_x on the test cycle is decreased by 51% and 55% on the Soot sensor. The LSTM shows a similar result on NO_x, however, all other models provide worse results. We conclude that transfer learning is a viable option for the convolutional neural network, improving massively on the test-cycle with relatively little training and data. To further conclude anything about the LSTM models, more testing of the configuration is required. It is also important to note that the re-training change the ability to generalise as the weights and biases are shifted toward the new set, "forgetting" the old configurations, making it slightly overfitted on the new data. Thus, we believe that the models that did not improve from transfer learning require re-training on data from more than one cycle, and to be tested on more than just another *NRTC*.

By weighing in the results given by [29] with the overall performance we conclude that the LSTM models provide higher accuracy while the CNN models leave a smaller computational footprint, due to the ability of parallel computations of

the 2D-input. The CNN model is therefore a viable model, especially in real-time applications such as *HiL*-system. These systems often use more than one model which means that the computational advantages become even bigger. Additionally, for the emission sensors it might be more important to capture the total amount of emissions released over an entire cycle. In this scenario, the metric *cycle total* is used, which provides great results for all models. This could further justify choosing CNN over LSTM models, even though the latter provides slightly higher accuracy.

Bibliography

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[2] R. Isermann, J. Schaffnit, and S. Sinsel. Hardware-in-the-loop simulation for the design and testing of engine-control systems. Control Engineering Practice, 7(5):643 – 653, 1999.

[3] Andric J. Sjöblom J. Sediako, A.D. and E. Faghani. Heavy duty diesel engine modeling with layered artificial neural network structures. SAE Technical Papers, April 2018.

[4] Alex Graves. Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence. Springer, Berlin, 2012.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Comput., 9(8):1735–1780, November 1997.

[6] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2):68–75, 2017.

[7] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[8] Yann Lecun and Y Bengio. Convolutional networks for images, speech, and time-series. The Handbook of Brain Theory and Neural Networks, 01 1995.

[9] M. Lopez and W. Yu. Nonlinear system modeling using convolutional neural networks. In 2017 14th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), pages 1–5, Oct 2017.

[10] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In CVPR Workshops, pages 512–519. IEEE Computer Society, 2014.

[11] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[12] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. Neural Networks, 4(2):251–257, 1991.

[13] Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In ICCV, pages 2146–2153. IEEE, 2009.

[14] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). CoRR, abs/1511.07289, 2016.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics, 2010.

[16] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2016. cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam.

[19] Daniel Svozil, Vladimir Kvasnicka, and Jiří Pospíchal. Introduction to multilayer feed-forward neural networks. Chemometrics and Intelligent Laboratory Systems, 39:43–62, 11 1997.

[20] Selim Aksoy and Robert M. Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. Pattern Recognition Letters, 22:563–582, 2000.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015.

[22] Yves Chauvin and David E. Rumelhart, editors. Backpropagation: Theory, Architectures, and Applications. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995.

[23] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. Trans. Neur. Netw., 5(2):157–166, March 1994.

[24] Lino Guzzella and Christopher Onder. Introduction to Modeling and Control of Internal Combustion Engine Systems. Springer; 2nd ed. 2010 edition (December 16, 2009), 01 2010.

[25] Directive 2004/ 26/ec of the european parliament and of the council of 21 april 2004 amending directive 97/68/ec on the approximation of the laws of the member states relating to measures against the emission of gaseous and particulate pollutants from internal combustion engines to be installed in non-road mobile machinery. https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2004:146:0001:0107:EN:PDF. Accessed: 2019-03-26.

[26] François Chollet et al. Keras. https://keras.io, 2015.

[27] Allin Cottrell. Regression analysis: Basic concepts. pages 1–16, 05 2019.

[28] Tatiana Prisyach, Valentin Mendelev, and Dmitry Ubskiy. Data augmentation for training of noise robust acoustic models. In Analysis of Images, Social Networks and Texts - 5th International Conference, AIST 2016, Yekaterinburg, Russia, April 7-9, 2016, Revised Selected Papers, pages 17–25, 2016.

[29] Reema Pinto. Embedded machine learning for live sensor simulation (unpublished). Master's thesis - datx05, Chalmers University of Technology, 2019.

A

Appendix 1

In Appendix A.1, the 13 normalised input signals are presented on the *NRTC-ww* cycle and in Appendix A.2 the histograms, displaying their individual distribution along all cycles.

A.1 Input signals on the *NRTC-ww*

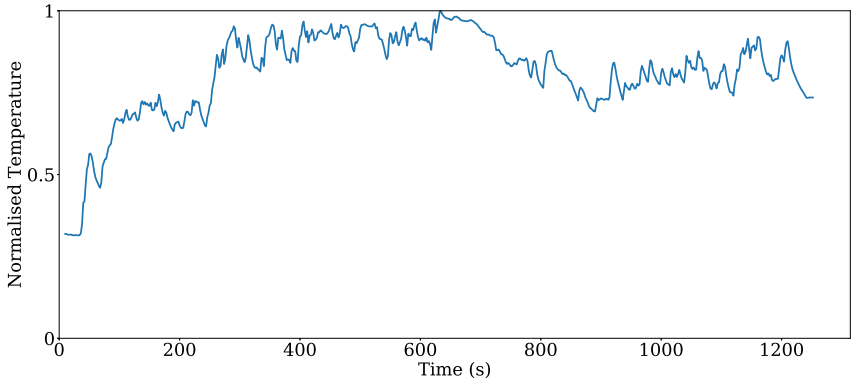


Figure A.1: The behaviour of the normalised *Exhaust Temperature* signal throughout the test-cycle *NRTC-ww*.

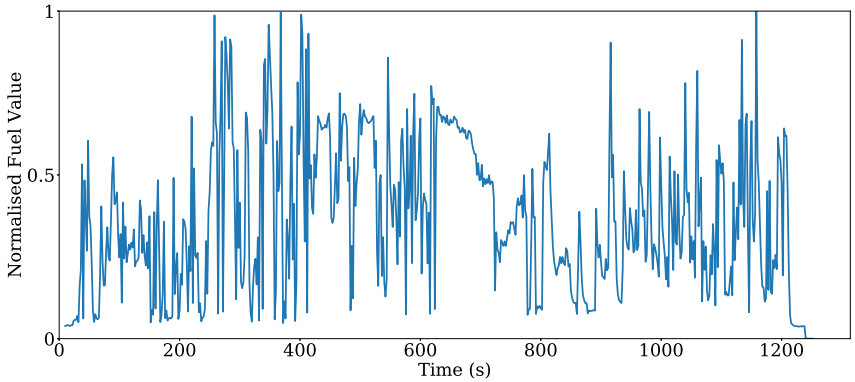


Figure A.2: The behaviour of the normalised *em_FuelValue* signal throughout the test-cycle *NRTC-ww*.

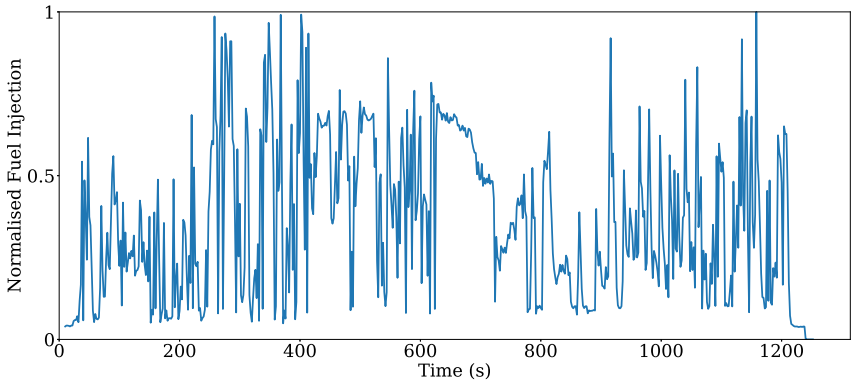


Figure A.3: The behaviour of the normalised *Main Injection* signal throughout the test-cycle *NRTC-ww*.

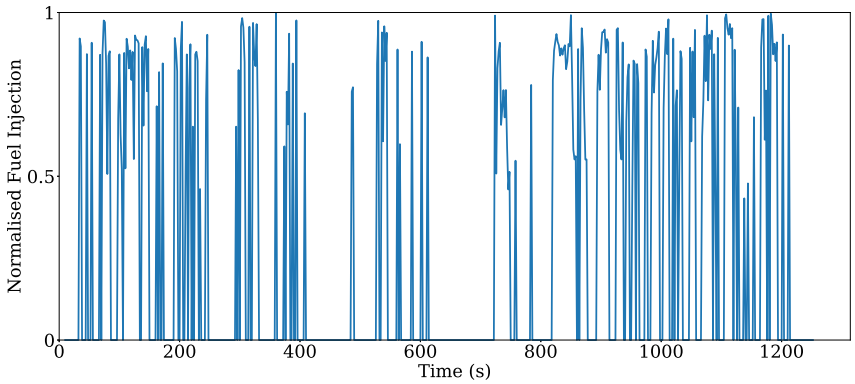


Figure A.4: The behaviour of the normalised *Post Injection* signal throughout the test-cycle *NRTC-ww*.

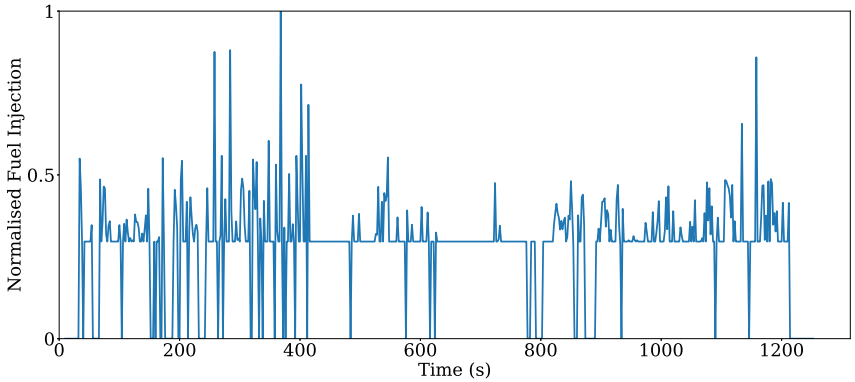


Figure A.5: The behaviour of the normalised *Pre Injection* signal throughout the test-cycle *NRTC-ww*.

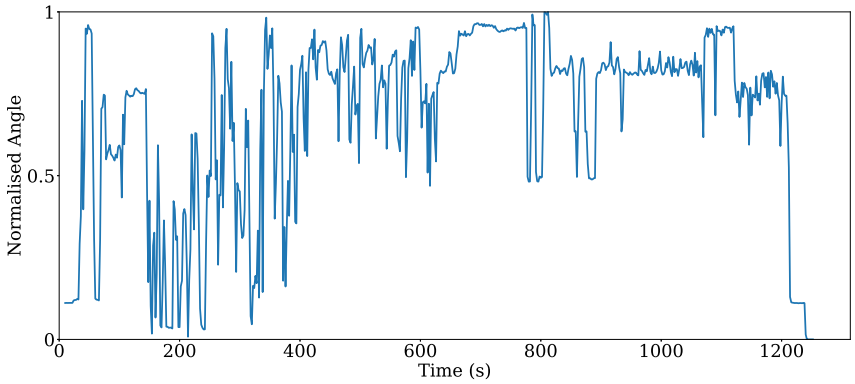


Figure A.6: The behaviour of the normalised *Pre Injection Angle* signal throughout the test-cycle *NRTC-ww*.

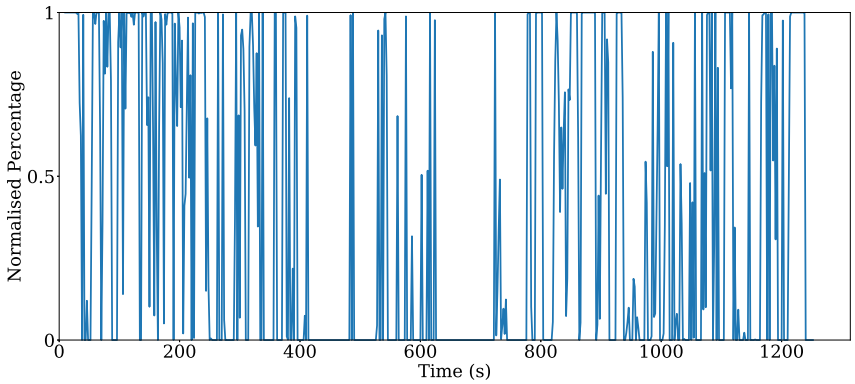


Figure A.7: The behaviour of the normalised *EGR Position* signal throughout the test-cycle *NRTC-ww*.

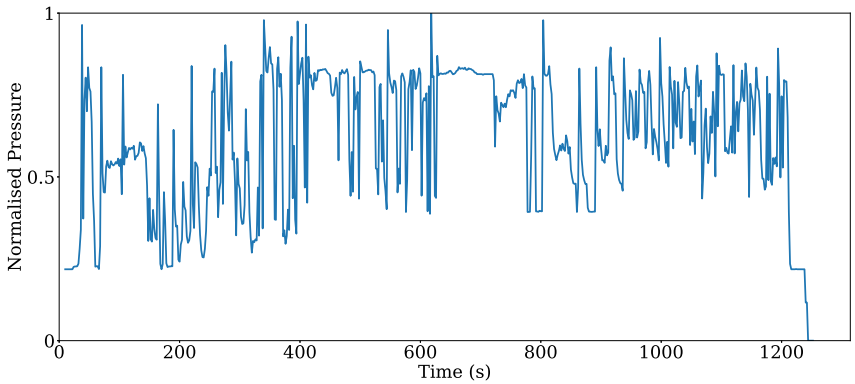


Figure A.8: The behaviour of the normalised *Rail Pressure* signal throughout the test-cycle *NRTC-ww*.

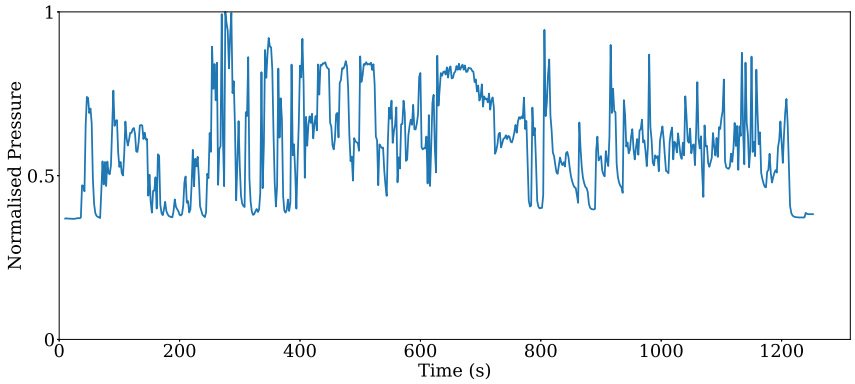


Figure A.9: The behaviour of the normalised *Inlet Position* signal throughout the test-cycle *NRTC-ww*.

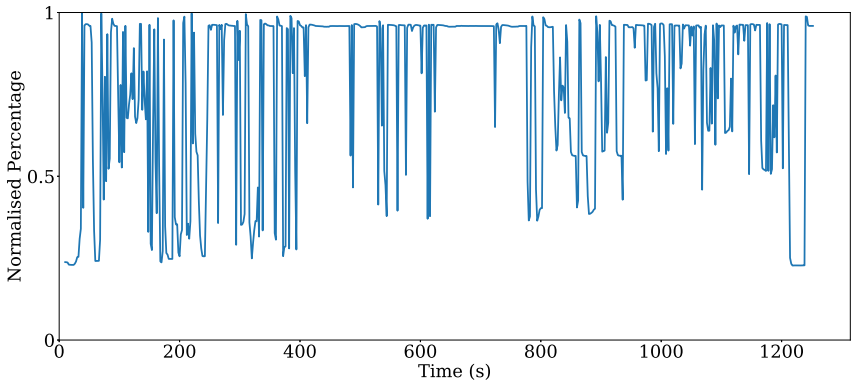


Figure A.10: The behaviour of the normalised *Throttle Position* signal throughout the test-cycle *NRTC-ww*.

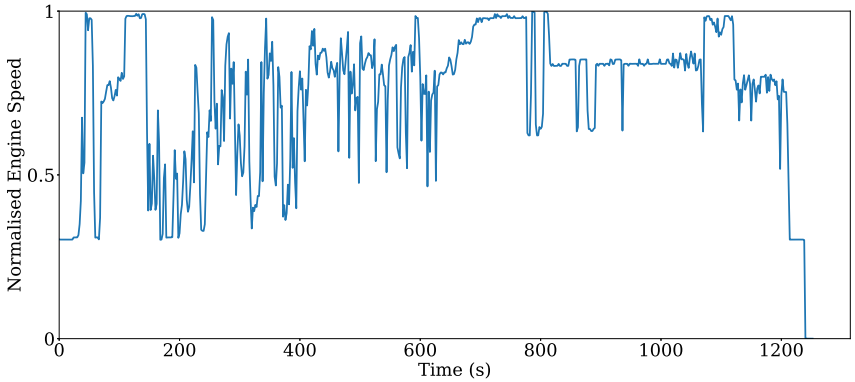


Figure A.11: The behaviour of the normalised *Engine Speed* signal throughout the test-cycle *NRTC-ww*.

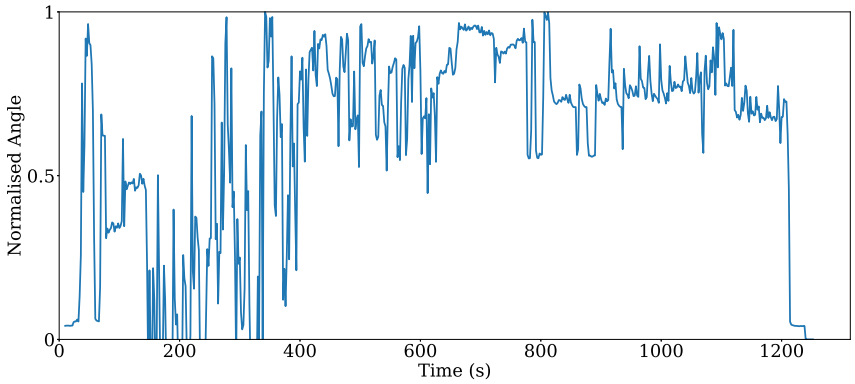


Figure A.12: The behaviour of the normalised *Injection Angle* signal throughout the test-cycle *NRTC-ww*.

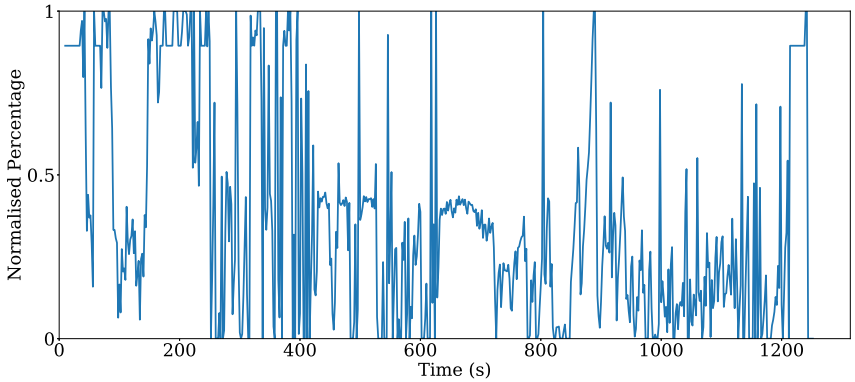


Figure A.13: The behaviour of the normalised *Wastegate Position* signal throughout the test-cycle *NRTC-ww*.

A.2 Histogram of the input signals

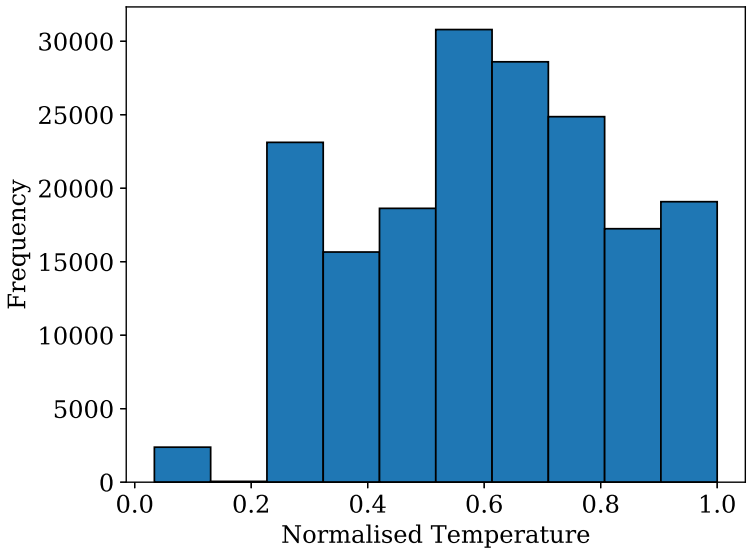


Figure A.14: The distribution of the normalised *Exhaust Temperature* signal over all input data.

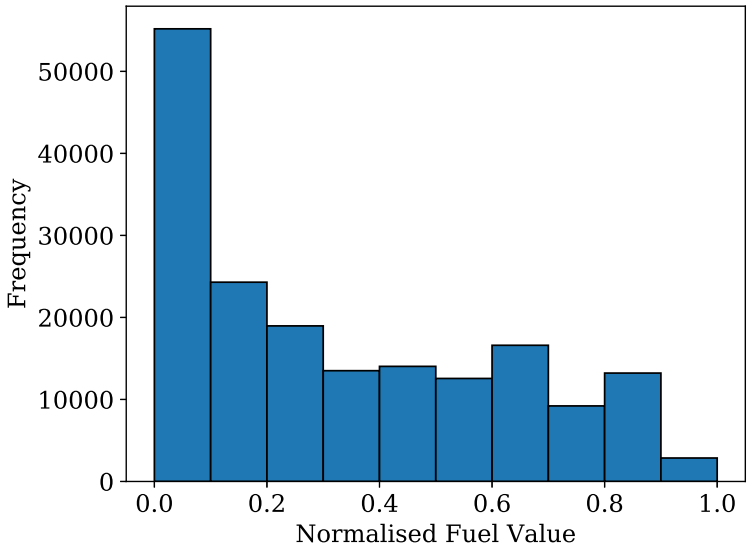


Figure A.15: The distribution of the normalised *em_FuelValue* signal over all input data.

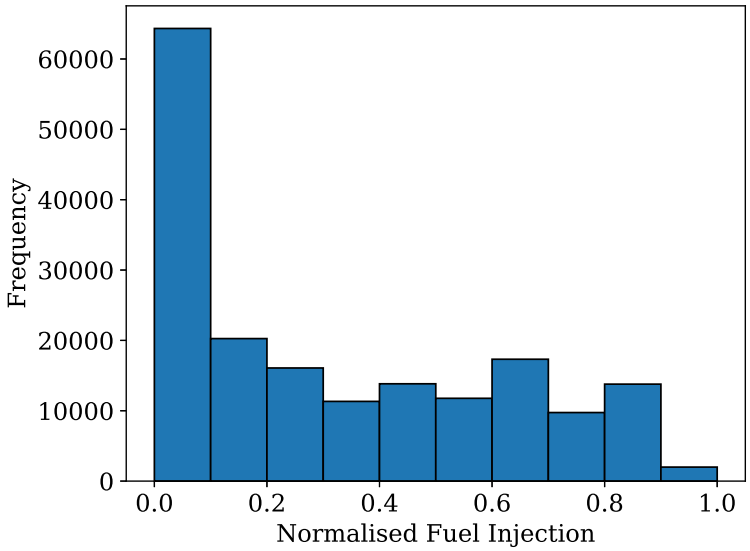


Figure A.16: The distribution of the normalised *Main Injection* signal over all input data.

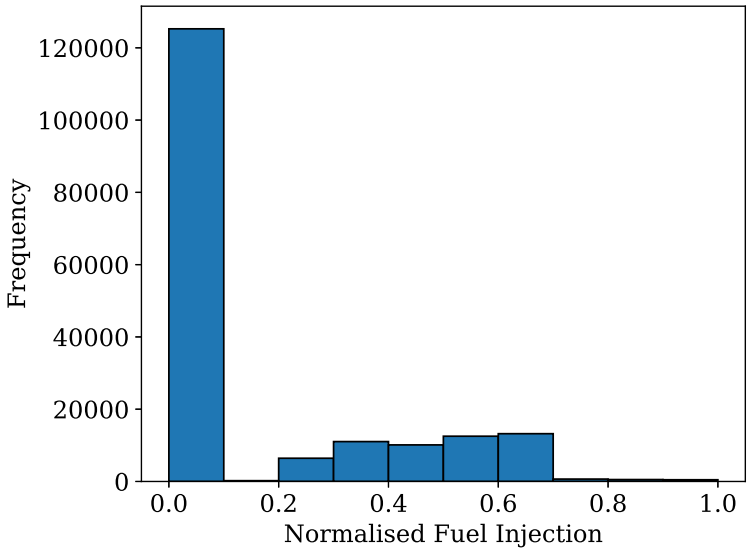


Figure A.17: The distribution of the normalised *Post Injection* signal over all input data.

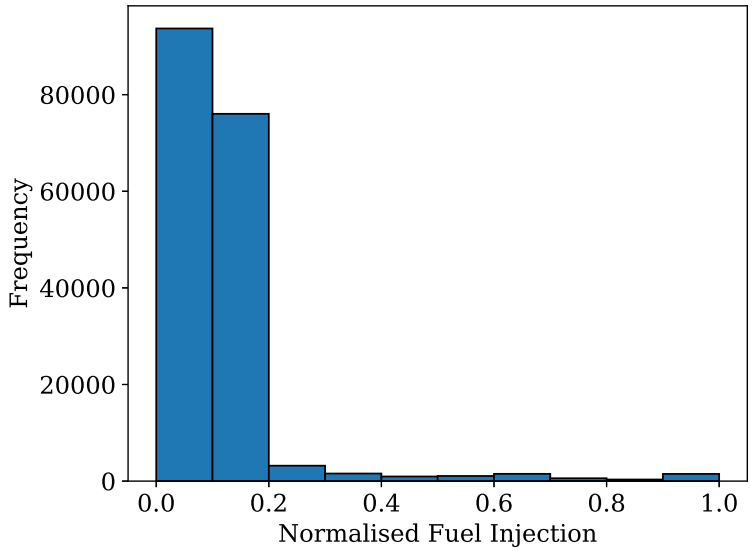


Figure A.18: The distribution of the normalised *Pre Injection* signal over all input data.

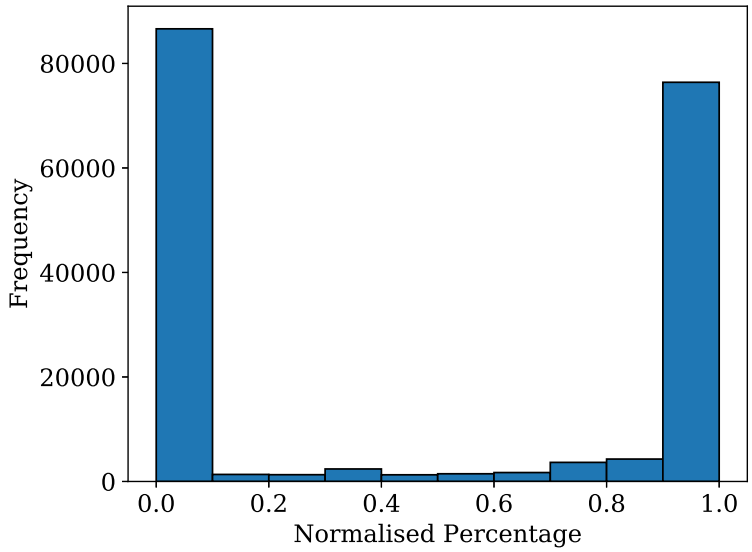


Figure A.19: The distribution of the normalised *EGR Position* signal over all input data.

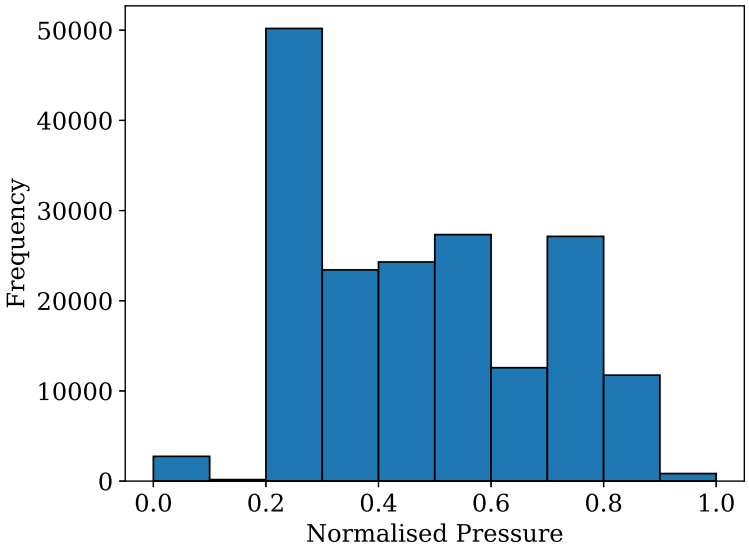


Figure A.20: The distribution of the normalised *Rail Pressure* signal over all input data.

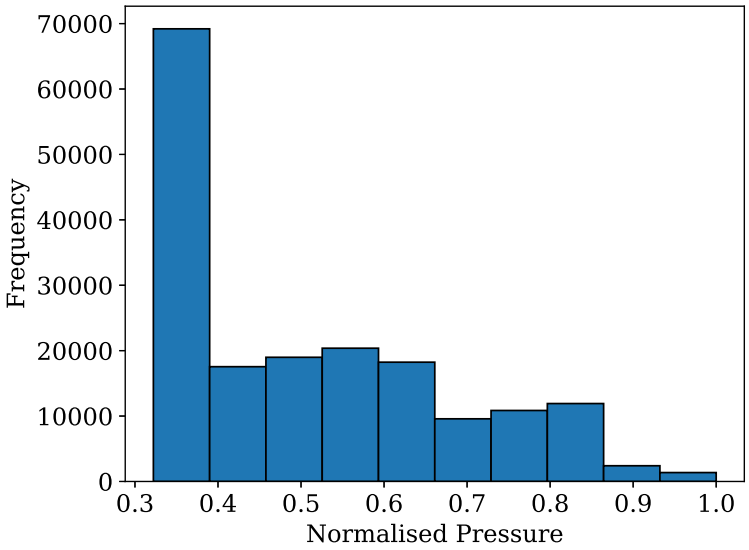


Figure A.21: The distribution of the normalised *Inlet Position* signal over all input data.

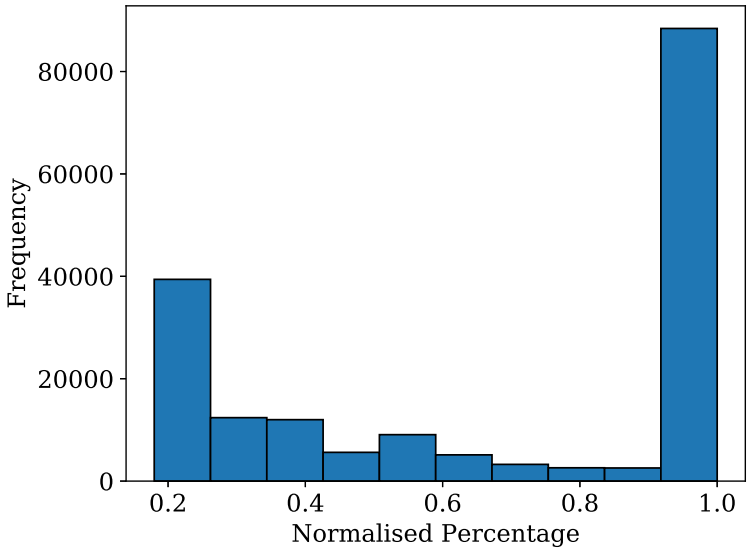


Figure A.22: The distribution of the normalised *Throttle Position* signal over all input data.

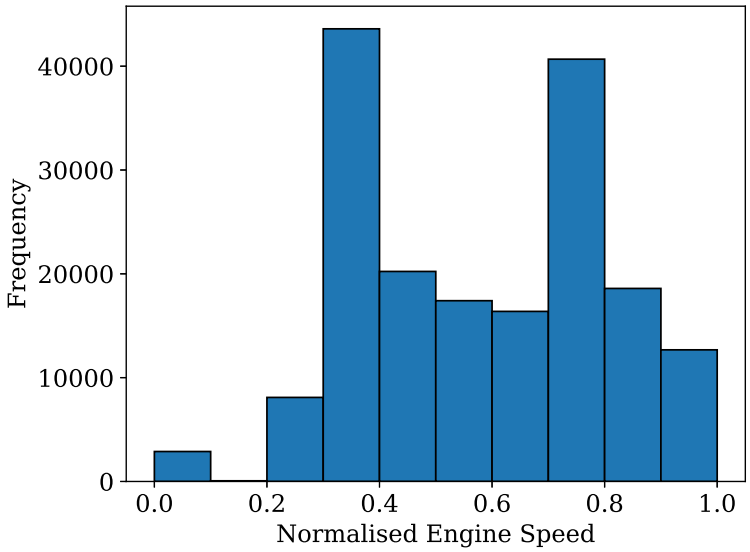


Figure A.23: The distribution of the normalised *Engine Speed* signal over all input data.

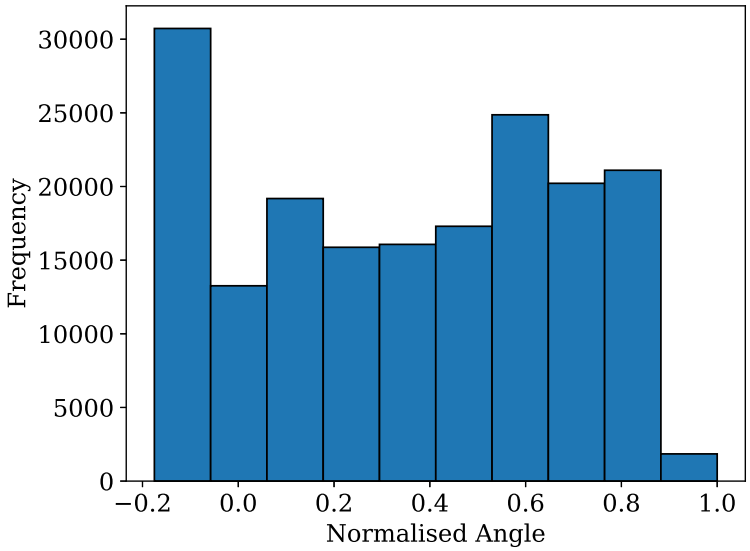


Figure A.24: The distribution of the normalised *Injection Angle* signal over all input data.

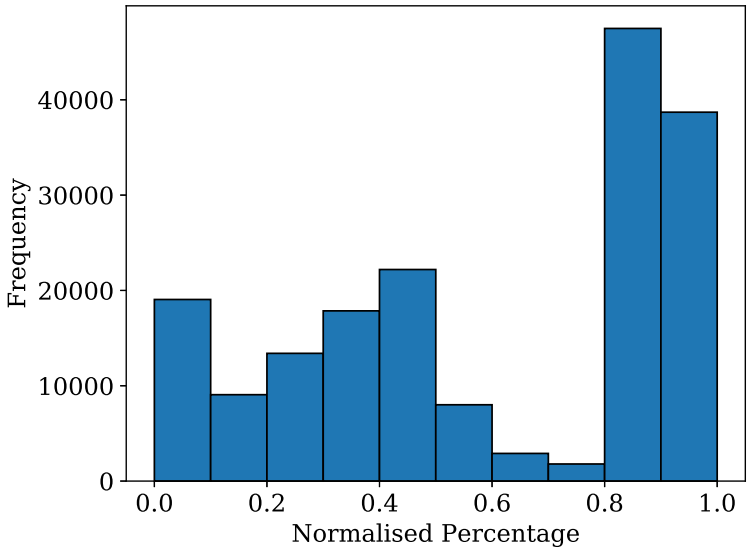


Figure A.25: The distribution of the normalised *Wastegate Position* signal over all input data.