Parameters Estimation in Marine Powertrain using Neural Networks

Vasiliki Tzoumezi

Diploma Thesis

School of Naval Architecture and Marine Engineering
National Technical University of Athens

Supervisor: Assistant Prof. George Papalambrou

Committee Member : Prof. N. Kyrtatos
Committee Member : Prof. Gr. Grigoropoulos

October 2020

Acknowledgements

This work has been carried out at the Laboratory of Marine Engineering (LME) at the School of Naval Architecture and Marine Engineering of the National Technical University of Athens, under the supervision of Assistant Professor George Papalambrou.

I would first like to thank my thesis supervisor, Assistant Professor George Papalambrou, for giving me the chance and motivation to broaden my horizons and work on the artificial intelligence topic for marine engines. I would also like to thank him for his patience, continuous support and immense knowledge. His spirit motivated me all this time to carry out this thesis.

I would like to thank Professor Nikolaos Kyrtatos for providing the opportunity to work with the full-scale hybrid diesel-electric marine propulsion powertrain of LME. I also thank him for being a member of my supervisors committee.

I would also like to thank Professor Gregory Grigoropoulos for evaluating my work and being a member of my supervisors committee.

I owe my heartfelt gratitude and sincere thanks to Mr. Nikolaos Planakis, PhD candidate of School of Naval Architecture and Marine Engineering for his contribution to overcoming several challenges. I am extremely thankful to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

I take this opportunity to express gratitude to all LME fellow members for their help and support.

Finally, I am sincerely grateful to my family for always being there for me and for the unceasing encouragement and support throughout my studies, to my friends for always believing in me and supporting me every step of the way, and to the person who inspired me to discover the beautiful world of artificial intelligence and an unexplored part of myself.

This accomplishment would not have been possible without any of you. Thank you.

Abstract

There is a wide range of engine performance parameters affecting the combustion process, while some of them have been considered of utmost importance for emission modeling and engine control issues. The *Fuel Consumption*, *MAP*, λ and *NOx* are part of the most indicative engine variables and have stimulated interest in this work. In fact, Artificial Neural Network (ANN) models for predicting the aforementioned quantities could replace the real sensors, which sometimes might be unable to perform direct measurements or to provide with trustworthy values, and on top of that could be quite expensive. These models are known as the neural network-based virtual sensors which could be implemented on-board and, through strong generalization performance, provide results with high accuracy. For this work, the Feed-Forward Neural Network (FFNN) and Time-Delay Neural Network (TDNN) architectures are investigated for the predictive models, and their results are accordingly compared. The networks are trained, validated and tested using experimental data collected from various trials on the laboratory test-bed. In addition, a fully parametric study have been conducted concerning the models inputs selection. The latter has been based on: the theoretical background of the engine function, the relationship between various engine parameters and the available quantities measured by real sensors. Of course, with regard to the acquired experience through modeling, both the inputs and the calculation mechanisms of the models were revised until achieving the most efficient performance. After that, the models were tested on data sets within the same range of the training set (i.e. the whole envelope of test-bed) and on completely unknown data, with different pattern and scaling. Both types of deep neural network models performed well, with no appreciable errors. For the whole modeling process the Python language was used and mainly the Keras library (interface of Tensorflow 2.0), except for the Data Preparation process which was completed in the MATLAB environment.

Contents

1 Introduction **11**
1.1 Problem Framework 11
1.2 Literature Review 12
1.3 Objective/Structure of Thesis 13

2 Neural Networks & Deep Learning **15**
2.1 Definition and Use of Neural Networks 15
2.2 Types of Models in Machine Learning 16
2.3 Linear Basis Function Models 16
2.3.1 Limitations of Fixed Basis Functions 17
2.4 Feed-Forward Neural Networks 18
2.4.1 Network Training 20
2.4.1.1 Parameter Optimization 22
2.4.2 Gradient descent optimization 23
2.4.3 Error Backpropagation 24
2.4.3.1 Evaluation of error-function derivatives 24
2.4.4 Basic Structure of Feed-Forward Neural Networks 25
2.5 Time-Delay Neural Networks 27

3 Diesel Engine Operation Characteristics **29**
3.1 Engine Performance 29
3.1.1 Fuel Consumption 29
3.1.2 Intake Manifold Absolute Pressure (MAP) 30
3.1.3 Air-fuel ratio equivalence factor - Lambda (λ) 30
3.1.4 Nitrogen Oxides (NOx) 31
3.1.4.1 Formation of NOx 31
3.1.4.2 NOx Regulations 32
3.1.5 Exhaust Gas Recirculation (EGR) System 32

4 Data Preparation **35**

4.1 Data Collection 35

4.2 Data Overview 37

4.3 Data Pre-Processing 40

4.3.1 Data Synchronization 40

4.3.2 Data Re-sampling 43

5 Model Design **45**

5.1 Model Inputs Selection 45

5.2 Training, Validation & Testing Datasets 47

5.3 Data Normalization 48

5.4 Fine-Tuning of Model Hyperparameters 48

5.4.1 Number of Hidden Layers 49

5.4.2 Number of Neurons per Hidden Layer 49

5.4.3 Activation Function 50

5.4.3.1 Sigmoid Function 50

5.4.3.2 ReLU (Rectified Linear Unit) 51

5.4.4 Optimizer 51

5.4.4.1 RMSprop 52

5.4.4.2 Adam (Adaptive Moment Estimation) 53

5.4.4.3 Adamax 54

5.4.5 Metrics 54

5.4.6 Early Stopping Callback 55

6 Training & Testing Results **57**

6.1 Procedure Flowchart 57

6.2 Fuel Consumption Models 59

6.2.1 FFNN Model 59

6.2.1.1 Training Results 59

6.2.1.2 Testing Results 67

6.2.2 TDNN Model 69

6.2.2.1 Training Results 69

6.2.2.2 Testing Results & Comparison with FFNN Performance . . 74

6.3 MAP Models 77

6.3.1 FFNN Model 77

6.3.1.1 Training Results 77

6.3.1.2 Testing Results 82

6.3.2 TDNN Model 84
6.3.2.1 Training Results 84
6.3.2.2 Testing Results & Comparison with FFNN Performance . . 87
6.4 Lambda (λ) Models 89
6.4.1 FFNN Model 89
6.4.1.1 Training Results 89
6.4.1.2 Testing Results 95
6.4.2 TDNN Model 97
6.4.2.1 Training Results 97
6.4.2.2 Testing Results & Comparison with FFNN Performance . . 101
6.5 NOx Models 104
6.5.1 FFNN Model 104
6.5.1.1 Training Results 104
6.5.1.2 Testing Results 109
6.5.2 TDNN Model 111
6.5.2.1 Training Results 111
6.5.2.2 Testing Results & Comparison with FFNN Performance . . 115

7 Conclusions and Future Work 117
7.1 Conclusions 117
7.2 Suggestions for Future Work 117

Bibliography 119

Chapter 1

Introduction

1.1 Problem Framework

As years pass by, both the Maritime and Engineering fields have put great effort into finding ways to achieve the ”optimum” diesel engine performance. Specifically, they have been always wanted to reach a level where the diesel engine works economically and effectively at the same time. These issues and thoughts are also included in the scope of many universities and research facilities, which try to find new tools, methods and approaches, in order to solve current problems, answer unanswerable questions and make progress in their field. Nowadays, also, an extra parameter has raised scientific, technological and industrial attention, and it concerns the reduction of diesel engine emissions to the lowest possible limit.

For these reasons, more and more researchers have invented new methods and ways to improve power train performance, fuel and after-treatment efficiency, especially by revising the existed control and diagnostic strategies. The most common system for defining the engine performance is the Electrical Control Unit (ECU) - an integrated element of the intelligent engine. It controls a series of actuators by reading values from a multitude of sensors within the engine bay and by interpreting the data using multidimensional performance maps. In more detail, the ECU’s function is to control the action of the following components and systems:

- The engine speed in accordance with a reference value from the application control system (an integrated governor control).
- Engine protection (overload protection as well as faults). Optimization of combustion to suit the running condition.
- Start, stop and reversing sequencing of the engine.
- Hydraulic (servo) oil supply (lube oil).
- Auxiliary blowers and turbocharging.[1]

Concisely, the ECU system collects, analyzes, processes and executes the data it receives from various sub-systems, particularly sensors. Therefore, it mainly controls actuators, ignition timing, variable valve timing and the like. The ECU system is the central processing unit of the Engine Management System (EMS). The latter also includes different engine

sensors and actuators located at various positions in the engine bay. The whole operation of this system aims at the optimum engine performance, emissions handling and fuel economy.

However, there should be an alternative in order to reduce the hardware complexity of the aforementioned system (EMS) and at the same time provide with useful information in cases where direct measurement of the signal is not possible (e.g. in-cylinder quantities) or whenever the available sensor might not ensure the required sensing characteristics (i.e. accuracy, dynamic performance).[2] This alternative could be the on-board implementation of dedicated models, either as predictors or embedded in model-based controllers or in diagnostic schemes. Virtual Sensors (VSs) can be used as substitute of real sensors providing useful information about the actual process. Actually, VSs must be designed in such a way as to simulate time-dependent processes and guarantee accuracy, stability and short computational time.[3] An intelligence system could help researchers to remove disadvantages when determining the engine performance parameters and emissions, especially where these characteristics seem to be complex and uncertain. In fact, Machine Learning (ML) and Artificial Neural Networks (ANN) can solve complex and non-linear problems, by exploring the relations between various elements and data. A major practical advantage of an ANN is its ability of learning by example, as it can solve problems even without having sufficient amount of precise and complete input data. Consequently, ANN models -used as real-time sensors- could be implemented both for emission modeling and engine control issues, especially in some areas where the conventional modeling methods fail.[4]

1.2 Literature Review

As aforementioned, researchers' main scope is to find out the most efficient and easily applicable way to implement neural-network based sensors, especially for emissions simulation. Before starting the model design, it is of the utmost importance to lay the groundwork by understanding the basic theory of artificial intelligence, by the aid of [5], and then to learn how to apply it in practice, as [6] proposes. With respect to the network type, various perspectives have been applied, with the majority of the applications to use architectures based on dynamic neural networks, such as the Recurrent Neural Networks (RNN), as in [3], or the Time Delay Neural Networks (TDNN), as in [7]. Specifically, in [3], RNN models have been used to predict NO emission during SI engine transient maneuvers, with an estimation error lower than 4%, even in presence of step load transients and Air-Fuel Ratio spikes. Also, in [7], multi-layer perception static and TDNN models have been investigated, motivated by the development of economic model predictive control (eMPC) strategies. Moreover, in some approaches there is a large variety of inputs to choose from; for instance, in [7], inputs such as *main injection timing*, *intake temperature* or *intake O_2 concentration*. On the other hand, some other models are based only in the most important and available quantities for predicting the output, as in the case of [8], where a physical analysis was performed to choose the most significant inputs. In addition, in [8] the best results have been provided by the NN with input derivative and the TDNN. This assumption was based on the generalization ability of the model, which is a fundamental requirement for ensuring the robustness respect to engine production spread and aging. Furthermore, an other interesting approach is that of [2], where the least square technique features were exploited to adapt the output of a RNN-based NOx virtual sensor, in such a way as to account for the effects due to engine aging and production spread as well as components drift. The prediction error was almost always below 15%, which corresponds to a satisfactory accuracy for the intended application. Finally, all the above perspectives and considerations have been taken into account during the whole process of the model design in this thesis, but,

of course, some alternative ideas and attempts have been applied.

1.3 Objective/Structure of Thesis

In this thesis, two deep neural network model structures are investigated, not only for emission modeling, but also for engine control issues of the Hybrid Integrated Propulsion Powertrain 2 (HIPPO-2) test-bed of Laboratory of Marine Engineering (LME). The modeled-predicted quantities are: **Fuel Consumption (lt/h)**, **Intake Manifold Absolute Pressure - MAP (kPa)**, **Air-Fuel ratio** λ and **NOx emissions (ppm)**. For their predictions two different types of neural network models have been used: the **Feed-Forward Neural Network (FFNN)** and the **Time-Delay Neural Network (TDNN)**, and their results have been compared accordingly. Of course, these models have been designed with the prospect of a real-time implementation in the test-bed environment. This thesis contains both the theoretical background and the practical application, and its structure is the following, In Chapter 2, the classical theory concerning the artificial neural networks is presented. In Chapter 3, the experimental setup and the various engine parameters to be predicted are described. In Chapter 4, the experimental data are presented together with their pre-processing techniques. After the Data Preparation section, the Model Design follows in Chapter5, which includes the selection of the model inputs, from an engineering angle, and of the network hyperparameters. Then, the training process and the double testing of both FFNN and TDNN models for all the target quantities are presented in Chapter 6, with all the necessary result figures. Finally, the conclusions of this work and further ideas on this subject are included in Chapter 7.

Chapter 2

Neural Networks & Deep Learning

2.1 Definition and Use of Neural Networks

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature.[9] In fact, neural networks can adapt to changing inputs; in order to generate the best possible result without needing to redesign the output criteria. Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements (neurons) working in parallel to solve a specific problem.

The simplest definition of a neural network, more properly referred to as an 'artificial neural network' (ANN), was provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defined a neural network as: "...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs.".[10]

Neural networks have been mainly used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques, because of their remarkable ability to derive meaning from complicated or imprecise data. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. This expert can then be used to provide projections given new situations of interest and answer "what if" questions.[11]

A quite impressive fact about Neural Networks is that they learn by example, meaning they cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The main disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable. However, its extended and profitable use in the majority of scientific, technological or business fields nowadays overcomes any "weakness".

2.2 Types of Models in Machine Learning

The majority of Machine Learning (ML) problems are mostly divided into the following categories:

- **Supervised Learning**: A type of ML where the model is provided with labeled training data. The learning algorithm tries to model relationships and dependencies between the target prediction output and the input labeled features. In that way, the model can predict the output values for new and completely unknown data, based on the relationships it learned. There are two main subcategories of supervised learning:

    1. **Regression Models**: In this kind of modeling, continuous values are predicted, e.g. an integer or floating point value. In this thesis, the developed neural networks are considered a subcategory of regression modeling, because of the continuous quantities as targets.
    2. **Classification Models**: In this kind of modeling, discrete values are predicted. The output variables are often called "labels" or "categories" and the mapping function predicts the class or category for a given observation.

- **Unsupervised Learning**: Its main goal is to identify meaningful patterns in the data. To accomplish this, the machine must learn from an unlabeled data set. In other words, the model has no hints how to categorize each piece of data and must infer its own rules for doing so.

2.3 Linear Basis Function Models

Before presenting the fundamentals of the different neural networks types, a description of the *Linear Basis Function Models* should be included. Despite the fact that in this study there is not always a linear connection between the various input and output data sets, it would be quite instructive and clarifying to write a few things about the simplest regression model, the **Linear Regression Model**. Although linear models have significant limitations as practical techniques for pattern recognition, particularly for problems involving input spaces of high dimensionality, they have nice analytical properties and form the foundation for more sophisticated models to be discussed in later chapters.

Given a training data set comprising N observations $\{x_n\}$, where $n = 1, ..., N$, together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of $\mathbf{x}$. In the simplest approach, this can be done by directly constructing an appropriate function $y(\mathbf{x})$ whose values for new inputs $\mathbf{x}$ constitute the predictions for the corresponding values of t. More generally, from a probabilistic perspective, the aim is to model the predictive distribution $p(t|\mathbf{x})$ because this expresses the uncertainty about the value of t for each value of $\mathbf{x}$. From this conditional distribution predictions of t can be made, for any new value of $\mathbf{x}$, in such a way as to minimize the expected value of a suitably chosen loss function. One of the most common choices of loss function for real-valued variables is the Squared Loss, $L(t, y(\mathbf{x})) = \{y(\mathbf{x}) - t\}^2$, for which the optimal solution is given by the conditional expectation of t.

The simplest linear model for regression is one that involves a linear combination of the input variables

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + ... + w_D x_D \tag{2.1}$$

where $x = (x_1, ..., x_D)^T$. This is often simply known as *linear regression.* The key property of this model is that it is a linear function of the parameters $w_0, ..., w_D$. It is also, however, a linear function of the input variables x_i , and this imposes significant limitations on the model. The class of models is therefore extended by considering linear combinations of fixed nonlinear functions of the input variables, of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}) \tag{2.2}$$

where $\phi_j(\mathbf{x})$ are known as *basis functions.* By denoting the maximum value of the index j by *M-1*, the total number of parameters in this model will be *M.*

The parameter w_0 allows for any fixed offset in the data and is sometimes called a *bias* parameter. It is often convenient to define an additional dummy 'basis function' $\phi_0(\mathbf{x}) = 1$ so that

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \tag{2.3}$$

where $\mathbf{w} = (w_0, ..., w_{M-1})^T$ and $\phi = (\phi_0, ..., \phi_{M-1})^T$.

By using nonlinear basis functions, we allow the function $y(\mathbf{x}, \mathbf{w})$ to be a non-linear function of the input vector $\mathbf{x}$. Functions of the form 2.2 are called linear models, however, because this function is linear in $\mathbf{w}$. It is this linearity in the parameters that will greatly simplify the analysis of this class of models. The example of polynomial regression is a particular example of this model in which there is a single input variable x, and the basis functions take the form of powers of x so that $\phi_j(x) = x^j$. One limitation of polynomial basis functions is that they are global functions of the input variable, so that changes in one region of input space affect all other regions. This can be resolved by dividing the input space up into regions and fit a different polynomial in each region, leading to *spline functions.*[12]

There are many other possible choices for the basis functions, for example the *sigmoidal basis function* of the form

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right) \tag{2.4}$$

where $\sigma(\alpha)$ is the *logistic sigmoid function* defined by

$$\sigma(a) = \frac{1}{1 + exp(-a)} \tag{2.5}$$

Equivalently, we can use the 'tanh' function because this is related to the logistic sigmoid by $\tanh(a) = 2\sigma(a) - 1$, and so a general linear combination of logistic sigmoid functions is equivalent to a general linear combination of 'tanh' functions.

2.3.1 Limitations of Fixed Basis Functions

Throughout this chapter, the focus has been on models comprising a linear combination of fixed, nonlinear basis functions. It has also been seen that the assumption of linearity in the parameters led to a range of useful properties including closed-form solutions to the least-squares problem. Furthermore, for a suitable choice of basis functions, arbitrary non-linearities can be modeled in the mapping from input variables to targets. It might appear,

therefore, that such linear models constitute a general purpose framework for solving problems in pattern recognition. Unfortunately, there are some significant shortcomings with linear models, which will cause study to turn to more complex models such as support vector machines and neural networks. The difficulty stems from the assumption that the basis functions $\phi_j(\mathbf{x})$ are fixed before the training data set is observed and is a manifestation of the curse of dimensionality. As a consequence, the number of basis functions needs to grow rapidly, often exponentially, with the dimensionality D of the input space.

Fortunately, there are two properties of real data sets that can be exploited to help alleviate this problem. First of all, the data vectors $\{\mathbf{x}_n\}$ typically lie close to a nonlinear manifold whose intrinsic dimensionality is smaller than that of the input space as a result of strong correlations between the input variables. Neural network models, which use adaptive basis functions having sigmoidal nonlinearities, can adapt the parameters so that the regions of input space over which the basis functions vary corresponds to the data manifold. The second property is that target variables may have significant dependence on only a small number of possible directions within the data manifold. Neural networks can exploit this property by choosing the directions in input space to which the basis functions respond.

2.4 Feed-Forward Neural Networks

The linear models for regression, as aforementioned, are based on linear combinations of fixed nonlinear basis functions $\varphi_j(\mathbf{x})$ and take the form

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) \tag{2.6}$$

where $f(\cdot)$ is the identity in the case of regression. The main goal is to extend this model by making the basis functions $\varphi_j(\mathbf{x})$ depend on parameters and then to allow these parameters to be adjusted, along with the coefficients $\{w_j\}$, during training. There are, of course, many ways to construct parametric nonlinear basis functions. Neural networks use basis functions that follow the same form as (2.6), so that each basis function is itself a nonlinear function of a linear combination of the inputs, where the coefficients in the linear combination are adaptive parameters.

This leads to the basic neural network model, which can be described a series of functional transformations. First M linear combinations of the input variables $x_1, ..., x_D$ are constructed in the form

$$\alpha_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.7}$$

where $j = 1, ..., M$, and the superscript (1) indicate that the corresponding parameters are in the first 'layer' of the network. The parameters $w_{ji}^{(1)}$ shall be referred to as *weights* and the parameters $w_{j0}^{(1)}$ as *biases*. The quantities a_j are known as *activations*. Each of them is then transformed using a differentiable, nonlinear *activation function h(·)* to give

$$z_j = h(\alpha_j). \tag{2.8}$$

These quantities correspond to the outputs of the basis functions in (2.6) that, in the context of neural networks, are called *hidden units*. The nonlinear functions *h(·)* are generally chosen to be sigmoidal functions such as the logistic sigmoid or the 'tanh' function.

Following (2.6), these values are again linearly combined to give *output unit activations*

$$\alpha_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.9}$$

where $k = 1, ..., K$, and K is the total number of outputs. This transformation corresponds to the second layer of the network, and again the $w_{k0}^{(2)}$ are bias parameters. Finally, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k. The choice of activation function is determined by the nature of the data and the assumed distribution of target variables and follows the same considerations as for linear models. Thus for standard regression problems, the activation function is the identity so that $y_k = a_k$.

All these various stages van be combined in order to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=1}^{M} w_{kj}^{(2)} h \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{2.10}$$

where the set of all weight and bias parameters have been grouped together into a vector $\mathbf{w}$. Thus the neural network model is simply a nonlinear function from a set of input variables $\{x_i\}$ to a set of output variables $\{y_k\}$ controlled by a vector $\mathbf{w}$ of adjustable parameters.

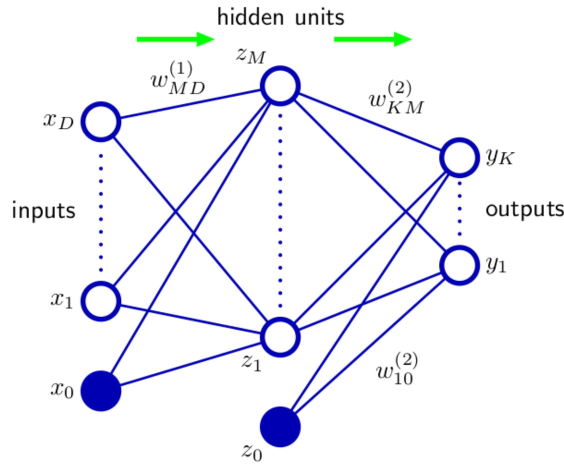


Figure 2.1: Network diagram for the two-layer neural network corresponding to (2.10). The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links inputs coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation.

This function can be represented in the form of a network diagram as shown in Figure 2.1. The process of evaluating (2.10) can then be interpreted as a *forward propagation* of information through the network. It should be emphasized that these diagrams do not represent probabilistic graphical models because the internal nodes represent deterministic variables rather than stochastic ones.

As previously discussed, the bias parameters in (2.7) can be absorbed into the set of weight parameters by defining an additional input variable x_0 whose value is clamped at

$x_0 = 1$, so that (2.7) takes the form

$$\alpha_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i. \tag{2.11}$$

The second-layer biases can be similarly absorbed into the second-layer weights, so that the overall network function becomes

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma \left(\sum_{j=0}^{M} w_{kj}^{(2)} h \left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i \right) \right). \tag{2.12}$$

As can be seen from Figure 2.1, the neural network model comprises two stages of processing, and the neural network is also known as the *multilayer percepton*, or MLP. A key difference compared to the perceptron, however, is that the neural network uses continuous sigmoidal nonlinearities in the hidden units, whereas the perceptron uses step-function nonlinearities. This means that the neural network function is differentiable with respect to the network parameters, and this property will play a central role in network training.

If the activation functions of all the hidden units in a network are taken to be linear, then for any such network an equivalent network without hidden units can always be found. This follows from the fact that the composition of successive linear transformations is itself a linear transformation. However, if the number of hidden units is smaller than either the number of input or output units, then the transformations that the network can generate are not the most general possible linear transformations from inputs to outputs because information is lost in the dimensionality reduction at the hidden units. In general, however, there is little interest in multilayer networks of linear units.

The network architecture shown in Figure 2.1 is the most commonly used one in practice. However, it is easily generalized, for instance by considering additional layers of processing each consisting of a weighted linear combination of the form (2.9) followed by an element-wise transformation using a nonlinear activation function. Note that there is some confusion in the literature regarding the terminology for counting the number of layers in such networks. Thus the network in Figure 2.1 may be described as a 3-layer network (which counts the number of layers of units, and treats the inputs as units) or sometimes as a "single-hidden-layer" network (which counts the number of layers of hidden units). In this thesis, the latter terminology will be kept in the development of all NN models.

2.4.1 Network Training

So far, the neural networks have been viewed as a general class of parametric nonlinear functions from a vector $\mathbf{x}$ of input variables to a vector $\mathbf{y}$ of output variables. A simple approach to the problem of determining the network parameters is to minimize a sum-of-squares error function. Given a training set comprising a set of input vectors $\{\mathbf{x}_n\}$, where $n = 1, ..., N$, together with a corresponding set of target vectors $\{\mathbf{t}_n\}$, the error function is minimized

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2. \tag{2.13}$$

However, a much more general view of network training could be provided by first giving a probabilistic interpretation to the network outputs. Here, the use of probabilistic predic-

tions will provide with a clearer motivation both for the choice of output unit nonlinearity and the choice of error function.

Starting with regression problems, and for the moment considering a single target variable t that can take any real value, it is assumed that t has a Gaussian distribution with a $\mathbf{x}$-dependent mean, which is given by the output of the neural network, so that

$$p(t|\mathbf{x}, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \tag{2.14}$$

where β is the precision (inverse variance) of the Gaussian noise. For the conditional distribution given by (2.14), it is sufficient to take the output unit activation function to be the identity, because such a network can approximate any continuous function from $\mathbf{x}$ to y. Given a data set of N independent, identically distributed observations $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$, along with corresponding target values $\mathbf{t} = \{t_1, ..., t_N\}$, the corresponding likelihood function can be constructed

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{w}, \beta). \tag{2.15}$$

Taking the negative algorithm, the error function is obtained

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{2.16}$$

which can be used to learn the parameters $\mathbf{w}$ and β. Note that in the neural networks literature, it is usual to consider the minimization of an error function rather than the maximization of the (log) likelihood, and so here this convention should be followed. Consider first the determination of $\mathbf{w}$. Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \tag{2.17}$$

where additive and multiplicative constants have been discarded. The value of $\mathbf{w}$ found by minimizing $E(\mathbf{w})$ will be denoted $\mathbf{w}_{ML}$ because it corresponds to the maximum likelihood solution.

Having found $\mathbf{w}_{ML}$, the value of β can be found by minimizing the negative log likelihood to give

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}_{ML}) - t_n\}^2. \tag{2.18}$$

Note that this can be evaluated once the iterative optimization required to find $\mathbf{w}_{ML}$ is completed.

There is a natural pairing of the error function (given by the negative log likelihood) and the output unit activation function. In the regression case, the network can be seen as having an output activation function that is the identity, so that $y_k = a_k$. The corresponding sum-of-squares error function has the property

$$\frac{\vartheta E}{\vartheta \alpha_k} = y_k - t_k \tag{2.19}$$

which should be used when discussing *error backpropagation*.

2.4.1.1 Parameter Optimization

The next task is that of finding a weight vector $\mathbf{w}$ which minimizes the chosen function $E(\mathbf{w})$. At this point, it is useful to have a geometrical picture of the error function, which can be seen as a surface sitting over weight space as shown in Figure 2.2 below.

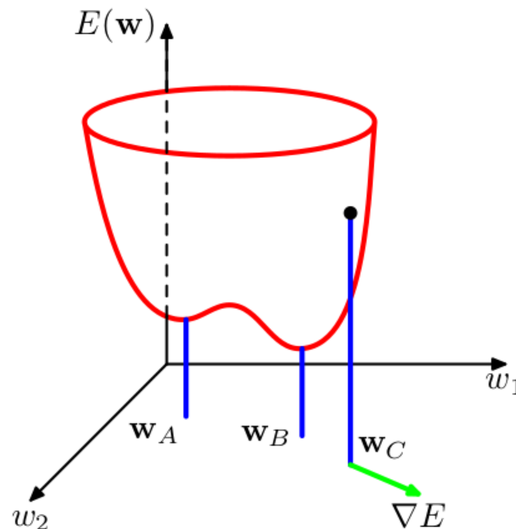


Figure 2.2: Geometrical view of the error function $E(\mathbf{w})$ as a surface sitting over weight space. Point $\mathbf{w}_A$ is a local minimum and $\mathbf{w}_B$ is the global minimum. At any point $\mathbf{w}_C$, the local gradient of the error surface is given by the vector ∇E.

First note that if a small step in weight space is made from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$ then the change in the error function is $\delta E \simeq \delta\mathbf{w}^T \nabla E(\mathbf{w})$, where the vector $\nabla E(\mathbf{w})$ points in the direction of greatest rate of increase of the error function. Because the error $E(\mathbf{w})$ is a smooth continuous function of $\mathbf{w}$, its smallest value will occur at a point in weight space such that the gradient of the error function vanishes, so that

$$\nabla E(\mathbf{w}) = 0 \tag{2.20}$$

as otherwise a small step could be made in the direction of $-\nabla E(\mathbf{w})$ and thereby further reduce the error. Points at which the gradient vanishes are called *stationary points*, and may be further classified into *minima*, *maxima*, and *saddle points*.

The main goal is to find a vector $\mathbf{w}$ such that $E(\mathbf{w})$ takes its smallest value. However, the error function typically has a highly nonlinear dependence on the weights and bias parameters, and so there will be many points in weight space at which the gradient vanishes (or is numerically very small). Indeed, for any point $\mathbf{w}$ that is a local minimum, there will be other points in weight space that are equivalent minima. For instance, in a two-layer network of the kind shown in Figure 2.1, with M hidden units, each point in weight space is a member of a family of $M!2^M$ equivalent points.

Furthermore, there will typically be multiple inequivalent stationary points and in particular multiple inequivalent minima. A minimum that corresponds to the smallest value of the error function for any weight vector is said to be a *global minimum*. Any other minima corresponding to higher values of the error function are said to be *local minima*.

For a successful application of neural networks, it may not be necessary to find the global minimum (and in general it will not be known whether the global minimum has been found) but it may be necessary to compare several local minima in order to find a sufficiently good solution.

Because there is clearly no hope of finding an analytical solution to the equation $\nabla E(\mathbf{w}) = 0$ a good alternative could be the iterative numerical procedures. The optimization of continuous nonlinear functions is a widely studied problem and there exists an extensive literature on how to solve it efficiently. Most techniques involve choosing some initial value $\mathbf{w}^{(0)}$ for the weight vector and then moving through weight space in a succession of steps of the form

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \delta\mathbf{w}^{(\tau)} \tag{2.21}$$

where τ labels the iteration step. Different algorithms involve different choices for the weight vector update $\delta\mathbf{w}^{(\tau)}$. Many algorithms make use of gradient information and therefore require that, after each update, the value of $\nabla E(\mathbf{w})$ is evaluated at the new weight vector $\mathbf{w}^{(\tau+1)}$.

2.4.2 Gradient descent optimization

The simplest approach to using gradient information is to choose the weight update in (2.21) to comprise a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)}) \tag{2.22}$$

where the parameter $\eta > 0$ is known as the *learning rate.* After each such update, the gradient is re-evaluated for the new weight vector and the process repeated. Note that the error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate ∇E. Techniques that use the whole data set at once are called *batch* methods. At each step the weight vector is moved in the direction of the greatest rate of decrease of the error function, and so this approach is known as *gradient descent* or *steepest descent.*

In order to find a sufficiently good minimum, it may be necessary to run a gradient-based algorithm multiple times, each time using a different randomly chosen starting point, and comparing the resulting performance on an independent validation set.

There is, however, an on-line version of gradient descent that has proved useful in practice for training neural networks on large data sets.[13] Error functions based on maximum likelihood for a set of independent observations comprise a sum of terms, one for each data point

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{2.23}$$

On-line gradient descent, also known as *sequential gradient descent* or *stochastic gradient descent*, makes an update to the weight vector based on one data point at a time, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n(\mathbf{w}^{(\tau)}) \tag{2.24}$$

This update is repeated by cycling through the data either in sequence or by selecting points at random with replacement. There are of course intermediate scenarios in which the updates are based on batches of data points.

2.4.3 Error Backpropagation

The main goal in this section is to find an efficient technique for evaluating the gradient of an error function $E(\mathbf{w})$ for a feed-forward neural network. It is apparent that this can be achieved using a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as *error backpropagation*, or sometimes simply as *backprop*.

It should be noted that the term "backpropagation" is used in the neural computing literature to mean a variety of different things. For instance, the multilayer perceptron architecture is sometimes called a "backpropagation network". The term "backpropagation" is also used to describe the training of a multilayer perceptron using gradient descent applied to a sum-of-squares error function. In order to clarify the terminology, it is useful to consider the nature of the training process more carefully. Most training algorithms involve an iterative procedure for minimization of an error function, with adjustments to the weights being made in a sequence of steps. At each such step, two distinct stages can be distinguished. In the first stage, the derivatives of the error function with respect to the weights must be evaluated. As it is evident, the important contribution of the backpropagation technique is in providing a computationally efficient method for evaluating such derivatives. Because it is at this stage that errors are propagated backwards through the network, the term backpropagation should be used specifically to describe the evaluation of derivatives. In the second stage, the derivatives are then used to compute the adjustments to be made to the weights. It is important to recognize that the two stages are distinct. Thus, the first stage, namely the propagation of errors backwards through the network in order to evaluate derivatives, can be applied to many other kinds of network and not just the multilayer perceptron. Similarly, the second stage of weight adjustment using the calculated derivatives can be tackled using a variety of optimization schemes, many of which are substantially more powerful than simple gradient descent.

2.4.3.1 Evaluation of error-function derivatives

In this section, the backpropagation algorithm is derived for a general network having arbitrary feed-forward topology, arbitrary differentiable nonlinear activation functions, and a broad class of error function. The resulting formulas will then be illustrated using a simple layered network structure having a single layer of sigmoidal hidden units together with a sum-of-squares error.

Many error functions of practical interest, for instance those defined by maximum likelihood for a set of independent and identically distributed (i.i.d.) data, comprise a sum of terms, one for each data point in the training set, so that

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{2.25}$$

Here, the problem of evaluating $\nabla E_n(\mathbf{w})$ should be considered for one such term in the error function. This may be used directly for sequential optimization, or the results can be accumulated over the training set in the case of batch methods.

Consider first a simple linear model in which the outputs y_k are linear combination of the input variables x_i so that

$$y_k = \sum_{i} w_{ki} x_i \tag{2.26}$$

together with an error function that, for a particular input pattern n, takes the form

$$E_n = \frac{1}{2} \sum k(y_{nk} - t_{nk})^2 \tag{2.27}$$

where $y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$. The gradient of this error function with respect to a weight w_{ji} is given by

$$\frac{\vartheta E_n}{\vartheta w_{ji}} = (y_{nj} - t_{nj})x_{ni} \tag{2.28}$$

which can be interpreted as a "local" computation involving the product of an "error signal" $y_{nj} - t_{nj}$ associated with the output end of the link w_{ji} and the variable x_{ni} associated with the input end of the link.[5]

Finally, the foundation for the comprehension of the mathematical background in neural networks has been successfully laid. Starting from a simple linear regression model, the aforementioned method-process ended up evaluating the error-function derivative for the gradient descent optimization. However, when designing and then constructing a neural network in real time the theoretic way does not always coincide with the practical one. For this reason, in the next section a more practical and simplified approach of a neural network's structure is presented.

2.4.4 Basic Structure of Feed-Forward Neural Networks

The basic structure of an artificial neural network consists of *neurons* that are grouped into *layers*. The most common and typical neural network structure consists of an *input* layer, one or more *hidden* layers and an *output* layer, as in Figure 2.3.

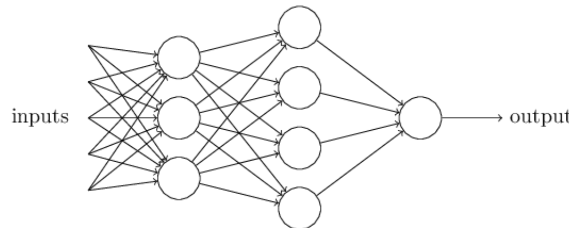


Figure 2.3: Typical Structure of a Multi-Layer Neural Network

Each "circle" of Figure 2.3 represents an artificial neuron, often called a *node*, which basically receives input from some other nodes, or from an external source, and then computes an output. In feed-forward neural networks, each of the neurons is fully connected to nodes on the next layer. A typical neuron has the following structure.

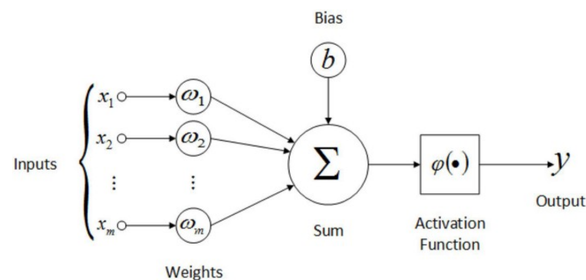


Figure 2.4: Typical Structure of an Artificial Neuron

Starting with the Input Neurons, they represent the number of features that the neural network uses to make its predictions. The input vector contains one input neuron per feature, and in regression models each dataset value represents an input neuron. Then, there are the Hidden Layers and Neurons per Hidden Layers -an important part of the model hyperparameters- whose number depends on the complexity of the model. Finally, the connections and calculations between the aforementioned neurons end up to the Output Neuron, which is the prediction of the model and therefore its target.

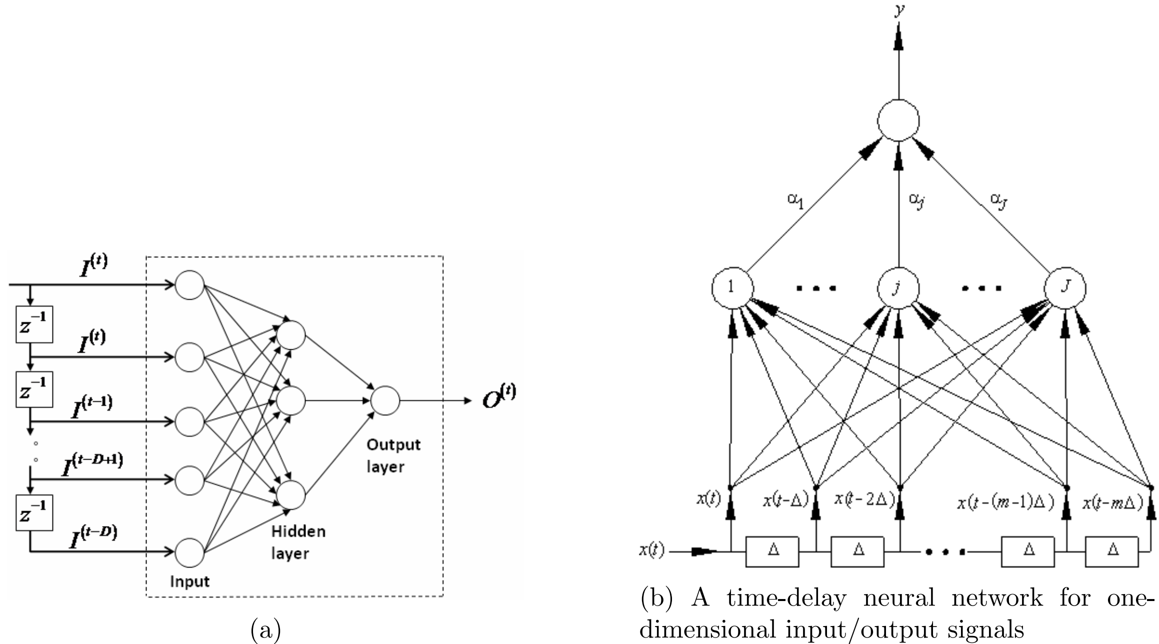
Every connection between two neurons in adjoining layers has a *weight* attached, which activates the neuron when having a positive value, or inhibiting it when being negative. In fact, a weight can be thought of as the "strength" of the connection. It is the parameter that transforms the input data within the network hidden layers, by being multiplied by its associated signal on the connection. For instance, in Figure 2.4 the inputs $\{x_1, x_2, ..., x_i\}$ are connected to neuron j with weights $\{w_{1j}, w_{2j}, ..., w_{ij}\}$ on each connection. After that, the neuron sums all the multiplication values it receives, adding the *bias* values. The latter represents how diverged the predictions are from their intended value, and provides every node with a trainable constant value. Weights and biases are both learnable parameters inside the network. A teachable neural network will randomize both the weight and bias values before learning initially begins. As training continues, both parameters are adjusted toward the desired values and the correct output.

The output of the sum function is then passed through a transfer function, known as an *activation function* $\varphi(\cdot)$. Its purpose is both to decide whether a neuron should be activated and to introduce *non-linearity* into the output of a neuron, especially in complex models. It is worthwhile mentioning that in regression problems there is no need of activation function for the output neuron. In other words, the last hidden layer has a linear activation function (i.e. no activation at all) in order to produce the predicted value, without limitations.

The main objective of a neural network is to predict the intended output value having the smallest error or conversely the largest accuracy. In order to achieve the optimum result, the weights and biases of the neurons should be updated according to the error at the output. The process of updating these values is known as *back-propagation*, which represents one of the several ways in which an artificial neural network can be trained. The back-propagation algorithm is a supervised training scheme, meaning that *"it learns from mistakes"*. Specifically, at first, all the weights and biases are randomly assigned. For every input in the training data set, the network is activated and its output is observed. This output is compared with the intended final value and the error between them is "propagated" back to the previous layer. Then an appropriate *optimization method* is used in order to "adjust" accordingly the weights and biases, with an aim of reducing the error at the output layer. This process is repeated until the error is minimized or is below a predetermined threshold, and consequently an optimum output result has been achieved. The number of iterations is usually predefined via the number of *epochs*, which sets how many times the training session will be completed.

2.5 Time-Delay Neural Networks

Time delay neural network (TDNN) is a multilayer artificial neural network which has a "memory" at the level of inputs. Its architecture is based on *dynamic* neural networks, whereas the aforementioned feed-forward neural networks are *static*. In TDNNs both input data sets and their delayed versions are included in the input layer. The output depends both on the current value and on values at previous time instants of inputs.In particular, the neural network evaluates the current output from "history" of inputs and its typical structure is shown in Figures 2.5a and 2.5b.



(a)

(b) A time-delay neural network for one-dimensional input/output signals

Figure 2.5: Typical Structures of Time-Delay Neural Networks

Consider the time-delay neural network architecture shown in Figure 2.5b. This maps a finite time sequence $\{x(t), x(t-\Delta), x(t-d\Delta), ..., x(t-m\Delta)\}$ into a single output y (this can also be generalized for the case when x and/or y are vectors). One may view this neural network as a discrete-time nonlinear filter.

The architecture in Figure 2.5b is equivalent to a single hidden layer feed-forward neural network receiving the $(m+1)$-dimensional "spatial" pattern $\mathbf{x}$ generated by a tapped delay line preprocessor from a temporal sequence. Thus, if target values for the output unit are specified for various times t, back propagation may be used to train the above network to act as a sequence recognizer.

Despite the fact that the time-delay neural network has been successfully applied to the problem of speech recognition, here the issue of time series prediction is discussed, since it captures the spirit of the type of processing done by the time-delay neural net. Given observed values of the state x of a non-linear, dynamical system at discrete times less than t, the goal is to use these values to accurately predict $x(t+p)$, where p is some prediction time step into the future (for simplicity, a one dimensional state x is assumed). Clearly, as p increases the quality of the predicted value will degrade for any predictive method. A method is robust if it can maintain prediction accuracy for a wide range of p values.

As is normally done in linear signal processing applications, one may use the tapped

delay line nonlinear filter of Figure 2.5b as the basis for predicting $x(t+p)$. Here, a training set is constructed of pairs $\{\mathbf{x}^k, x(t_k+p)\}$, where $\mathbf{x}^k = [x(t_k), x(t_k-\Delta), x(t_k-2\Delta), ..., x(k_t-m\Delta)]^T$. Back propagation may now be employed to learn such a training set. Reported simulation results of this prediction method show comparable or better performance compared to other non neural network-based techniques.

Theoretical justification for the above approach is available in the form of a very powerful theorem by Takens (1981) [14], which states that there exists a functional relation of the form

$$x(t+p) = g[x(t), x(t-\Delta), ..., x(t-m\Delta)] \tag{2.29}$$

with $d < m < 2d$, as long as the trajectory $x(t)$ evolves towards compact attracting manifolds of dimension d. This theorem, however, provides no information on the form of g or the value of it. The time-delay neural network approach provides a robust approximation for g in (2.29) in the form of the continuous, adaptive parameter model

$$y = \sum_{j=1}^{J} a_j f_h\Big(\sum_{i=1}^{m+1} w_{ji} x(t-(i-1)\Delta)\Big) \tag{2.30}$$

where a linear activation is assumed for the output unit, and f_h is the nonlinear activation of hidden units.[15]

Chapter 3

Diesel Engine Operation Characteristics

3.1 Engine Performance

As already discussed, the main scope of the developed neural network models is the prediction of some fundamental main engine parameters, which could be useful for further analysis of the engine performance and the exhaust gas emissions. Before going deeper into the presentation of the used parameters and their combinations, a concise presentation of the predicted values in accordance with the engine function would be helpful.

3.1.1 Fuel Consumption

Diesel engines, as internal-combustion ones, are considered a subcategory of the heat engines, because via the transformation of energy they produce work by operating between high and low temperatures, and pressures as well. During the process of combustion, the energy of fuel chemical bonds is converted into thermal energy, and finally into the mechanical energy. For this reason, diesel engines need fuel in order to ignite in the internal combustion chamber and set the start of the aforementioned energy conversion. A correctly tuned diesel engine consumes fuel according to its power requirements and, therefore, the more power it produces the more fuel it consumes.

In the case of a ship, a reduction of fuel consumption is easily achieved by reducing speed, so the ship owners control the engine load to set the most efficient speed. The Table 3.1 below shows a potential for fuel saving, according to [16].

Table 3.1: Relation between Speed Reductions and Fuel Saving parameters.

Speed reduction	Fuel Saving
2%	4%
4%	8%
6%	12%
8%	16%
10%	19%

3.1.2 Intake Manifold Absolute Pressure (MAP)

The MAP variable is the pressure in the intake manifold of the main engine, before the cylinders and right after the compressor of the turbocharger. It is also known as "turbocharger boost pressure" and in simple words is the generated high pressure sent to the engine. This unit is measured by the manifold absolute pressure sensor (MAP sensor), which provides instantaneous manifold pressure information to the engine's Electronic Control Unit (ECU). The data is used to calculate air density and determine the engine's air mass flow rate, which in turn determines the required fuel metering for optimum combustion and influence the advance or retard of ignition timing.

3.1.3 Air-fuel ratio equivalence factor - Lambda (λ)

Diesel engines, as thermal ones, use fuel and oxygen (from air) to produce energy through combustion. To guarantee the effectiveness and quality of the combustion process, certain quantities of fuel and air need to be supplied in the combustion chamber. A *complete combustion* takes place when all the fuel is burned and in the exhaust gas there will be no quantities of unburned fuel. **Air-fuel ratio** (AF or AFR) is the ratio between the air mass m_a and fuel mass m_f, used by the engine when running:

$$AFR = \frac{m_a}{m_f} \tag{3.1}$$

The inverse ratio is called fuel-air ratio (FA or FAR) and it is calculated as:

$$FAR = \frac{m_f}{m_a} = \frac{1}{AFR} \tag{3.2}$$

The ideal (theoretical) air-fuel ratio, for a complete combustion, is called **stoichiometric** air-fuel ratio. For a gasoline (petrol) engine, the stoichiometric air-fuel ratio is around 14.7:1, i.e. for every one gram of fuel, 14.7 grams of air are required. The combustion is possible even if the AFR is different than stoichiometric.

When the air-fuel ratio is higher than the stoichiometric ratio, the air-fuel mixture is called **lean**, whereas when the air-fuel ratio is lower than the stoichiometric ratio, the air-fuel mixture is called **rich**.

In reality, internal combustion engines do not work exactly with ideal AFR, but with values close to it. Therefore, there will be an ideal and an actual air-fuel AFR. The ratio between the actual air-fuel ratio (AFR_{actual}) and the ideal/stoichiometric air-fuel ratio (AFR_{ideal}) is called **Equivalence Air-Fuel Ratio** or **Lambda (λ)**.

$$\lambda = \frac{AFR_{actual}}{AFR_{ideal}} \tag{3.3}$$

Depending on the value of lambda, the engine is told to work with lean ($\lambda > 1.00$), stoichiometric ($\lambda = 1.00$) or rich ($\lambda < 1.00$) air-fuel mixture. Due to the nature of the combustion process, Compression Ignition (CI) diesel engines always run on lean mixtures, with AFR between 18:1 and 70:1. The term "lean mixtures" means that there is more oxygen than required to burn completely the amount of fuel and after the combustion there is excess oxygen in the exhaust gases.

3.1.4 Nitrogen Oxides (NOx)

Nowadays, one of the most discussed issues in the shipping field is the reduction of the main pollutant emissions from diesel engines. The Nitrogen Oxides – NOx (nitric oxide NO & nitrogen dioxide NO_2) are considered part of the most hazardous emissions, as they contribute to the formation of smog and acid rain, as well as affecting tropospheric ozone.

3.1.4.1 Formation of NOx

NOx are formed during the combustion process within the burning fuel sprays. At these elevated flame or combustion temperatures nitrogen is no longer inactive and reacts with oxygen to form nitric oxide (NO) and nitrogen dioxide (NO_2). The immediate reaction is the formation of NO. Later in the process, during expansion and in the exhaust system, part of the NO will convert to form NO_2. NOx is controlled by local conditions in the spray, with temperature and oxygen concentrations being the dominant influences. The higher the temperature and the longer the residence time at high temperature, the more NOx will be created.

The critical air-fuel ratio equivalence factor (λ) of the local mixture of fuel and air for NOx formation is close to 1 i.e. maximum NOx creation occurs when the local air-fuel ratio is close to stoichiometric. The critical time period for NOx creation is when burned gas temperatures are at a maximum i.e. between the start of combustion and shortly after peak pressure. It has been shown that almost all NO formation occurs within 20° of crank rotation following start of combustion. After the time of peak pressure, burned gas temperatures decrease as the cylinder gases expand. The decreasing temperature due to expansion and due to mixing of high temperature gases with cooler burned gas freezes the NO chemistry. NO forms both in the propagated flame front and in the post flame gases. In engines, however, combustion occurs at high pressure so the flame reaction zone is extremely thin (in the order of 0.1mm) and residence time within this zone is short. Thus, NO formation in the post-flame gases almost always dominates any flame-front produced NO which effectively de-couples the combustion and NO formation processes. However, the reactions which produce NOx take place in an environment created by the combustion reactions, so the two processes are still intimately linked. [17]

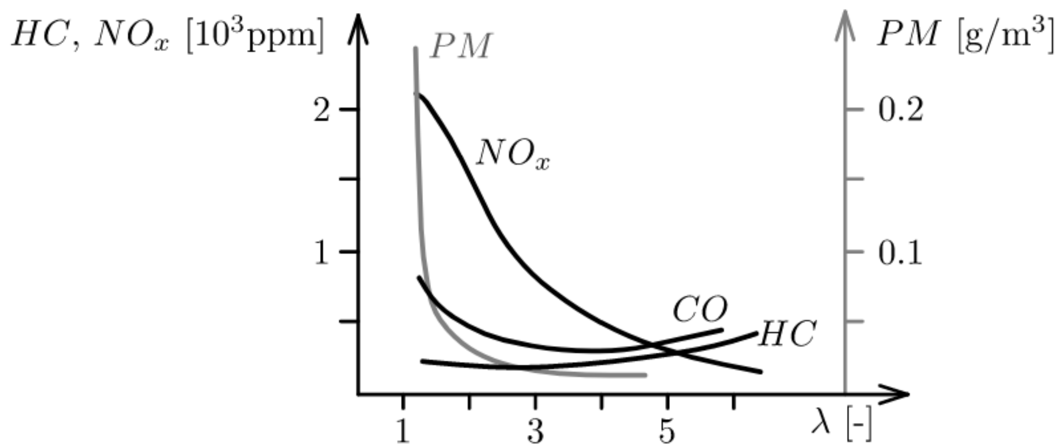


Figure 3.1: Engine-out emission of Nitrogen oxide NOx, hydrocarbon HC, and particulate matter PM of a direct-injection Diesel engine as a function of air-fuel ratio λ.

3.1.4.2 NOx Regulations

The NOx control requirements of *Regulation 13 of MARPOL Annex VI* apply to each marine diesel engine with a power output of more than 130 kW installed on a ship. A marine diesel engine is defined as any reciprocating internal combustion engine operating on liquid or dual fuel. There are two exceptions: engines used solely for emergencies and engines on a ships operating solely within the waters of the state in which they are flagged. The later exception only applies if these engines are subject to an alternative NOx control measure.

Different levels (Tiers) of control apply based on the ship construction date and within any particular Tier the actual limit value is determined from the engine's rated speed. Tier I and Tier II limits are global, while the Tier III standards apply only in NOx Emission Control Areas.

Tier	Ship construction date on or after	Total weighted cycle emission limit (g/kWh) n = engine's rated speed (rpm)		
		$n < 130$	$n = 130 - 1999$	$n \geq 2000$
I	1 January 2000	17.0	$45 \cdot n^{(-0.2)}$ e.g. 720 rpm - 12.1	9.8
II	1 January 2011	14.4	$44 \cdot n^{(-0.23)}$ e.g. 720 rpm - 9.7	7.7
III	1 January 2016	3.4	$9 \cdot n^{(-0.2)}$ e.g. 720 rpm - 2.4	2.0

Table 3.2: MARPOL Annex VI NOx Emission Limits[18]

3.1.5 Exhaust Gas Recirculation (EGR) System

The Exhaust Gas Recirculation (EGR) is one of the primary methods that can pe adopted in order to reduce NOx emissions and comply with the Tier III regulations.

In this technology, part of the exhaust gas is re-circulated back into the intake manifold. This process leads to a significant reduction in nitrogen oxides (NOx) emissions because it reduces the two elements underlying its production: oxygen in excess and combustion temperature. The emissions reduction is achieved from the fact that the exhaust gases, mainly carbon dioxide (CO_2), nitrogen (N_2), water (H_2O) and oxygen (O_2), act as a "diluent", which in combination with the high specific heats associated with the triatomic molecules, result in the direct reduction of the adiabatic flame temperature and the formation kinetics of nitrogen oxides (NOx). Since the specific heat capacity of both CO_2 and water vapor is greater than that of oxygen, the gas temperatures within the engine cylinder during combustion is reduced.

By recirculating the exhaust gas into the intake, some of the oxygen required for the combustion is replaced by inert (exhaust) gases, which leads to the reduction of excess oxygen. Also, because the exhaust gases absorb some of the heat generated during combustion, the maximum combustion temperature per engine cycle is also reduced.

The EGR system significantly reduces the amount of NOx, but if too much exhaust gases are introduced into the intake, it can have an impact on the increase in emissions of carbon monoxide (CO), hydrocarbons (HC) and particulate matter (PM), as a result of incomplete combustion due to lack of air (oxygen). The EGR system is active mainly during partial engine loads and at low and medium engine speed areas, where oxygen is

in excess. In the high engine load (torque), the EGR system is deactivated, the cylinders being filled only with air, ready for combustion.

Depending on the pressure of the recirculated exhaust gases, there are two types of external EGR systems: Low Pressure and High Pressure. In the experimental setup of this work the High Pressure type is installed, where the exhaust gases are harvested before entering the turbine and reintroduced in the intake manifold after the compressor, as in Figure 3.2. It is important to note that although the Figure 3.2 does not correspond entirely to the installed EGR system of the test-bed in this case, it is included for ease of reference.

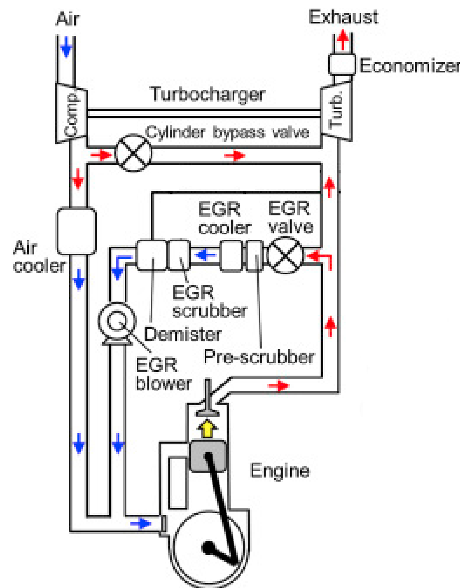


Figure 3.2: Schematic diagram of High-Pressure Exhaust Gas Recirculation (HP EGR) for marine diesel engines.[19]

One of the main components of an EGR system is the EGR Valve, which allows the exhaust gases to flow from the exhaust manifold into the intake manifold. The EGR valve is closed when the engine is starting up. At low speeds, only a small amount of power is required, and therefore only a small amount of oxygen, so the valve gradually opens. However as more torque and power is required, for example during full acceleration, the EGR valve closes to ensure as much oxygen enters the cylinder. In our case, a crucial parameter, which has been useful during the modeling, is the **EGR Valve Position (%)**, meaning the percent measuring of the *"how open is the valve ?"* value.

Chapter 4

Data Preparation

One of the most significant factors for the successful performance and predictions of a neural network is the combination of quantity and quality of data, training and testing ones. In machine learning algorithms and in predictive models, the collected raw data for the modeling usually cannot be used directly, without any preparation. Typically, there is a certain pattern to be followed for the data preparation, which commonly includes: Data Cleaning, Feature Selection, Data Transformation, Feature Engineering and Dimensionality Reduction. The aforementioned tasks, however, are not used for the design of every neural network. Their selection depends on the nature of both the problem and the available data, and plays a major role on the adaptivity of the developed neural network.

4.1 Data Collection

The **Hybrid Integrated Propulsion Powertrain 2 (HIPPO-2) test-bed**, seen below in Figure 4.1, consists of a 261 kW diesel engine connected to a 315 kW electric dynamometer (Brake). A 90 kW electric Motor/Generator is attached at the free end of the dynamometer.

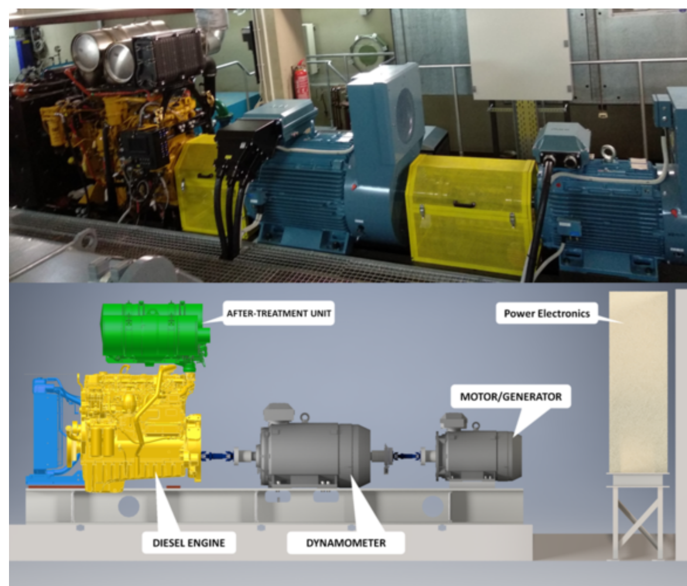


Figure 4.1: The hybrid diesel-electric test-bed HIPPO-2 at LME

The prime mover is a CATERPILLAR 6-cylinder, 9.3-liter diesel engine, with a rated power output of 261 kW at 1800-2200 rpm. The diesel engine is fitted with state of the art emission reduction technologies, an Exhaust Gas Recirculation (EGR) and a Selective Catalytic Reactor (SCR) NOx abatement systems. The electric motor/generator is a AC asynchronous-induction 3-phase motor, with a rated power of 90 kW. In this setup, the thermal and electric engine provide mechanical power simultaneously, with identical rotational speeds. All the machines of the HIPPO-2 system can be operated either in torque- or speed-control mode.

The performance monitoring and evaluation of the hybrid system is performed with the various sensors that are installed in the testbed, as shown in Figure 4.2. The whole test-bed is controlled and monitored by a dSPACE MicroAutobox II controller board, programmed in the MATLAB/Simulink environment. The MicroAutoBox II is a real-time system for performing fast function prototyping and it can operate without user intervention, just like an Engine Control Unit (ECU).

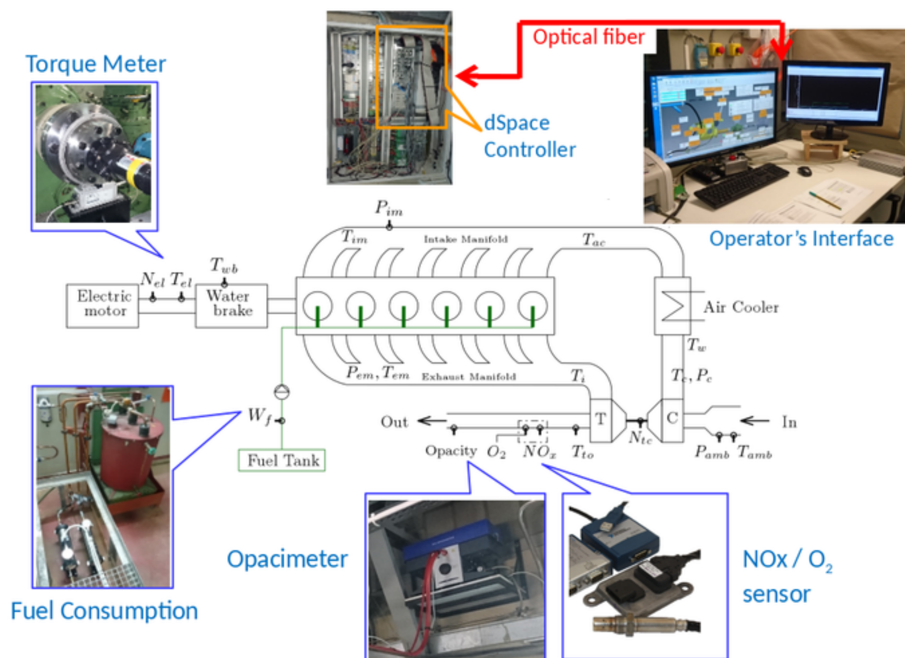


Figure 4.2: Sensors, rapid prototyping and operator interface of LME/NTUA experimental facilities

During experimental testing various loading profiles can be applied. The mechanical load, applied by the dynamometer can simulate for example the vessel's propeller demand, with variable speed and torque, or a generator's loading profile with changing load at constant speed.[20] Also, useful information is extracted at the time of test-bed operation, covering either the whole envelope of the engine or just a specific region. In particular, the engine variables -defined and measured by the sensors- are stored for further handling, analysis and investigation. As aforementioned, neural network models are designed for the prediction of some significant engine performance parameters, using the measured data either as inputs or indicative outputs for the training process.

4.2 Data Overview

In this case, the raw data were derived from real-time measurements of the *Hybrid Integrated Propulsion Powertrain 2 (HIPPO-2) test-bed* and extracted from the existed sensors. Afterwards, the data were modified by the laboratory personnel and researchers in order to have the most convenient unit measurement for analysis and to be easily processable. Specifically, the main data set, which is used for training, covers the whole operating envelope of the main engine, as Figure 4.3 implies. In fact, the *Speed* factor covers a range of 1000 rpm (rounds per minute), starting from 900 rpm and ending to 1900 rpm, as a combination of multiple step functions. This data set resulted from the merging of various smaller data sets, each one from the test-bed running at specific speed (e.g. 900 rpm, 1000 rpm, etc.). In addition to this, some extra smaller data sets are used for testing and present a totally different behavior from the training one. These sets represent more noisy measurements, affected mainly from the speed governor of the test-bed, which is affected by the water brake or by a heavier propeller curve. It is worthwhile mentioning that the graphs of these data sets are included in each section of TDNN models, in order to depict separately the different inputs of each model, and not all together without noting the differences in distributions.

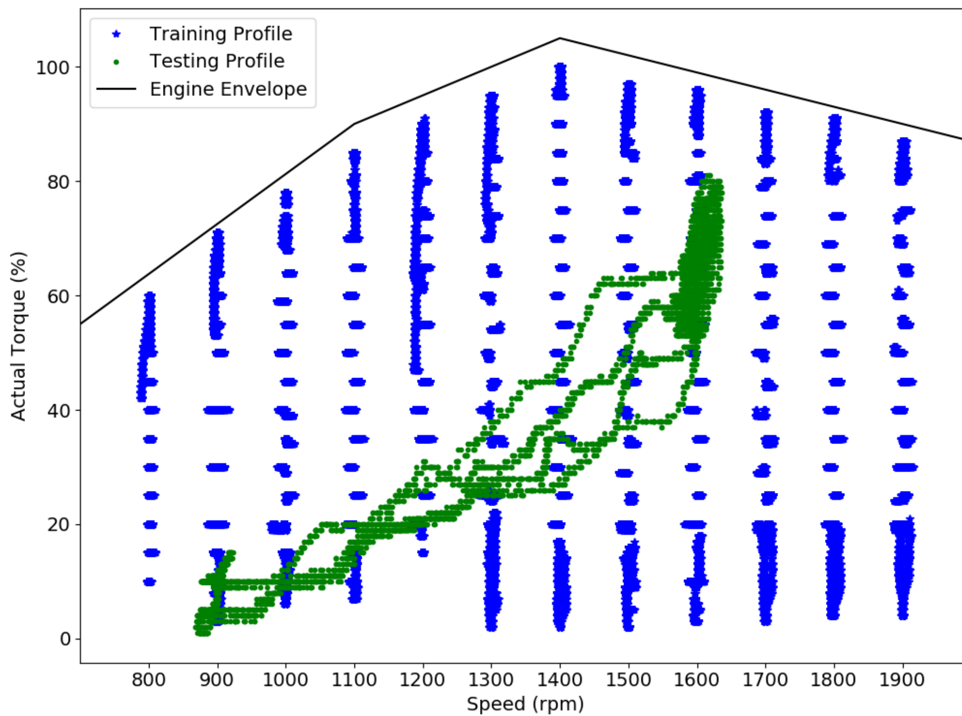


Figure 4.3: Engine operating conditions coverage

In the Table 4.1 below, all the extracted engine variables are collected.

Table 4.1: Engine Performance Parameters

Parameter	Unit
Actual Torque (i.e. electronic fuel index)	%
Motor Torque (i.e. brake torque)	Nm
Speed	rpm
Fuel Consumption	lt/h
MAP	kPa
Exhaust Gas Mass Flow Rate	g/s
λ	-
EGR Valve Position	%
NOx	ppm

The predicted values of the FFNN and TDNN models, as already discussed, are:

1. **Fuel Consumption (lt/h)**
2. **Intake Manifold Absolute Pressure (MAP)**
3. **Lambda (λ)**
4. **NOx (ppm)**

However, in the following Figures all the used data are illustrated, in order to have a clear view of them and their schemes. The time (x-axis) in all diagrams is expressed in *seconds*, to be more realistic.

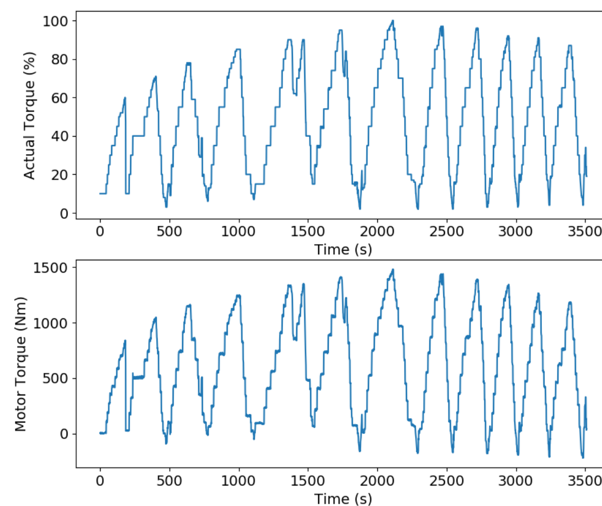


Figure 4.4: Actual (%) and Motor (Nm) Torque Diagrams

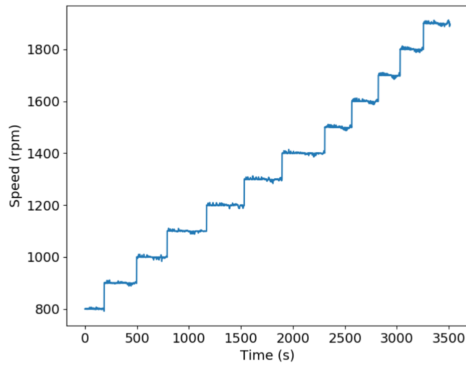


Figure 4.5: Speed Diagram

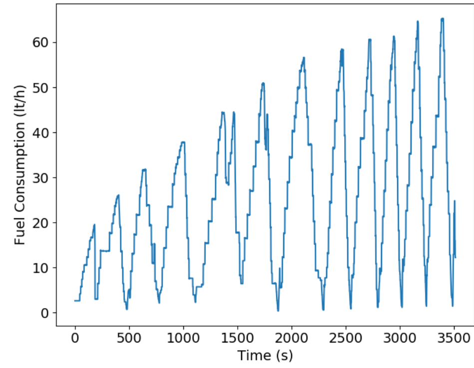


Figure 4.6: Fuel Consumption Diagram

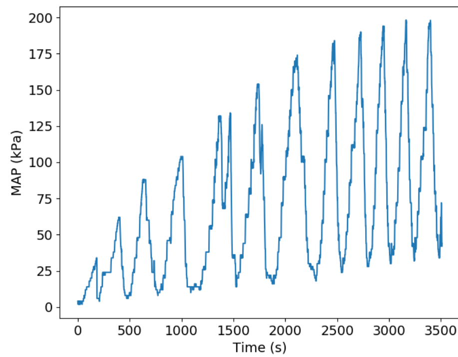


Figure 4.7: Engine Intake Manifold Pressure Diagram

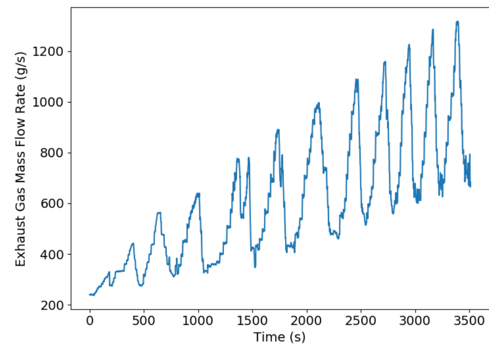


Figure 4.8: Exhaust Gas Mass Flow Rate Diagram

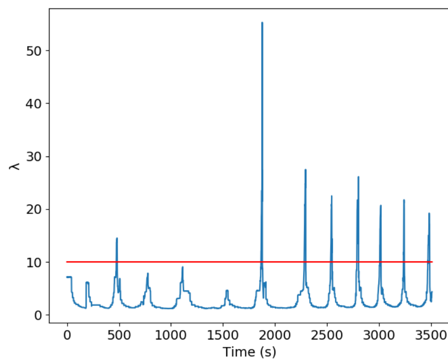


Figure 4.9: Lambda (λ) Diagram - The red lined implements the ”saturation limit”, above which the data are saturated and therefore will not be kept.

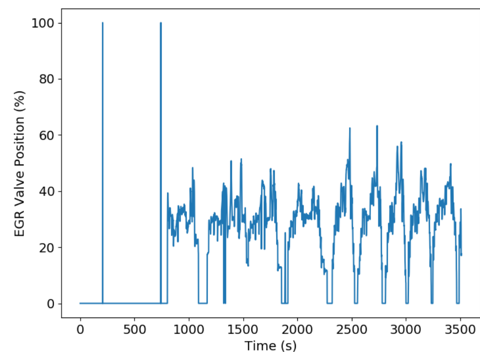


Figure 4.10: EGR Valve Position Diagram - At the starting of the test-bed the EGR valve remains closed.

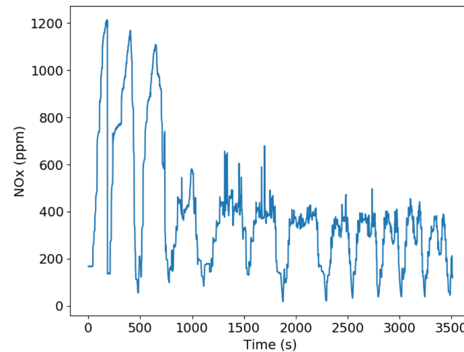


Figure 4.11: NOx Diagram

4.3 Data Pre-Processing

4.3.1 Data Synchronization

All individual training and testing data sets could be considered time series, as a series of data points ordered in time. They all have a time step of 0.01 second, according to the set measuring interval of sensors. Because of this constant time step, without increase and decrease, each value of a data set could also be thought as a sample. The modeling and the final prediction of a specific value-series could be considered time-independent. In simple words, there is no utility to know the exact time-step of a "NOx" change, but how this change differentiates with the change of "λ" or that of "EGR Valve Position". In addition to this, after all the stages of data-preparation, the initial time step may be lost, so the most important thing to know would be the "new" time step and not the moments of measurements.

As far as the sampling of data is concerned, the input data sets of a neural network model should be synchronized with the predicted, output value. The reason is the importance of the dynamic fluctuations of data and whether happen at the same time step or how many samples the different time steps vary from each other. Of course, in scale of centiseconds (0.01 sec) a difference of some tens could be ignored, but this should be investigated at the beginning.

The **Cross Correlation** is one of the most common measures of similarity of two series in signal processing, but it also has applications in pattern recognition. It shows the displacement of one series relative to the other, hence it is used for the data synchronization and erase of lag in this study. In probability and statistics, the term cross-correlations refers to the correlations between the entries of two random vectors $\mathbf{X}$ and $\mathbf{Y}$, while the correlations of a random vector $\mathbf{X}$ are the correlations between the entries of $\mathbf{X}$ itself, those forming the its correlation matrix. If each of $\mathbf{X}$ and $\mathbf{Y}$ is a scalar random variable which is realized repeatedly in a time series, then the cross-correlations of $\mathbf{X}$ with $\mathbf{Y}$ across time are temporal cross-correlations. In probability and statistics, the definition of correlation always includes a standardizing factor in such a way that correlations have values between -1 and $+1$. In time series analysis and statistics, the cross-correlation of a pair of random process is the correlation between values of the processes at different times, as a function of the two times. Let $(\mathbf{X}_t, \mathbf{Y}_t)$ be a pair of random processes, and t be any point in time (t may be an integer for a discrete-time process or a real number for a continuous-time

process). Then X_t is the value (or realization) produced by a given run of the process at time t. Suppose that the process has means $\mu_X(t)$ and $\mu_Y(t)$ and variances $\sigma_X^2(t)$ and $\sigma_Y^2(t)$ at time t, for each t. Then the definition of the cross-correlation between times t_1 and t_2 is:

$$R_{XY}(t_1, t_2) = E[X_{t_1}\overline{Y_{t_2}}] \tag{4.1}$$

where E is the *expected value* operator, meaning the arithmetic mean.

In this thesis, for the calculation of cross correlation the environment of MATLAB was used and in particular the *xcorr* built-in function. The latter returns the cross-correlation of two discrete-time sequences. It takes an array as reference (e.g. Fuel Consumption), which is the predicted array, and an other array (e.g. Actual Torque (%)) to find its lag from the first one. The first array is constant and the second one is shifted various times (as a function of the lag) in order to repeat the calculation of the cross-correlation between the two series. The number of times that the process is repeated depends on the amount of data and as the lag increases the potential matches will decrease. The lag with the highest cross-correlation value is where the two series match the best, and this point will be used as reference to the transformation of both arrays. The sequences have to be of the same length, assuming N, in order to be automatically normalized so that the cross-correlation varies from 0 to 1, calculated as follows:

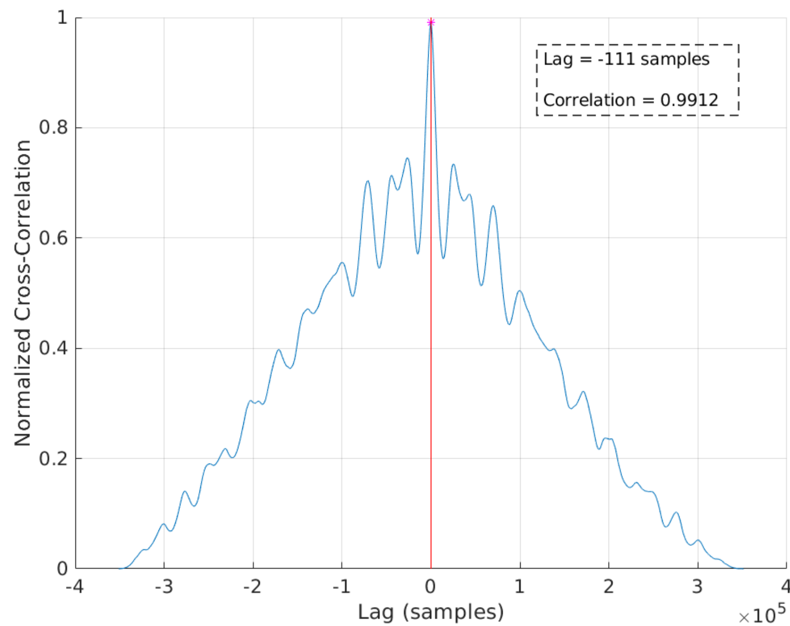
$$\widehat{R}_{xy,coeff}(m) = \frac{1}{\sqrt{\widehat{R}_{xx}(0)\widehat{R}_{yy}(0)}}\widehat{R}_{xy}(m) \tag{4.2}$$

The cross-correlation equation has an output of all cross-correlations per lag, which covers the interval: $lag \in [-N+1, N-1]$, where N is the length of each data set. Consequently, the length of the cross-correlation or lag array is equal to $2N-1$ and the lag is counted in samples (i.e. array elements). If its value is negative, it means that the reference array (e.g. Fuel Consumption) follows the second one (e.g. Actual Torque (%)), or if its value is positive, it means that the reference array precedes the other one. If only the common, synchronous samples of the two arrays are kept and then the cross correlation is re-calculated, the result will be a lag of 0.

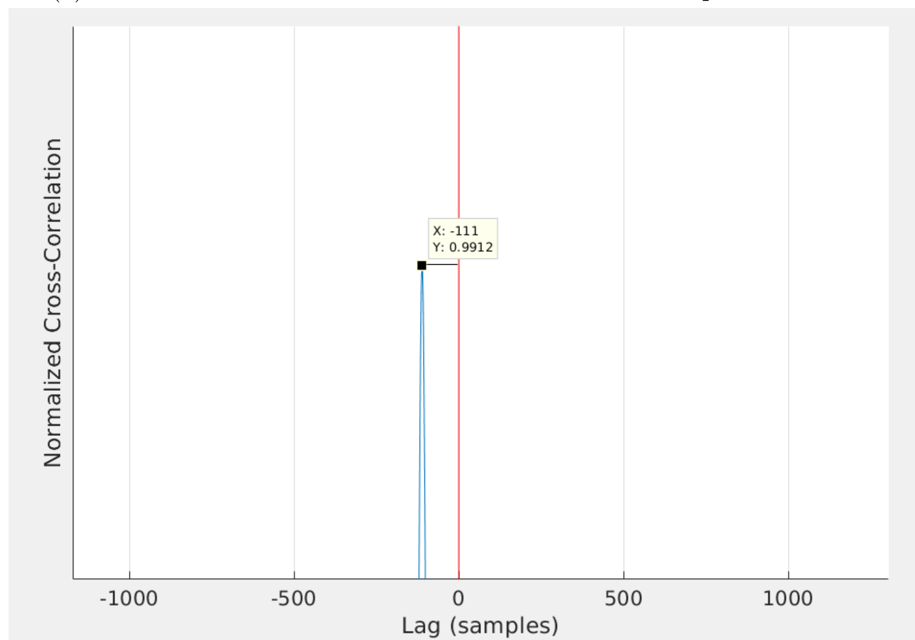
In the examined case, the cross-correlation was calculated for every predicted value, between the output data set and each input data set that was going to be used for the prediction. The main idea was to delete the lag between each combination of input & output data sets and synchronize them in order to begin at the same time-step, but according to the explanation of how the *xcorr* function works, this was not possible in all cases. For instance, for a neural network which predicts the "Fuel Consumption" through the processing of "Actual Torque", "Motor Torque", "Engine Intake Manifold Pressure", "Speed" and "Exhaust Gas Mass Flow Rate", it is quite complicated to have all the data synchronized with the "Fuel Consumption" and with the same array length. For each different lag between the arrays and the "Fuel Consumption" there should be a length decrease of all arrays in order to have the same length and be synchronized. However, there is no need for all these complex transformations in arrays and their indexing, when referring to a lag of some tens of samples. For this reason, if the "Actual Torque" array precedes the "Fuel Consumption" with $lag = -x_1$ and the "Speed" follows the reference array with $lag = x_2$ the transformation is done in order to have the "Fuel Consumption" and "Actual Torque" synchronized, and the "Speed" to follow them. The predicted array must be the reference of counting, i.e. it must precedes, and no other array should have already started. As a result, even though the criterion of synchronization of arrays is partially satisfied, the non-erased lag is insignificant when thinking of it in real time units (e.g 50 samples lag corresponds only to 0.5 seconds), which would not harm the performance of the neural network.

It is worth mentioning that, not only is the large training data set pre-processed, but also some smaller data sets used just for testing, in order to be comparable with the inputs of the trained model.

Below, there is a typical *Cross-Correlation* diagram.



(a) Normalized Cross-Correlation between Fuel Consumption and MAP

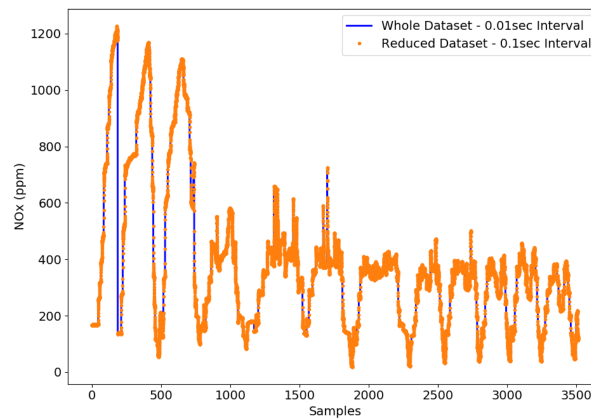


(b) Deviation between the maximum cross-correlation point and the Lag=0 vertical line.

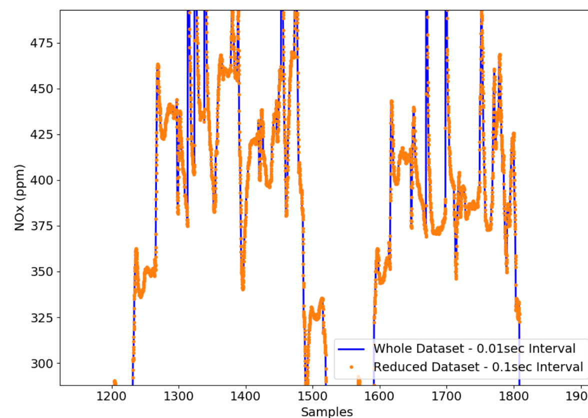
Figure 4.12: Cross-Correlation between Fuel Consumption and Engine Intake Manifold Pressure

4.3.2 Data Re-sampling

The large training data set contains the most significant and representative measurement data for the engine operation, which are just a part of the total extracted data. All data sets have a time step of 0.01 second, according to the set measuring interval of sensors, as already mentioned. For example, the large training data set, which covers the whole main engine envelope, is equivalent to 351174 samples of 0.01 seconds, i.e. 59 minutes of running, approximately. As it is obvious, it contains a large amount of samples, even after the fixing of lag, which will slow down the training of the neural network, or make it more complex. For this reason, before proceeding into the development of the model, a reduction of the inputs size would be quite helpful, without affecting the nature of measurements and the distribution of data points. In particular, the data were kept with a step of 10 samples, and after that the interval between two consecutive positions is 0.1 seconds. As Figure 4.13 demonstrates, even the resampled NOx data set contains more than enough points for training.



(a)



(b) Zoomed region, full of points.

Figure 4.13: Initial (blue line) and Re-sampled (orange dots) NOx Dataset.

Chapter 5

Model Design

After the data pre-processing and the configuring of both the basic training data set and the extra validation data sets, the Model Design is the most important part, demanding patience and attention. This section is about the process of creating a neural network, choosing its general structure and basic hyperparameters. All these details have been set specifically for each model and according to the quality and quantity of the predicted data. For this reason, all the noteworthy parameters will be explained and generally defined in this section. The whole process for all models was designed using the Python language, and in such a way that are easily adaptable and suitable to similar application. The libraries used from the data handling to the final model prediction are the following: Tensorflow 2.0, Keras (High-level API of Tensorflow 2.0), SciPy, NumPy, Pandas, Seaborn, Scikit-learn, Matplotlib.

5.1 Model Inputs Selection

As already mentioned, in this thesis, the designed ANN models should predict separately four different units: Fuel Consumption (lt/h), Intake Absolute Manifold Pressure (MAP (kPa)), Air-to-Fuel ratio equivalence factor λ and NOx (ppm). When designing a neural network, the first thing to define is the output, which is obviously the purpose of the model. After that the inputs should be selected accordingly and then the model hyperparameters and calculation methods. However, there is a "thin line" between the inputs selection and the network design. To start with, the inputs of the model are selected according to the available measurements-data, the nature of the problem and the existing theory. After that the model design follows, where the basic hyperparameters are selected and then the first attempts of training take place. If the training process overfits quickly or if the errors are high, the model should be re-designed. However, an unsuccessful result is not always fault of the model calculation mechanism, but it could be caused due to inappropriate inputs. In fact, evaluating a neural network on an input that differs markedly from the training data might cause erratic and flawed predictions. [21] For this reason, in the case of deep neural networks not only are the hyperparameters redefined, but also the inputs of the model. In addition, an other major factor when selecting the model inputs is the availability of data from the installed sensor systems or the ability to extract the appropriate units from the measured quantities via simple calculations or maps. [22].

The Fuel Consumption parameter is counted in liters per hour and is equivalent to the fuel mass flow rate (kg/sec), which is closely related to the mass of fuel injected in the combustion chamber per cycle. In order to achieve a specific air-fuel ratio λ, the MAP

should be somewhat pre-defined in order to insert the right amount of air and therefore of fuel in the engine. Also, the Exhaust Gas Mass Flow Rate (g/s) unit represents the mass of the total exhaust gas per second, and is dependable of the mass of air and fuel as the following formula implies: $m_{exh.gas} = m_{fuel} + m_{air}$

Nitrogen oxides formation in combustion process is known to be dependent on the local O_2 and N_2 concentrations and on the local temperature, with the rate of formation of NOx well represented by the Zeldovich mechanism. The knowledge of these parameters could allow to directly define a simple NOx estimation model. Unfortunately, it is not possible to reliably measure the local in-cylinder temperature and oxygen concentration, especially with the target of defining a model for on board estimation. In that way the problem may become more complicated, but the unknown values are dependent on the inputs of combustion process that are managed by the engine management system. The NOx concentration estimation at engine outlet can be described as a function of: $NOx[ppm] = f_{NOx}(m_f, m_{eng}, m_{EGR}, \lambda, n, SOI, P_{rail}, T_{stroke}, AirH)$

The NOx concentration estimation cannot be carried out considering all the highlighted physical inputs, but only the information (measurement and estimation) available to the ECU. Furthermore, to reduce the model complexity, the number of inputs was limited, considering only the most significant inputs in term of NOx correlation [8] and those which do not worsen the training process. For this reason, in the designed model the λ parameter was not included as various attempts proved that it did not improve the model training and performance. This may occurred because of the extremely high values (spikes) of λ in specific points. Also, the fact that NOx formation is a highly non-linear phenomenon made the prediction of NOx species even harder, so less inputs were finally chosen in order to avoid too early overfitting and large errors. Finally, the NOx concentration estimation at engine outlet can be described as a function of: $NOx[ppm] = f_{NOx}(ActualTorque(\%), Speed(rpm), EGRValvePosition(\%))$.

An other major factor for choosing the model inputs is the effortless re-usability of the networks, so a small number of inputs combined with the most easily accessible quantities would be ideal. With this in mind and according to the theoretical background in Section 3.1 and to the aforementioned criteria for inputs selection, the λ and MAP outputs were modeled as in Figure 5.1. It is worthwhile noting that the Speed (rpm) and Actual Torque (%) variables are standard inputs to all models and can be thought as operating point selectors[22]. The Speed, specifically, is the main factor of the problem as all units are defined according to its “step” form and this becomes evident in all the ”Pairwise Distributions” Figures of Chapter 6.

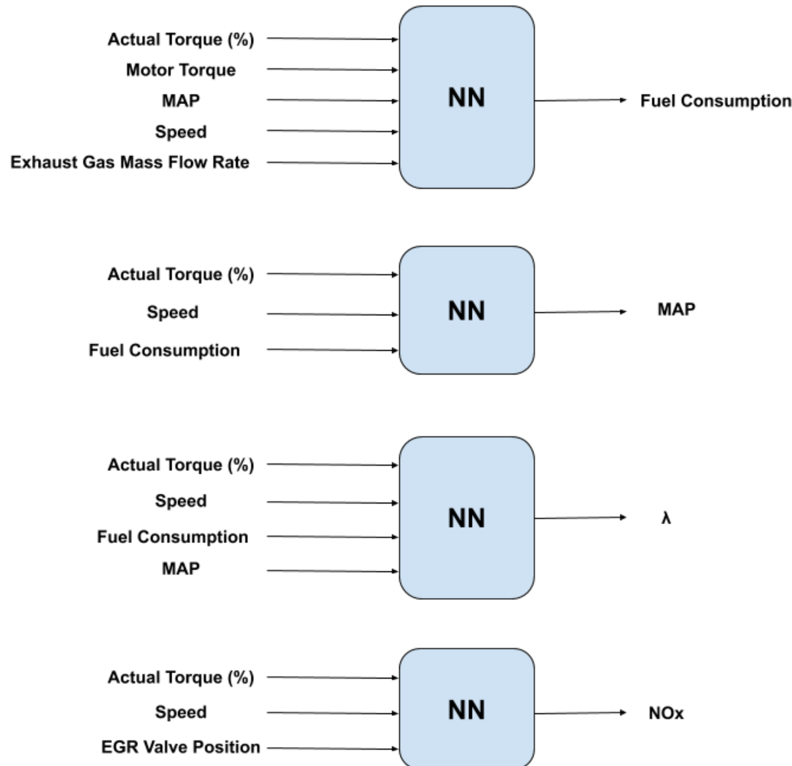


Figure 5.1: Selected Inputs for all the models.

5.2 Training, Validation & Testing Datasets

As aforementioned, the MATLAB environment was basically used for the data preparation, in particular for the calculation of cross-correlation. The new re-sampled data were saved in a *.mat* file and loaded into Python environment for further processing. The *.mat* file contains all the useful values for the training, where all of them are formed as separate lists with the same length. Afterwards, all these lists are merged into a data structure called *DataFrame*, where each column represents a different measurement with a unique label on top of it. Then, the data have to be split into Training data set, which is used to fit the model and Testing data set, for providing an unbiased evaluation of the final model fit on the training data set. The testing data set is used once a model is completely trained and in order to assess the performance of the neural network. Particularly, due to the large size of the aforementioned initial and pre-processed data set, the 70% was taken for training and the rest 30% for testing. This separation is done in a random way, to ensure that the train and test data sets are representative of the original one. Moreover, a part of the training data, called Validation data set, is used to evaluate a model fit on the training data set while tuning model hyperparameters (e.g. the weights, learning rate, etc.). The interesting thing about it is that it functions as a "hybrid": it is training data used for testing, but neither as part of the low-level training nor as part of the final testing. In the following designed neural networks, the validation data sample takes the 20% of training data set values.

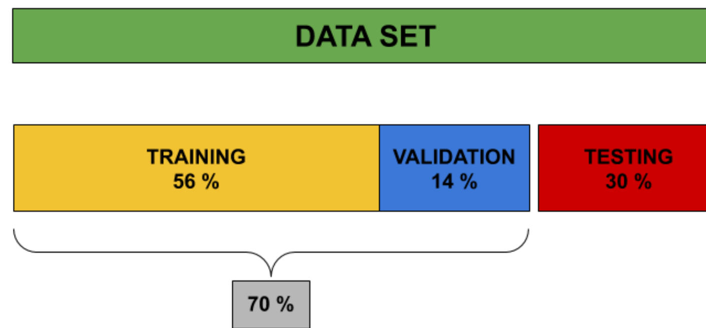


Figure 5.2: Training, Validation & Testing data sets

5.3 Data Normalization

An other crucial factor before starting the training procedure is the Data Scaling. The input variables of the model have many different units (e.g. lt/h, ppm, g/s, Nm, kPa etc.) and therefore many different scales, which may become an obstacle to model training process. On top of that, a target variable with a large spread of values may result in large error gradient values, causing weights to change dramatically, making the learning process unstable. For all these reasons and due to model complexity (deep learning model), the data scaling is applied through the **Data Normalization** method, so the model learns faster and produces more accurate results. All the data are normalized within the range [0,1], according to the following equation:

$$\mathbf{X}_{scaled} = \frac{\mathbf{X} - min(\mathbf{X})}{max(\mathbf{X}) - min(\mathbf{X})} \tag{5.1}$$

As far as the unknown validation data samples are concerned, they have been normalized according to training data set maximum and minimum values. Collating the range of values in both the above data sets, it seems that the training one covers a great area of data where the other just a part of it. For this reason, if the values are not scaled according to the larger limits there will be a divergence between the original and the predicted results, which will be falsely and misleadingly attributed to network parameters.

5.4 Fine-Tuning of Model Hyperparameters

At this point, all the pre-processing requirements are met in order to build the model. So in this section the basic selected parameters of the designed neural networks are going to be presented and explained. It is worth mentioning that there are many guidelines and empirical instructions for the model design procedure. However, there is no "golden rule" when choosing a hyperparameter; it is upon the designer perspective and highly dependent on the "Trial-and-Error" results. The latter term refers to the continuous attempts of fitting the model, choosing different hyperparameters each time in order to have an overview of the most efficient of them. Before proceeding to the hyperparameters selection and description, the selected type of model for all the developed networks is the "Sequential" model, because it is the easiest and most convenient model to build in Keras. According to its naming, it

enables the user to create a model "sequentially", with the output of each layer to be an input to the next specified layer.

5.4.1 Number of Hidden Layers

For many problems, a model with just a single hidden layer can get reasonable results. A multilayer perceptron (MLP) -class of feed-forward ANN- with just one hidden layer can theoretically model even the most complex functions, provided it has enough neurons. However, for complex problems, deep networks have a much higher parameter efficiency than shallow ones. In other words, they can model complex functions using exponentially fewer neurons than shallow nets, allowing them to reach much better performance with the same amount of training data.

Real-world data is often structured in such a hierarchical way, and deep neural networks automatically take advantage of this fact: lower hidden layers model low-level structures (e.g. line segments of various shapes and orientations), intermediate hidden layers combine these low-level structures to model intermediate-level structures (e.g. squares, circles), and the highest hidden layers and the output layer combine these intermediate structures to model high-level structures (e.g. faces). Not only does this hierarchical architecture help Deep Neural Networks (DNNs) converge faster to a good solution, but it also improves their ability to generalize to new, unknown datasets, completely different from the input ones.[6]

For the aforementioned reasons, the designed models of this thesis have a DNN architecture in order to provide the optimum results and be able to predict sufficiently even on the completely unknown testing datasets. The complexity of structure of predicted values plays a major role when choosing the number of hidden layers and for this reason the following models have different layouts. For example, the TDNNs, in some cases, may have less layers than the FFNNs, due to their multitudinous inputs.

Moreover, in all models the *Dense Layer* type was used, which is the original deeply connected neural network layer and the most common and frequently used.[23] Dense layer does the below operation on the input and return the output:

$$output = activation(dot(input, kernel) + bias) \tag{5.2}$$

Where,

- **input**: the input data
- **kernel**: the weights matrix created by the layer
- **dot**: Numpy dot product (multiplication product) of all input and its corresponding weights
- **bias**: (if applicable) a bias vector created by the layer to optimize the model
- **activation**: the element-wise activation function passed as the activation argument

5.4.2 Number of Neurons per Hidden Layer

As explained in a previous section, the number of neurons in the input and output layers is determined by the type of input and output the task requires. For all the following models the output neuron is just one predicted array, e.g. Fuel Consumption, MAP, NOx and λ,

and the input neurons are more than one, but contingent upon the predicted value and their relation. As for the hidden layers, it used to be common to size them to form a pyramid, with fewer and fewer neurons at each layer. However, this practice has been largely abandoned because it seems that using the same number of neurons in all hidden layers performs just as well in most cases, or even better; plus, there is only one hyperparameter to tune, instead of one per layer. That said, depending on the dataset, it can sometimes help to make the first hidden layer bigger than the others. It should be taken into consideration that, if a layer has too few neurons, it will not have enough representational power to preserve all the useful information from the inputs and some of them may be lost.[6]

Before the developed models were finalized, various combinations and adaptations had been implemented and the most efficient were kept. To assess the final results a combination of the hierarchical way and that of the same number of neurons was implemented in order to achieve the "best" results. Further details about the exact number of hidden layers and neurons will be given in each model presentation section.

5.4.3 Activation Function

For a given node, the inputs are multiplied by the weights in a node and summed together, as in (5.2). This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output. The simplest activation function is referred to as the *Linear Activation*, where no transform is applied at all. A network comprised of only linear activation functions is very easy to train, but cannot learn complex mapping functions. This kind of functions is still used in the output layer for networks that predict a quantity (e.g. regression problems), hence in this thesis all the neural networks in the last hidden layer use linear activation or no activation at all, which is exactly the same. Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data. To assess the output values, two different non-linear activation functions were used in the different types of models, according to the output structure and the trial-and-error attempts.

5.4.3.1 Sigmoid Function

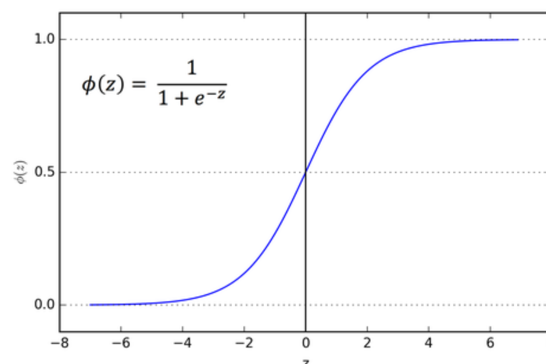


Figure 5.3: Sigmoid (Logistic) Function

The sigmoid function (also called *logistic function*) exists between (0,1) as shown in Figure 5.3. After normalizing all the input and output data sets both for training and testing, they belong in the range of 0 and 1, so the sigmoid function could be the right one. A

great advantage of sigmoid function is its smooth distribution and simple derivative $\phi(z) * (1 - \phi(z))$, which is differentiable everywhere on the curve. Hence, the smooth gradient prevents any "jumps" in output values. Also, around z=0 where z is neither too large or too small (in between the two outer vertical dotted grid lines in Figure 5.3), there is relatively more deviation as z changes, enabling the network to predict values in this range with less error. This happens because the sigmoid function is really sensitive to changes around its mid-point of its input. Nevertheless, this sensitivity is limited for very high or very low values of z. As a matter of fact, if z has a very negative value, then the output is approximately 0 and if z has a very positive value, the output is approximately 1, so the predictions are completely clear but there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.

5.4.3.2 ReLU (Rectified Linear Unit)

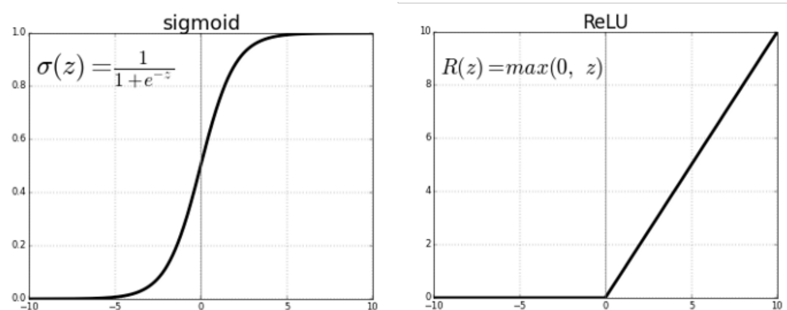


Figure 5.4: Sigmoid Function vs ReLU Function

Although the use of nonlinear activation functions allows neural networks to learn complex mapping functions, they effectively prevent the learning algorithm from working with deep networks. The ideal solution would be an activation function that looks and acts like a linear function, but is, in fact, a nonlinear function allowing complex relationships in the data to be learned. The function must also provide more sensitivity to the activation sum input and avoid easy saturation. The solution is to use the *Rectified Linear Activation Function (ReL)* and the node or unit that implements this activation function is referred to as a **Rectified Linear Activation Unit (ReLU)**. This function allows the network to converge very quickly and it does not have the vanishing gradient problem suffered by sigmoid function. Linearity means that the slope does not plateau, or "saturate," when the input gets large. ReLU issue where all the negative values becoming zero decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately. Having said that, this problem does not affect the designed models in our case, because all the input units have positive values in all samples.

5.4.4 Optimizer

Training a very large deep neural network can be painfully slow. One possible solution to this problem is to use a faster optimizer than the regular Gradient Descent Optimizer, as described previously in the theoretical section. The optimizers are algorithms or methods

used to change the attributes of a neural network, such as weights and learning rate in order to reduce losses. In this case the learning rate is stable and equal to 0.001 or 0.0005, depending on the predicted value. In this section the Adamax and RMSprop optimizers will be presented, as implemented in the designed networks. [6]

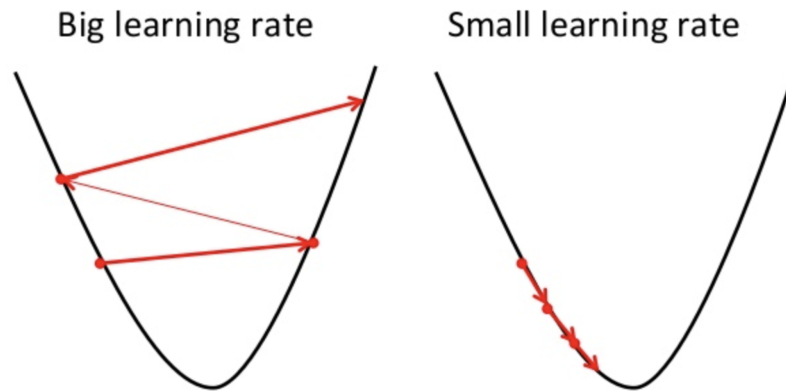


Figure 5.5: Different Learning Rates Behavior

5.4.4.1 RMSprop

Almost always, gradient descent with momentum converges faster than the standard gradient descent algorithm. In the standard gradient descent algorithm, larger steps are taken in one direction and smaller steps in another direction which slows down the algorithm. This is what momentum does, it restricts the oscillation in one direction so that our algorithm can converge faster.

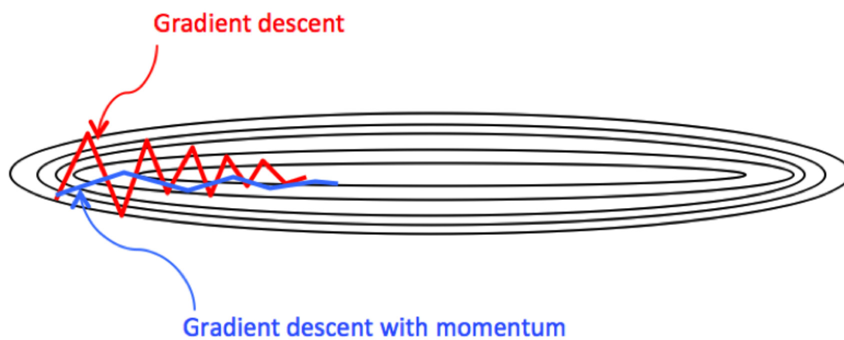


Figure 5.6: Gradient Descent Steps

The RMSprop optimizer is similar to the gradient descent algorithm with momentum, by restricting the oscillations in the vertical direction. Therefore, the learning rate could be increased and the algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated, expressed by the following equations:

Gradient descent with momentum:

$$\begin{aligned} \upsilon_{dw} &= \beta \cdot \upsilon_{dw} + (1-\beta) \cdot dw \\ W &= W - \alpha \cdot \upsilon_{dw} \\ \upsilon_{db} &= \beta \cdot \upsilon_{db} + (1-\beta) \cdot db \\ b &= b - \alpha \cdot \upsilon_{db} \end{aligned} \tag{5.3}$$

RMSprop Optimizer:

$$\begin{aligned} s_{dw} &= \beta \cdot s_{dw} + (1-\beta) \cdot dw^2 \\ W &= W - \eta \cdot \frac{dw}{\sqrt{s_{dw}} + \epsilon} \\ s_{db} &= \beta \cdot s_{db} + (1-\beta) \cdot db^2 \\ b &= b - \eta \cdot \frac{db}{\sqrt{s_{db}} + \epsilon} \end{aligned} \tag{5.4}$$

The value of momentum is denoted by β and is usually set to 0.9 and the η parameter is the *learning rate*, as in (2.22). Sometimes the value of υ_{dw} could be really close to 0. Then, the value of our weights could blow up. To prevent the gradients from blowing up, a parameter ϵ is included in the denominator which is set to a small value.

5.4.4.2 Adam (Adaptive Moment Estimation)

Adam optimizer combines the ideas of Stochastic Gradient Descent with momentum and RMSprop algorithms. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.

$$\begin{aligned} \upsilon_{dw} &= \beta_1 \cdot \upsilon_{dw} + (1-\beta_1) \cdot dw \\ \widehat{\upsilon}_{dw} &= \frac{\upsilon_{dw}}{1-\beta_1^t} \\ \upsilon_{db} &= \beta_1 \cdot \upsilon_{db} + (1-\beta_1) \cdot db \\ \widehat{\upsilon}_{db} &= \frac{\upsilon_{db}}{1-\beta_1^t} \\ s_{dw} &= \beta_2 \cdot s_{dw} + (1-\beta_2) \cdot dw^2 \\ \widehat{s}_{dw} &= \frac{\upsilon_{dw}}{1-\beta_2^t} \\ s_{db} &= \beta_2 \cdot s_{db} + (1-\beta_2) \cdot db^2 \\ \widehat{s}_{db} &= \frac{\upsilon_{db}}{1-\beta_2^t} \\ W &= W - \eta \cdot \frac{\widehat{\upsilon}_{dw}}{\sqrt{\widehat{s}_{dw}} + \epsilon} \\ b &= b - \eta \cdot \frac{\widehat{\upsilon}_{db}}{\sqrt{\widehat{s}_{db}} + \epsilon} \end{aligned} \tag{5.5}$$

In the above equations, t represents the iteration number, as in (2.21), starting at 1. The momentum decay hyperparameter β_1 is typically initialized to 0.9, while the scaling decay hyperparameter β_2 is often initialized to 0.999. As earlier, the smoothing term ϵ

is usually initialized to a tiny number such as 10^{-7}. Since Adam is an adaptive learning rate algorithm, it requires less tuning of the learning rate hyperparameter, with 0.001 as a default value.

5.4.4.3 Adamax

In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a (scaled) L_2 norm of their individual current and past gradients. The L_2 norm based update rule can be generalized to a L_p norm based update rule. Note that β_2 is also parameterized as β_2^p:

$$s_{dw} = \beta_2^p \cdot s_{dw} + (1 - \beta_2^p) \cdot dw^p \tag{5.6}$$

Norms for large p values generally become numerically unstable, which is why L_1 and L_2 norms are most common in practice. However, L_∞ also generally exhibits stable behavior. For this reason, *Diederik P. Kingma* and *Jimmy Lei Ba* [24] propose Adamax and show that s_{dw} with L_∞ converges to the following more stable value. In Adamax, the infinity norm-constrained is represented as $s_{inf,dw}$, to avoid confusion with Adam, and is defined:

$$s_{inf,dw} = \beta_2^\infty \cdot s_{dw} + (1 - \beta_2^\infty) \cdot dw^\infty \tag{5.7}$$

The (5.7) can be plugged into the Adam update equation by replacing $\sqrt{\widehat{s}_{db}} + \epsilon$, in order to obtain the Adamax update rule:

$$W = W - \eta \cdot \frac{\widehat{\upsilon}_{dw}}{s_{inf,dw}} \tag{5.8}$$

Note that as $s_{inf,dw}$ relies on the max operation, it is not as suggestible to bias towards zero as υ_{dw} and s_{dw} in Adam, which is why there is no need to compute a bias correction $s_{inf,db}$.[25]

5.4.5 Metrics

Mean Squared Error: It measures the average of the squares of the errors, that is the average squared difference between the estimated values and the actual values. It is always non-negative according to its definition and the values closer to zero are preferable. If a vector of n predictions is generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\widehat{Y}$ being the predicted values, then the within-sample MSE of the predictor is computed as:

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (Y_i - \widehat{Y}_i)^2 \tag{5.9}$$

In other words, the MSE is the *mean* $(\frac{1}{n} \sum_{i=1}^{n})$ of the *squares of the errors* $(Y_i - \widehat{Y}_i)^2$.

Mean Absolute Error: It measures the average of the absolute errors, that is the average absolute difference between the estimated values and the actual values. It uses the same scale as the data being measured. If a vector of n predictions is generated from a sample of n data points on all variables, and Y is the vector of observed values of the variable being predicted, with $\widehat{Y}$ being the predicted values, then the within-sample MSE of the predictor is computed as:

$$MAE = \frac{\sum_{i=1}^{n} \left| Y_i - \widehat{Y}_i \right|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n} \tag{5.10}$$

Mean Absolute Percentage Error (MAPE): It measures the error between the observed and the predicted values as a percentage.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}|\frac{Y_i - \widehat{Y}_i}{Y_i}| \tag{5.11}$$

The absolute value in this calculation is summed for every predicted point in time and divided by the number of fitted points n. Multiplying by 100% makes it a percentage error.

5.4.6 Early Stopping Callback

Callbacks in Keras library are objects that can perform actions at various stages of training (e.g. at the start of an epoch, before or after a single batch, etc.). One available and really helpful callback during training is the *Early Stopping Class*. The latter stops training when a monitored metric has stopped improving. Most of the times, this metric is the loss during the process of validation, in other words the *Validation Loss*. Assuming the goal of a training is to minimize the validation loss. With this, the metric to be monitored would be ”val loss”, and mode would be ”min”. A training loop will check at end of every epoch whether the loss is no longer decreasing, considering the “min_delta” (minimum change in the monitored quantity to qualify as an improvement) and patience if applicable. Once it’s found no longer decreasing, the training terminates.[23] In this point it would be useful to illustrate the difference between the Epoch and the Batch. In fact, the batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters, and in all models in this thesis was taken equal to 256. The number of epochs, as already mentioned, is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

Chapter 6

Training & Testing Results

This chapter includes both the training and testing results of the FFNN and TDNN models, designed to predict the following values:

- Fuel Consumption (lt/h)
- MAP (Intake Manifold Absolute Pressure) (kPa)
- Air-fuel ratio equivalence factor - Lambda (λ)
- NOx (ppm)

For the development of these models different combinations of inputs have been selected, according to the nature of each output value. The available data sets have been partitioned into three parts (Training, Validation & Testing), as already mentioned. It is worth reminding that, training data is for parameters (weights and bias) training, validation data is used to monitor the network performance during the training process and avoid overfitting and test data, finally, are not involved in the training process and thus can be used to validate the model performance after the training is finished.[7] In addition, there are some extra -unknown to the models- data sets which have been used only for testing and their results will be presented separately.

6.1 Procedure Flowchart

Before presenting the predictions results of the models it would be quite helpful and informative to include a schematic representation of the modeling process. The following flowchart summarizes the whole model development process, applied in this thesis. As aforementioned, there is a possibility of altering the Model Inputs if the performance is poor and does not improve with an other combination of hyperparameters, but this case was not included as a "Decision" in the flowchart.

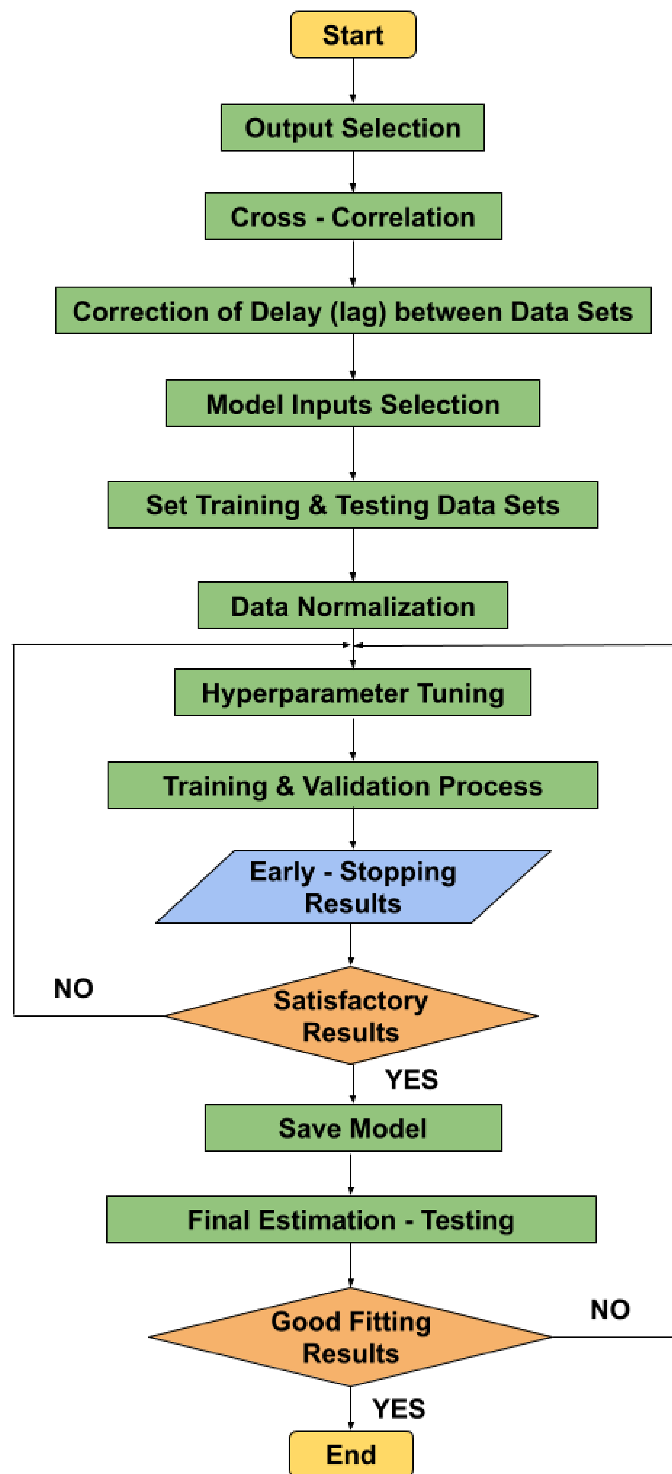


Figure 6.1: Flowchart of Artificial Neural Networks Modeling Process, in this study.

6.2 Fuel Consumption Models

6.2.1 FFNN Model

6.2.1.1 Training Results

First of all, after the pre-processing of the available data set, the next step is called Exploratory Data Analysis (EDA), including tools and methods for data visualization and further decoding. One of the most effective and useful tools is the **pairplot** function of Seaborn library, which visualizes pairwise relationships between the multiple variables. In the pairs plot, also called a scatterplot matrix, the diagonal shows the univariate histograms of the individual columns, while the scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables.[26] The Figure 6.2, below, shows the multiple pairwise bivariate distributions in the data set used for Fuel Consumption prediction.

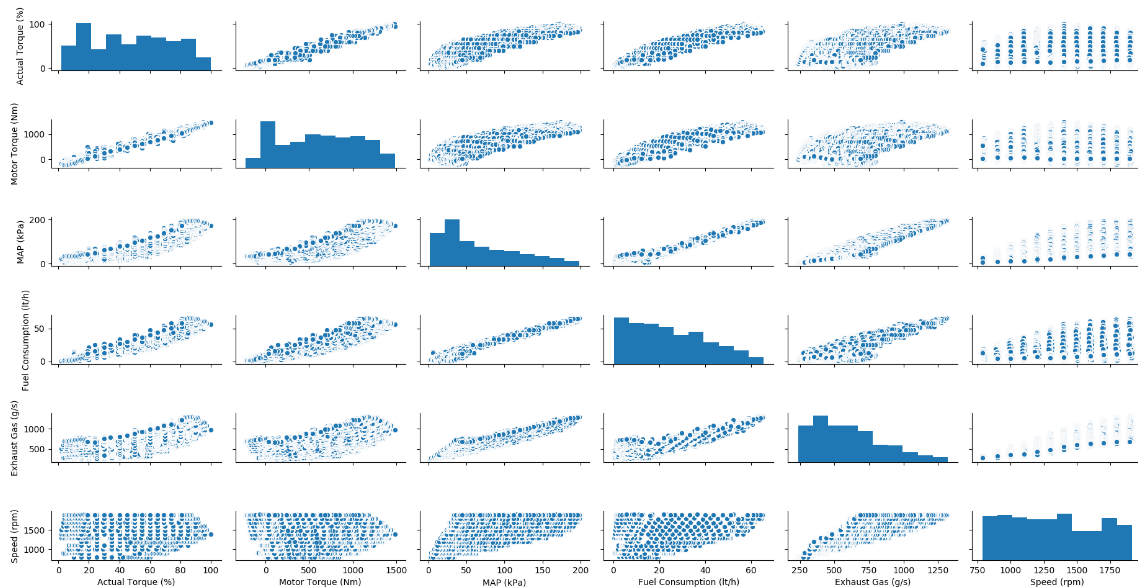


Figure 6.2: Pairwise Distributions

As per Figure 6.2, the majority of variables are "linearly connected", except for the *Speed* parameter which is distributed as a step function, according to Figure 4.5. The linear relationship (or linear association) between the various inputs facilitates the training of the neural network, because there are no "difficult" or complex cases for the model to learn. As reported earlier, the quantity and correlation of inputs play a major role to both network training and testing process, but it does not mean that a model with no linearly associated inputs will perform badly.

An other visualization technique, part of EDA, is the **heatmap** function of Seaborn. A heatmap is a two-dimensional graphical representation of data, where the individual values that are contained in a matrix are represented as colors. In this case, this function was used in order to visualize the pairwise correlation of variables in the data set. Generally speaking, correlation examines and quantifies the relationship between two variables, or sets of data. A typical method of measuring correlation is the Pearson Correlation Coefficient (PCC),

which ranges from -1 to +1. Whether the PCC is positive or negative indicates whether the relationship is a positive correlation (i.e. as one variable increases, the other variable generally increases as well) or a negative correlation (i.e. as one variable increases, the other variable generally decreases). The absolute value of the PCC indicates the strength of the relationship, where the closer it is to 1 the more strongly related the two variables are, while a PCC of 0 indicates no relationship whatsoever.[27]

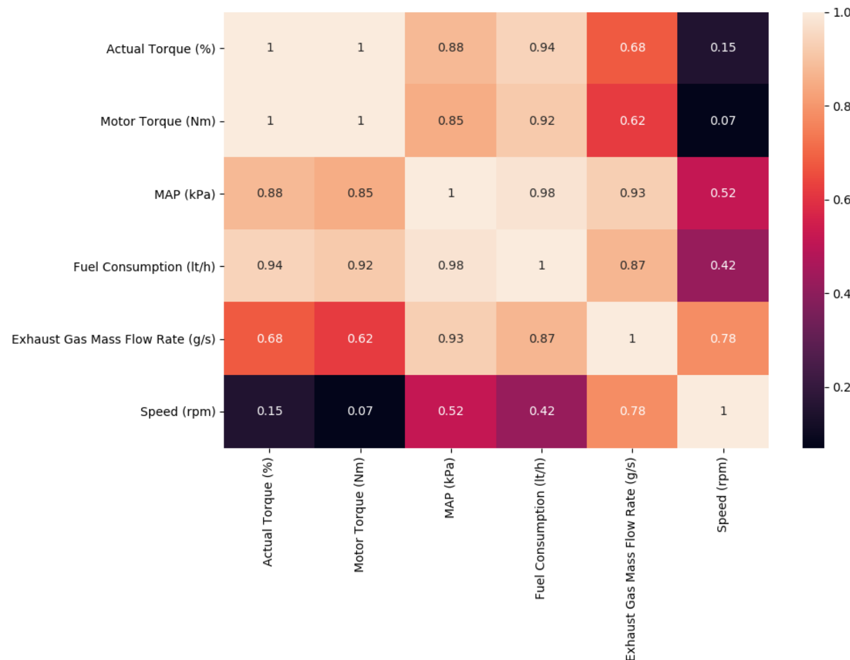


Figure 6.3: Heatmap of Pearson Correlation Coefficient

According to Figure 6.3, only positive correlations have been calculated so all variables are proportionally connected, with the *Speed* unit to have the "weakest" relationship with the majority of the rest inputs.

For the first model of ***Fuel Consumption*** prediction 3 fully connected (Dense) hidden layers have been used. There are 15 neurons in the first two layers, with a sigmoid activation function and 1 neuron in the last layer, with no activation, so the returned value will be the same with the output. The last layer is structured in that way because the model solves a regression problem with arbitrary values and the last activation should be linear. The model structure is automatically plotted by a function in Keras, with the following format.

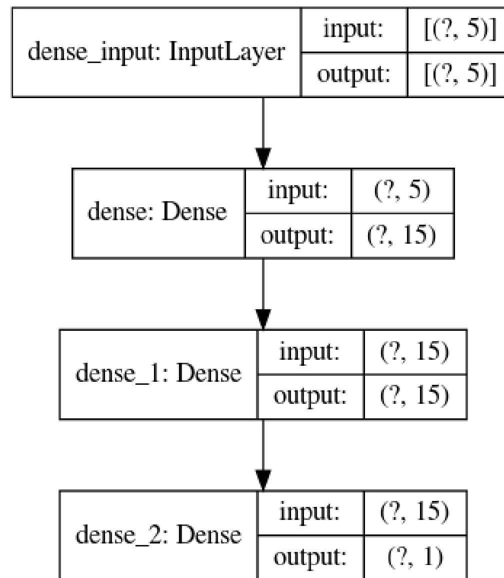


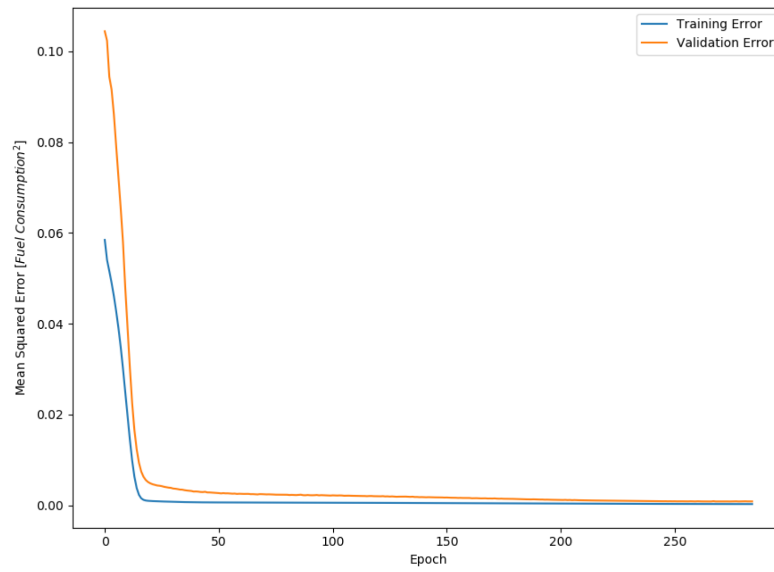
Figure 6.4: Model Layout automated by Keras

The "?" symbol in Figure 6.4 means that the model is constructed in order to accept inputs of any dimension, i.e. input: [(?,5)] indicates 5 vectors with unknown length (number of samples) as input. In that way the model becomes adaptive to various input dimensions, especially during testing process where the number of samples per input may constantly change. The aforementioned structure forms a total of 346 trainable parameters (weights & nodes), which is approximately the 1/10 of the 24,575 training data points.

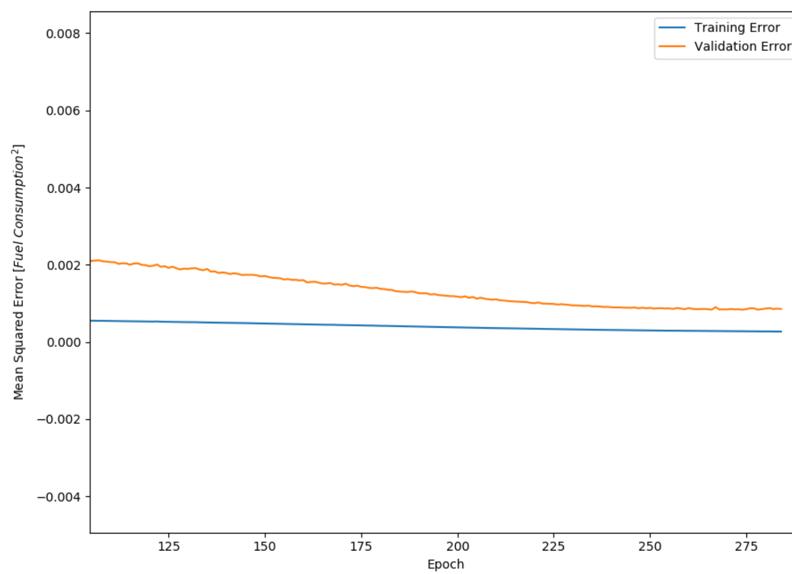
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 15)                90
_________________________________________________________________
dense_1 (Dense)              (None, 15)                240
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 16
=================================================================
Total params: 346
Trainable params: 346
Non-trainable params: 0
```

Figure 6.5: Algorithm Output showing Total Trainable Model Parameters

The training process was iterated for almost 280 epochs with Adamax optimization and monitoring of the Mean Squared Error metric as validation loss. As far as the Early Stopping Callback is concerned, it has to terminate the training if for more than 10 consecutive epochs there is no improvement to the validation loss, that is to increase or remain the same. The visualization of the training process is achieved through the visualization of selected evaluation metrics, which quantify the performance of the neural network. Figures 6.6 and 6.7 depict the progress of the two most representative error-loss functions of the model.

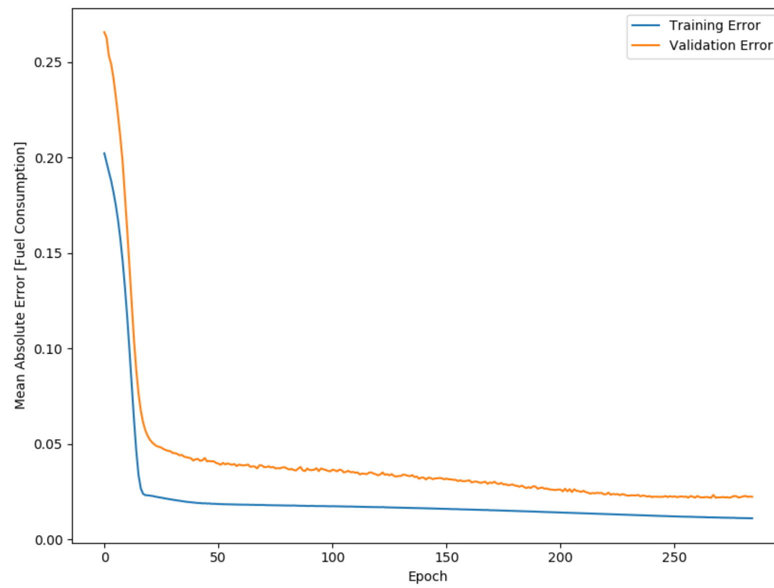


(a) Mean Squared Error during all epochs.

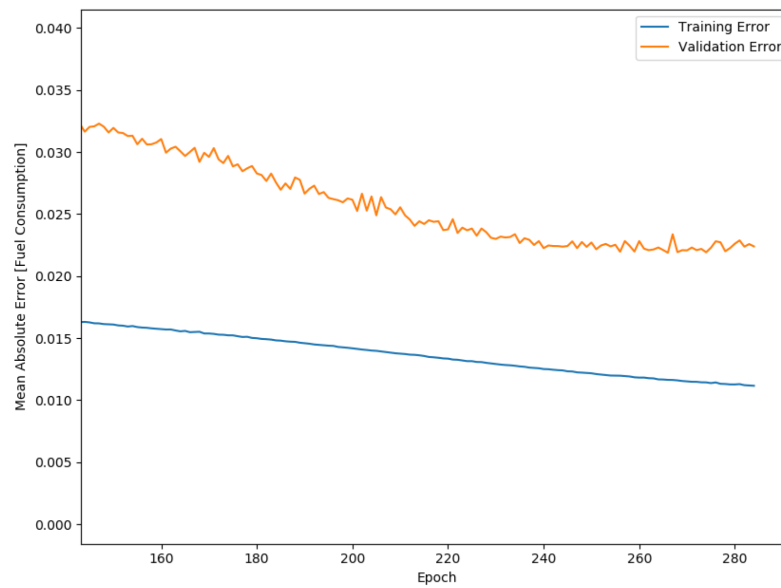


(b) Zoomed region showing the "smooth" variation of validation error.

Figure 6.6: Mean Squared Error during training



(a) Mean Absolute Error during all epochs.



(b) Zoomed region showing the variation of validation error.

Figure 6.7: Mean Absolute Error during training

According to the above figures, in the beginning of the training session both errors obtain large values that in less than 25 epochs are abruptly reduced. This happens because the model has just started to be trained without learning for an adequate number of epochs for the specified batch size. Also, after the sharp decrease, in both figures the training and validation errors adopt an almost linear distribution which slowly goes down and finally terminates approximately at the 280th epoch, which is obvious in the Figure 6.7b, where the overfitting has started with the increase of validation error and the continuous decrease of the training error. Also, the fact that training error slightly underestimates the validation and finally the two errors converge demonstrates a case of a good fit. Due to the loss of the model being lower on the training data set than the validation data set, a gap between the train and validation loss learning curves is created and called "generalization gap." Consequently, the plot of learning curves shows a good fit because:

- The plot of training loss decreases to a point of stability.
- The plot of validation loss decreases to a point of stability and has a small gap with the training loss.

Last but not least, the MSE shows a smoother distribution than the MAE because of the difference in scaling between the two loss types and the higher values of MAE plotted with larger steps on y-axis. It is worth reminding that these errors correspond to the training process where all the values are normalized in the range of (0,1), so they are not accompanied with a realistic unit measurement.

Finally, Figure 6.8 shows the fitting result of the neural network with one of the best performances. An outlier point can be seen near (0.2,0.0) coordinates but it is inconsiderable in comparison to the total number of samples, thus it does not affect at all the accuracy of the model.

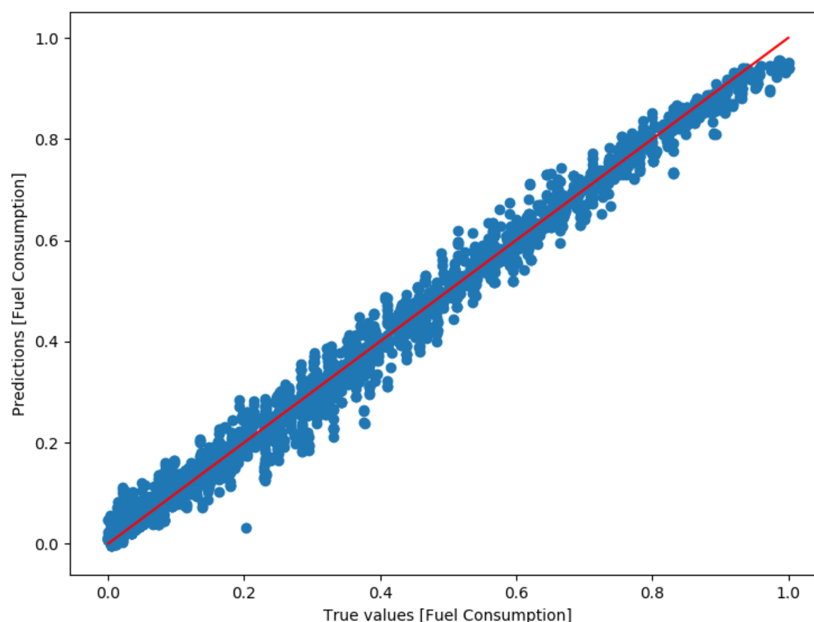


Figure 6.8: Model Testing: Predicted vs Original Data

As a sequence to Figure 6.8, the following figures depict the prediction of a testing data set which is extracted from the same available data samples as training data set. In that way, even if the data have been randomly selected, they still have a "familiar" distribution to the training data set and that enables the neural network to predict the values with such an accuracy. In fact, the largest deviation shown in the zoomed area of Figure 6.9b is less than 5 lt/h which is translated to less than 10% error. Also, the most important thing is that the Predicted Values follow the trend of the Original Values without a delay in x-axis (Samples-"Time" delay) but with just a slight difference in the y-axis values as expected, otherwise the network would be perfect. The error between the predicted and the real values is expressed below, by the most typical and commonly accessible error functions:

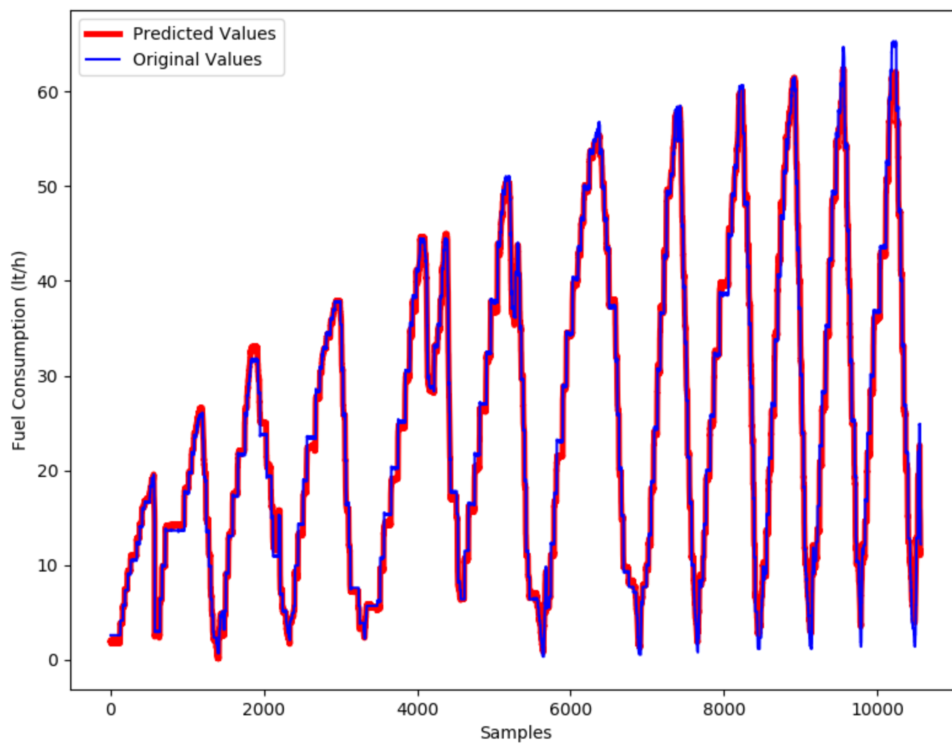
Testing Errors:

- MSE = 1.6209 $(lt/h)^2$
- MAE = 0.8756 lt/h
- MAPE = 8.2450 %

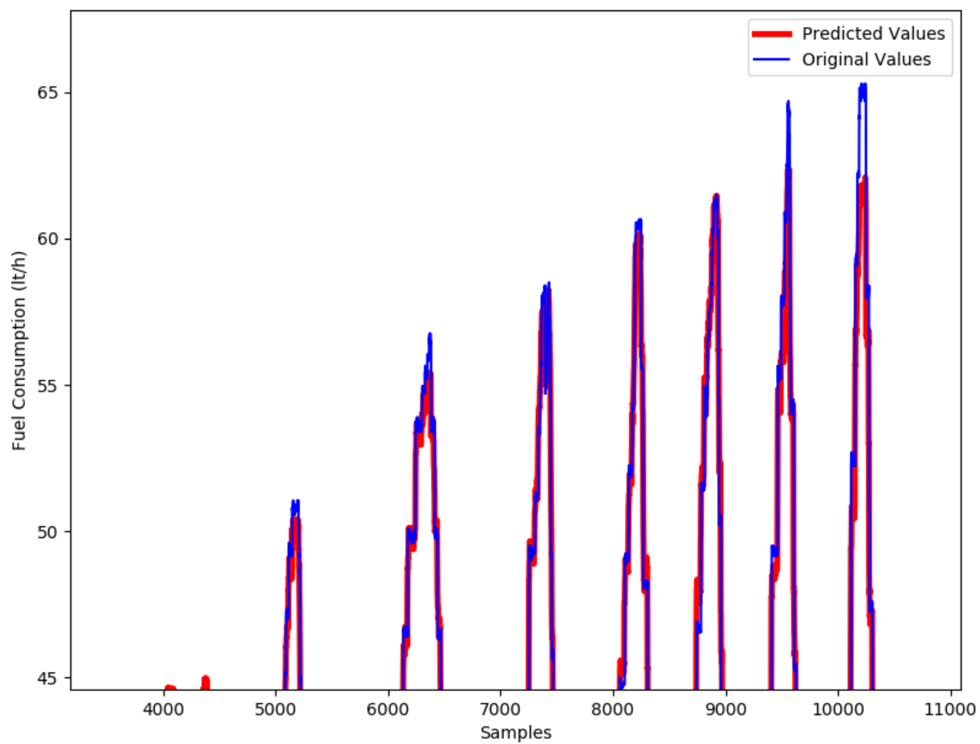
Finally, a table which summarizes all the characteristics of the explained model is presented.

Table 6.1: Fuel Consumption FFNN Hyperparameters & Characteristics

Inputs	Actual Torque (%) Motor Torque (Nm) Speed (rpm) MAP (kPa) Exhaust Gass Mass Flow Rate (g/s)
Hidden Layers	3
Nodes Per Hidden Layer	15-15-1
Total Trainable Parameters	346
Activation Function	Sigmoid-Sigmoid-Linear
Optimizer	AdaMax (Learning Rate = 0.0005)



(a) Results of the fuel consumption predictions.



(b) Zoomed region showing the "deviation" between the predicted and the original values.

Figure 6.9: Trajectories of measured and predicted fuel consumption values.

6.2.1.2 Testing Results

After that, the model is tested on completely unknown data in order to examine its ability to handle unseen test data, completely unrepresentative of the training data set. The measured signals used as the unknown test set are shown in Figure 6.10 and follow the same trend. In comparison with the training data set which cover the whole envelope of the test-bed (the speed increases from 900 to 1900 rpm with a step of 100 rpm), the testing data set represents a transient augmentation of load so the speed reaches approximately the 1600 rpm from the starting point of 900 rpm. These data have been produced by the experiments of study [20], with transient propeller load.

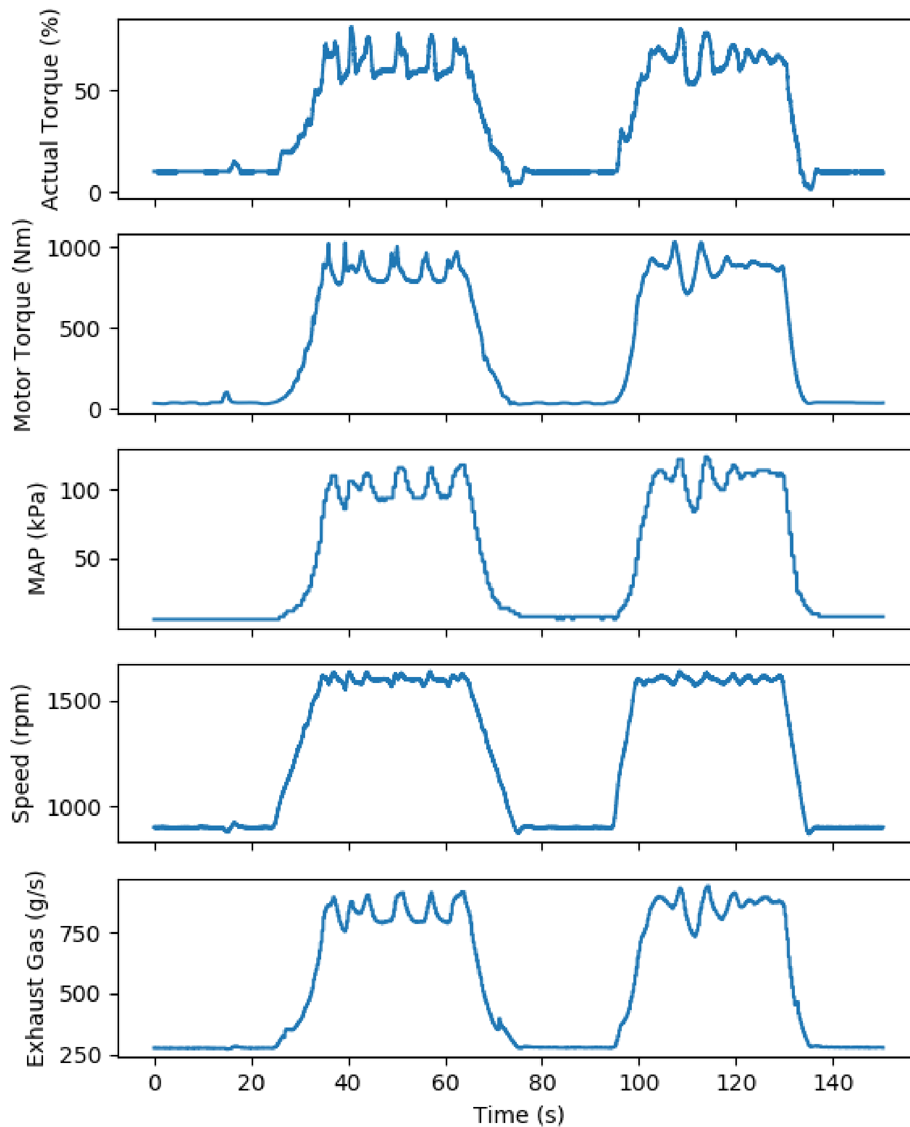


Figure 6.10: Testing Data - Inputs

The fitting results are more than adequate as both the following figures and the individual errors imply.

- MSE = 2.7672 $(lt/h)^2$
- MAE = 1.0523 lt/h

It is worth mentioning that this time the Mean Absolute Percentage Error (MAPE) could not be calculated. In general, MAPE function can be problematic whenever there is a division by zero or a value close to zero, because the actual value is in the denominator. In that way, it may produce unrealistic results or an infinitely large value, labeled as "inf".

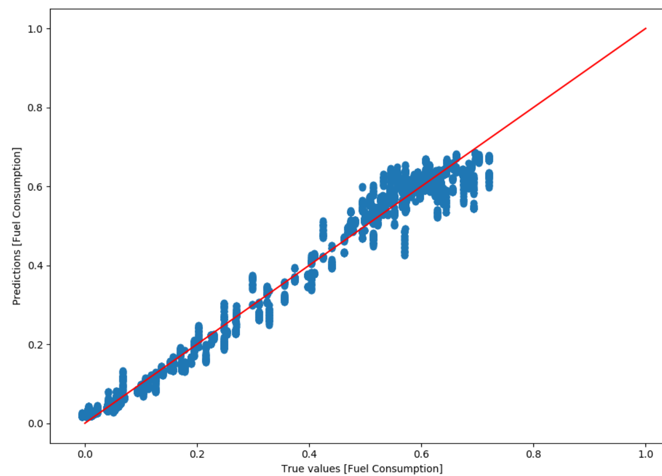


Figure 6.11: Model Testing in Unknown Data: Predicted vs Original Data

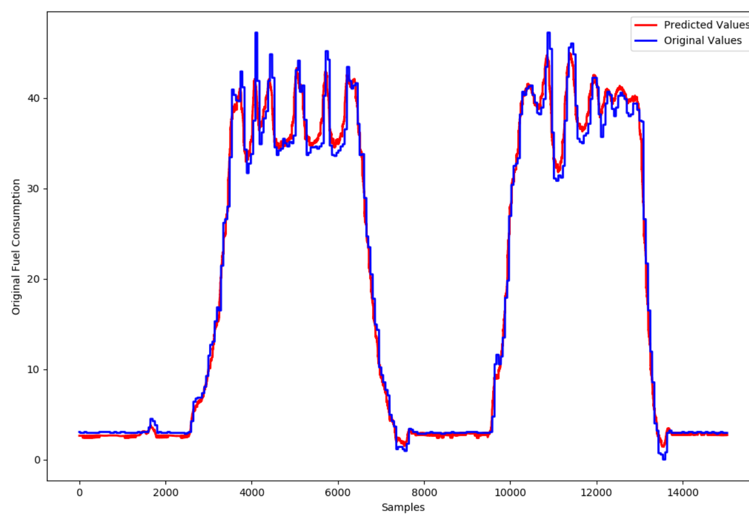
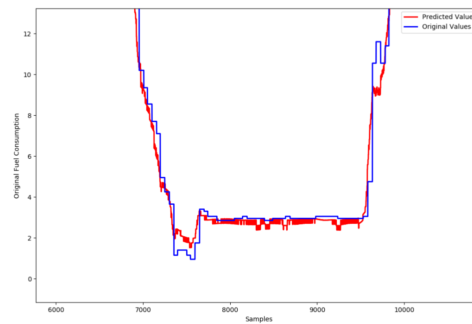
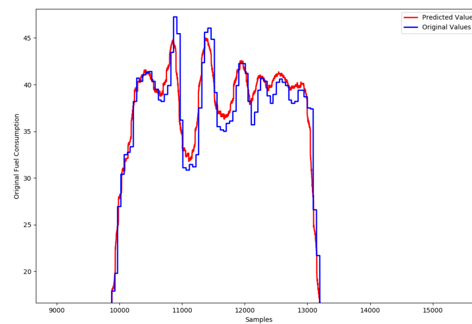


Figure 6.12: FFNN - Trajectories of measured and predicted fuel consumption values, for the unknown data.



(a)



(b)

Figure 6.13: Various zoomed regions showing the "deviation" between the predicted and the original values.

Taking a closer look at the subfigures of Figure 6.13, the predicted values are almost identical to the original and both the slope and the noisy regions of the "Original Values" curve are predicted with high precision.

6.2.2 TDNN Model

As far as the prediction of Fuel Consumption with a TDNN is concerned, the input values are extracted from the same data set but this time each input has a delay of 0.1, 0.2 and 0.3 seconds. For instance, the data set of "Speed" exists 4 times as an input, without a delay, with a delay of 0.1 sec, of 0.2 sec and of 0.3 sec. The step of the delay may seem insignificant from time perspective in real life, but in the world of signal processing and model predictions can make a difference.

6.2.2.1 Training Results

For a TDNN the creation of either a graph with *Pairwise Distributions* or a *Heatmap of Pearson Correlation Coefficient* would make no sense due to the large number of inputs to the neural network. For the prediction of Fuel Consumption, $5 \times 4 = 20$ inputs are used, and except for the undecipherable graphs and matrices, the correlation between these inputs would be quite similar to the one depicted in Figures 6.2 and 6.3, where the units are unique.

This model is more complicated than the FFNN model, because of the 4 times bigger input *DataFrame*. For this reason, its layout should be simpler and with less neurons per hidden layer in order to be easily trained. Accordingly, 3 fully connected (Dense) hidden layers have been also used. There are 2 neurons in the first two layers, with a sigmoid activation function and 1 neuron in the last layer, with no activation, so the returned value will be the same with the output. The total trainable parameters are 51 and the model structure is automatically plotted by a function in Keras, with the following format.

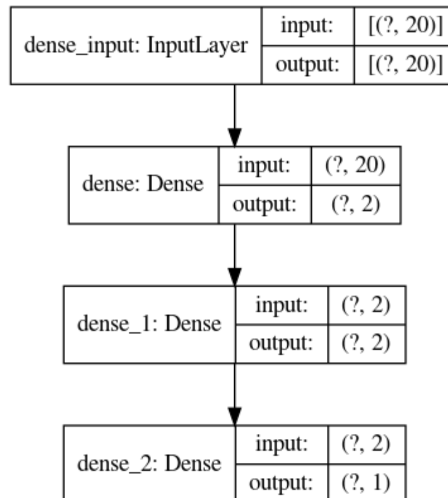


Figure 6.14: Model Layout automated by Keras

The training process was iterated for 283 epochs, really close to the number of FFNN, with Adamax optimization and monitoring of the Mean Squared Error metric as validation loss. The visualization of the training process is achieved through the Figures 6.15 and 6.16, which depict the progress of the Mean Squared Error (MSE) and Mean Absolute Error (MAE) respectively, the two most representative error-loss functions of the model.

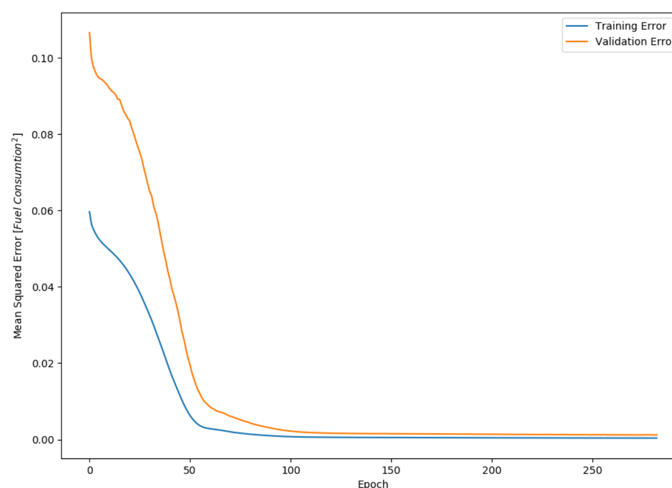


Figure 6.15: Mean Squared Error during training.

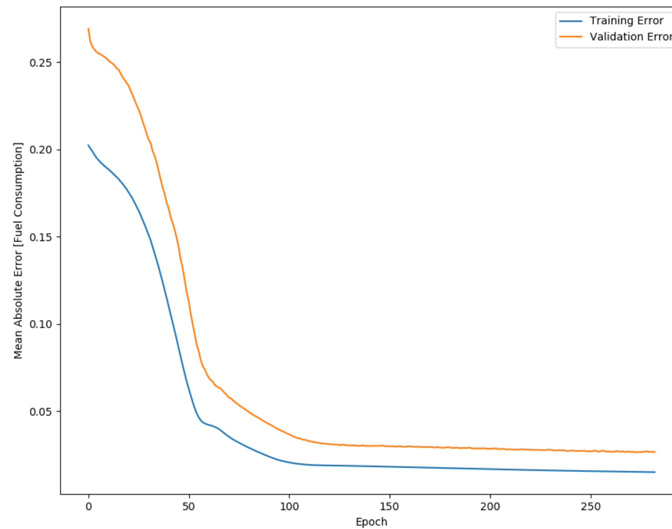


Figure 6.16: Mean Absolute Error during training.

In general, after the 100th epoch of training both errors have very similar trends with those of FFNN model. The different distributions in the first 100 epochs and the larger slopes of the curves imply that the TDNN model has a bigger "learning period". Nevertheless, this is not a problem because the initial "large" gap between the curves shrinks, the learning curves converge and they finally reach very low levels of errors.

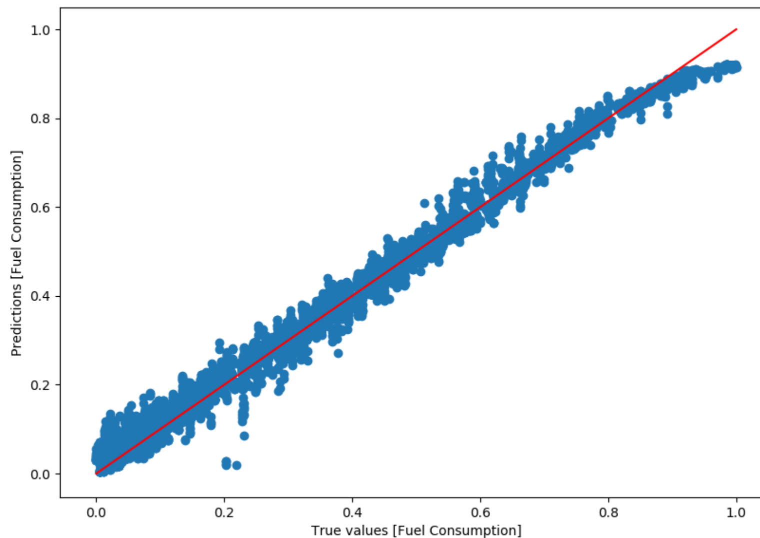


Figure 6.17: Model Testing: Predicted vs Original Data

The fitting result of the neural network with one of the best performances is shown above, in Figure 6.17, where the aforementioned outlier point did not disappear but multiplied. In this case there are more outlier points near the (0.2,0.0) point coordinates, but

again they are inconsiderable in comparison to the total number of samples. Also, the larger Fuel Consumption values are not so successfully predicted, according to the deviation of points from the y=x line in Figure 6.17. This ”failure” can also be seen in Figure 6.20b, where the difference between the original and the predicted values is approximately 5 lt/h, right after sample 9,500 and between 10,000 and 10,500. However, such deviations are expected and found in specific points or regions and not in the whole range of data.

As a sequence to Figure 6.17, the following figures depict the prediction of the TDNN model, where both the training and testing data sets are extracted from the same available data samples in a random way, as in FFNN case. Again, the Predicted Values follow the trend of the Original Values without a delay in x-axis (Samples-”Time” delay) but with just a difference in the y-axis values, which is slightly bigger than the corresponding difference in FFNN, as in details of Figure 6.20a. Finally, the errors between the predicted and the real values and a summarizing table of model layout are presented below.

Table 6.2: Comparison between FFNN & TDNN predictions on testing data from the same data set of training inputs.

	FFNN	TDNN
MSE ($(lt/h)^2$)	1.6209	2.3345
MAE (lt/h)	0.8756	1.1252
MAPE (%)	8.2450	13.7614

Table 6.3: Fuel Consumption TDNN Hyperparameters & Characteristics

Inputs (×4)	Actual Torque (%) Motor Torque (Nm) Speed (rpm) MAP (kPa) Exhaust Gass Mass Flow Rate (g/s)
Inputs Delays	0.1, 0.2 & 0.3 sec
Hidden Layers	3
Nodes Per Hidden Layer	2-2-1
Total Trainable Parameters	51 (42-6-3)
Activation Function	Sigmoid-Sigmoid-Linear
Optimizer	AdaMax (Learning Rate = 0.0005)

Except for the error calculation methods and the comparative graphs of original and predicted data, there is also an other alternative for demonstrating the performance of a model. This alternative is the ”Error Histogram” plot, which shows the distribution of the errors between target and predicted values after model fitting, as in Figure 6.18. In such a procedure the distribution of errors, being by its nature continuous, is treated as a discrete (quasi-step) distribution. It is implicitly assumed that the errors are of a finite selected number of magnitudes, instead of assuming all practically possible values of the entire range.[28] As these error values indicate the difference between the predicted and original values, they can be negative. The x-axis represents the various, selected values of errors and y-axis represents the number of samples in the data set. Each bar indicates how many samples of the data have been predicted with the corresponding error of the x-axis.

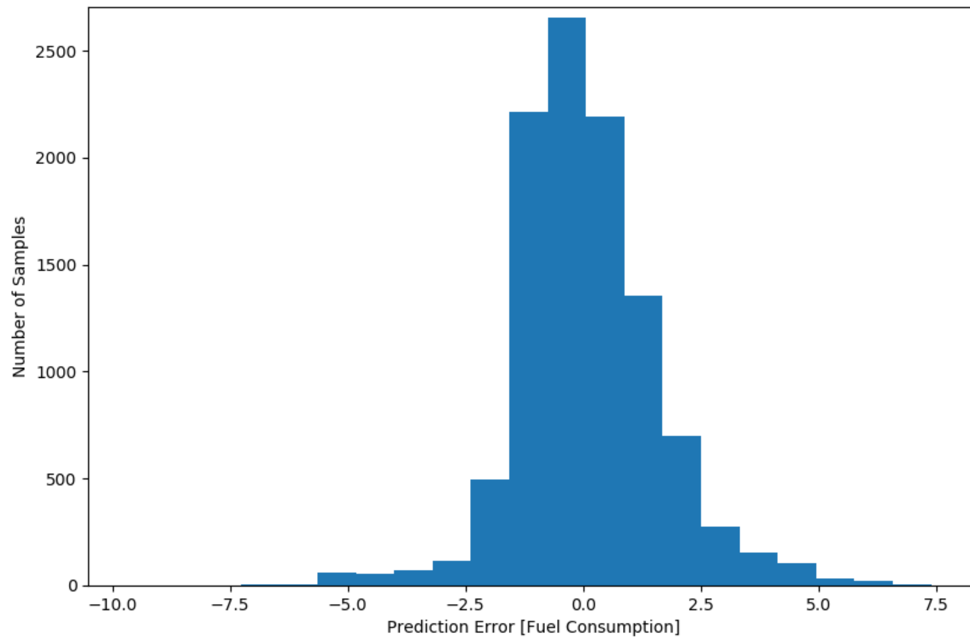


Figure 6.18: Error Histogram, where values of x-axis correspond to *Error=Targets-Outputs*

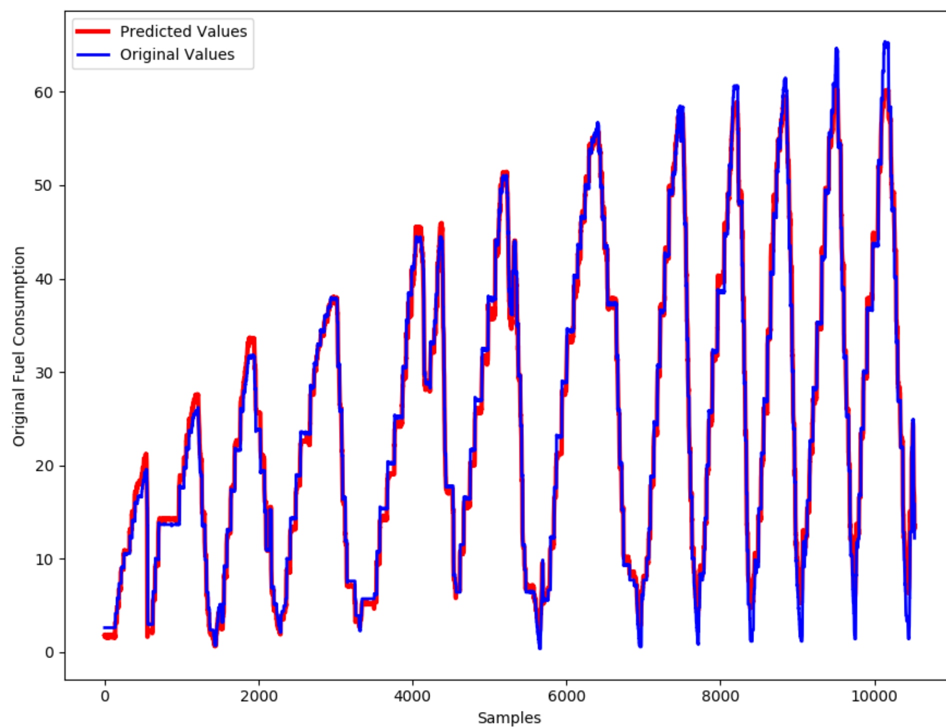
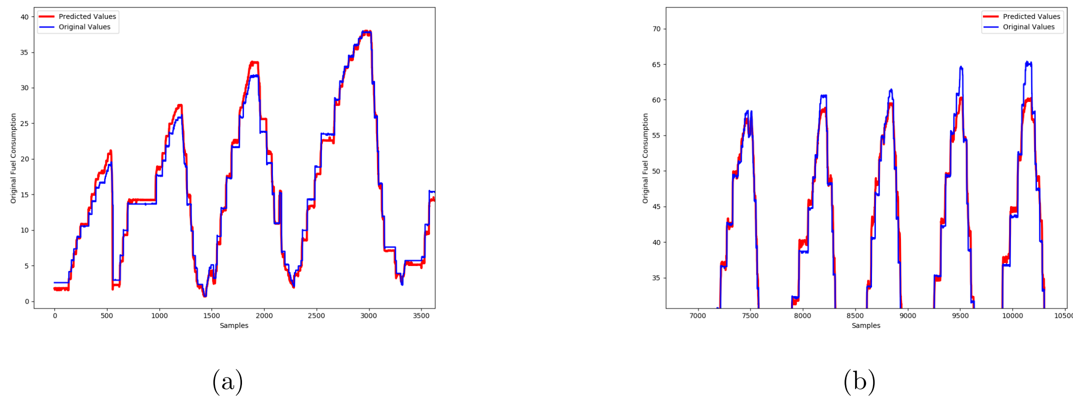


Figure 6.19: Trajectories of measured and predicted fuel consumption values.



(a) (b)

Figure 6.20: Zoomed regions showing the ”deviation” between the predicted and the original values.

6.2.2.2 Testing Results & Comparison with FFNN Performance

After that, the TDNN model is tested on the same unknown data set as FFNN, 6.10, for extra validation and evaluation.

The fitting results again are more than adequate as both the following figure and the individual errors imply.

- MSE = 1.2022 $(lt/h)^2$
- MAE = 3.1445 lt/h

At this point there is no interest in analyzing the behavior of the prediction curve in Figure 6.22 independently, hence there are no screenshots with details and ”zoomed” areas provided. Having said that, it would be of great importance to compare the predictions of the two different neural networks for the same curve, and observe the details and the differences between them.

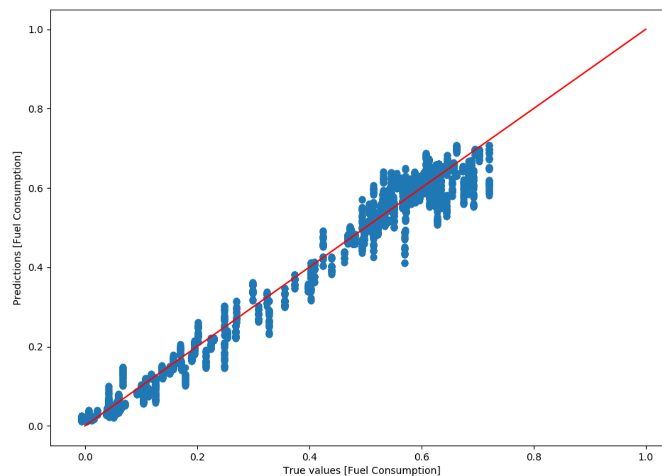


Figure 6.21: Model Testing in Unknown Data: Predicted vs Original Data

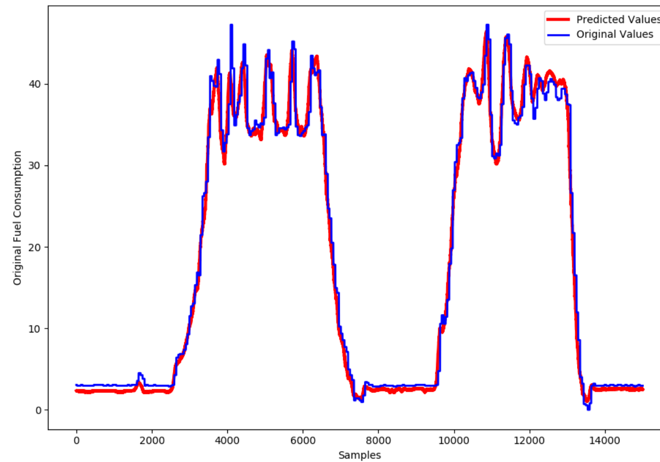


Figure 6.22: TDNN - Trajectories of measured and predicted fuel consumption values, for the unknown data.

Before presenting the concluding graph of the Fuel Consumption modeling, it is worth mentioning that the TDNN model predictions have larger error values than FFNN, tested with the same input data set, as it can be also seen in the Table 6.4.

Table 6.4: Comparison between FFNN & TDNN predictions on unknown testing data

	FFNN	TDNN
MSE (lt/h)2	2.7672	3.1445
MAE (lt/h)	1.0523	1.2022

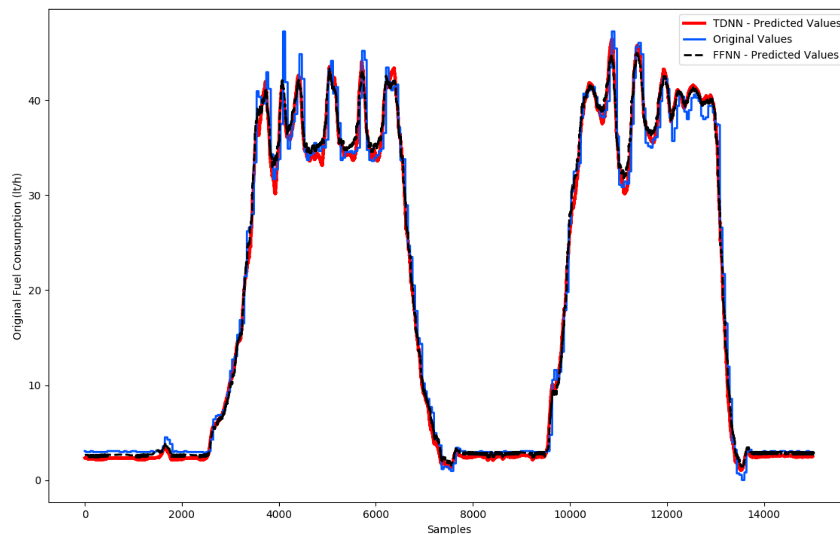
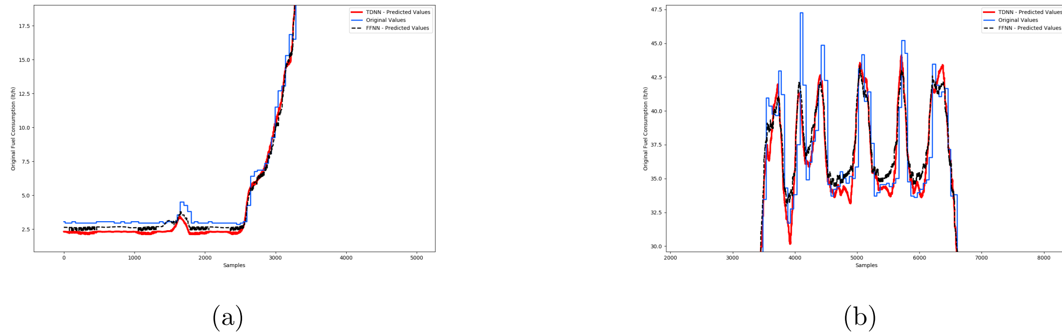


Figure 6.23: Final comparison between experimental Fuel Consumption (Original Values), FFNN and TDNN trajectories.



(a) (b)

Figure 6.24: Various zoomed regions showing the "deviation" between the FFNN & TDNN predicted and the original values.

Figure 6.24 shows an overview of the successful prediction of the unknown validation data set both by FFNN and TDNN. In more detail, Figure 6.24a implies that in the almost steady fuel consumption values the FFNN predictions get closer to the original values, and that somewhat could explain the larger error of the TDNN. Also, Figure 6.24b shows the overall ability of the FFNN to predict more accurately the fuel consumption curve "jumps", except for the sharpest one around 4,000 samples and 47.5 lt/h region.

6.3 MAP Models

6.3.1 FFNN Model

6.3.1.1 Training Results

The unit of MAP follows the same pattern as that of Fuel Consumption, but with different scaling. For its prediction less inputs were chosen and the relationships between them and the output (MAP) are mostly linear or somewhat parabolic. Of course, the Speed unit is the exception, because all of its pair plots with other units represent the working area of the experiment. Consequently, the Pearson Correlation Coefficients between the inputs and the output are almost similar with those of Fuel Consumption FFNN model, due to the same measurements used in both models. It is worthwhile noting that the models for Fuel Consumption predictions were firstly created and in an attempt to have more inputs. However, it was noticed that also less inputs could have produced good results, so with this idea in mind for the MAP models only 3 inputs were selected. The aforementioned information is depicted in the following figures.

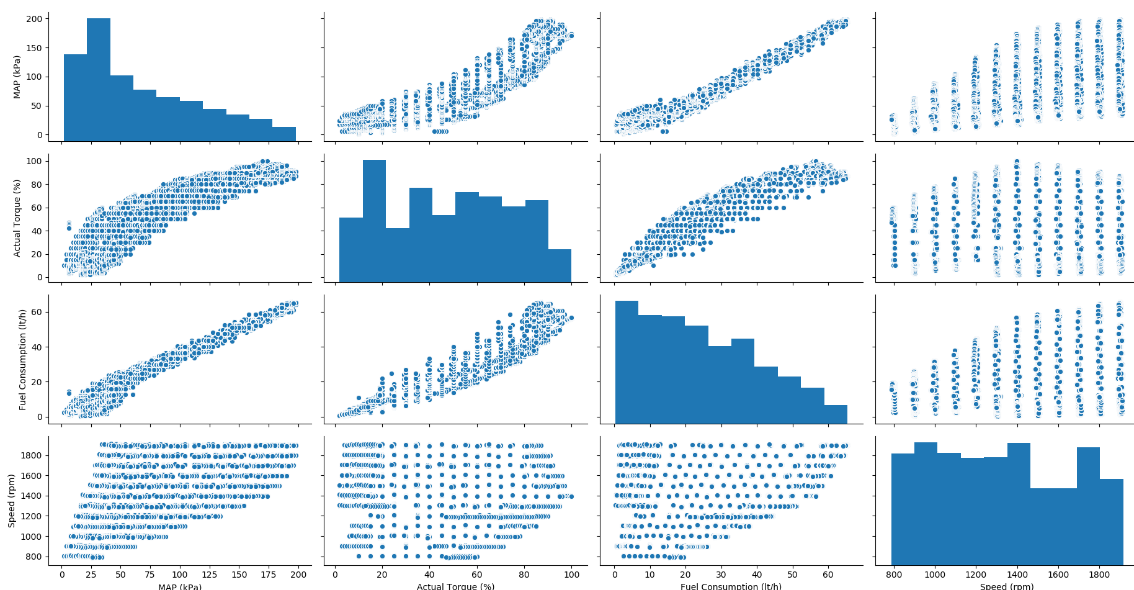


Figure 6.25: Pairwise Distributions

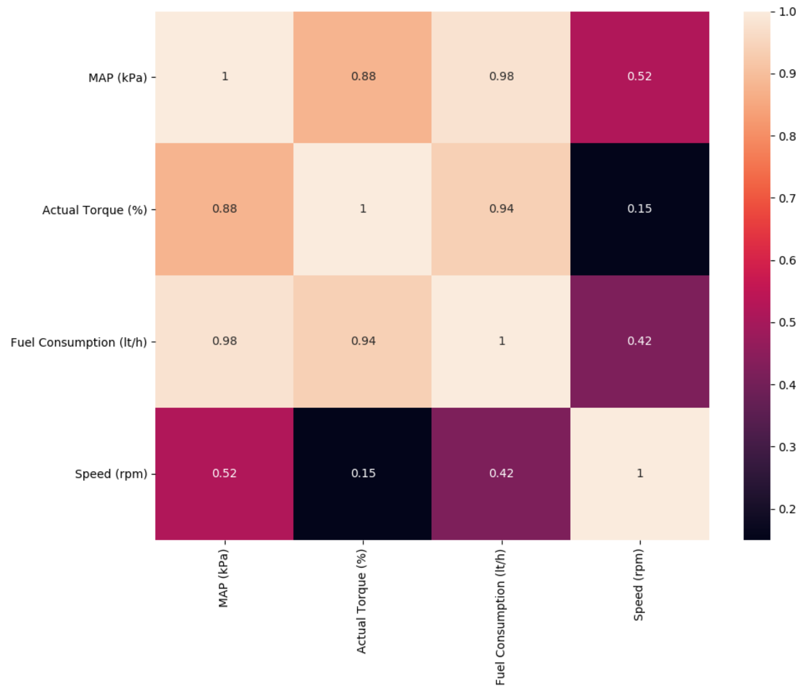


Figure 6.26: Heatmap of Pearson Correlation Coefficient

For the FFNN model of MAP prediction 3 fully connected (Dense) hidden layers were used, with 15 neurons in the first two hidden layers and the ReLU activation function and 1 neuron in the last hidden layer with no activation function. The network layout and its hyperparameters & trainable parameters are presented below.

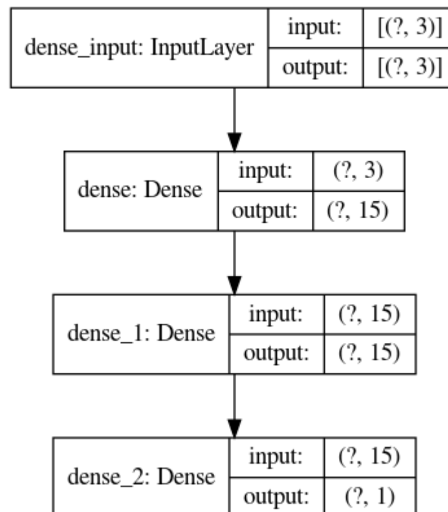


Figure 6.27: Model Layout automated by Keras

Table 6.5: MAP FFNN Hyperparameters & Characteristics

Inputs	Actual Torque (%) Speed (rpm) Fuel Consumption (lt/h)
Hidden Layers	3
Nodes Per Hidden Layer	15-15-1
Total Trainable Parameters	316 (60-240-16)
Activation Function	ReLU-ReLU-Linear
Optimizer	AdaMax (Learning Rate = 0.0005)

Even though the training process lasted for almost 50 epochs, less than in the previous model, the overall image of training and validation error in both metrics graphs denotes a good fitting result. The two MSE curves are really smooth and finally come too close to each other after the 20th epoch, whilst the MAE curves seem to have a bigger gap between them, without however being a problem according to the error scale on y-axis.

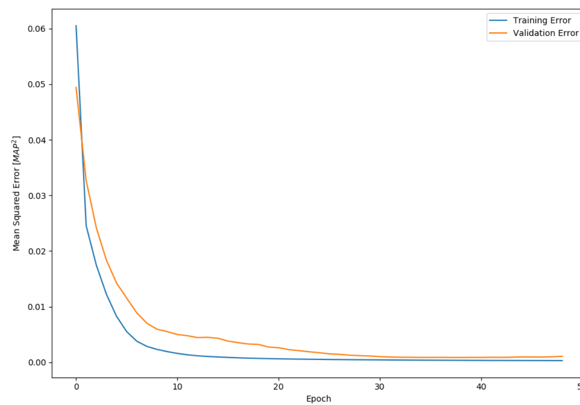


Figure 6.28: Mean Squared Error during training.

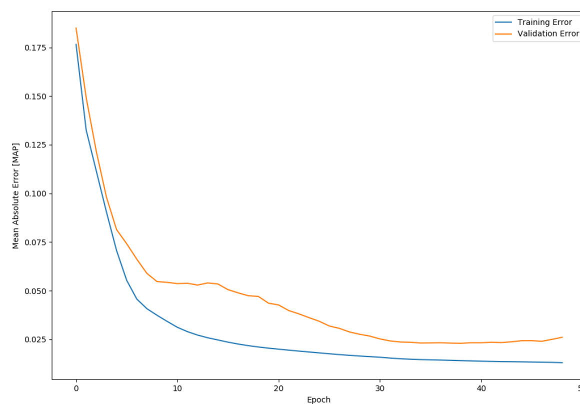


Figure 6.29: Mean Absolute Error during training.

The aforementioned "good fitting" becomes more obvious when plotting the scatter plot of the real and predicted values, where the majority of points are fitted ideally, with just some insignificant outliers which do not affect at all the model performance.

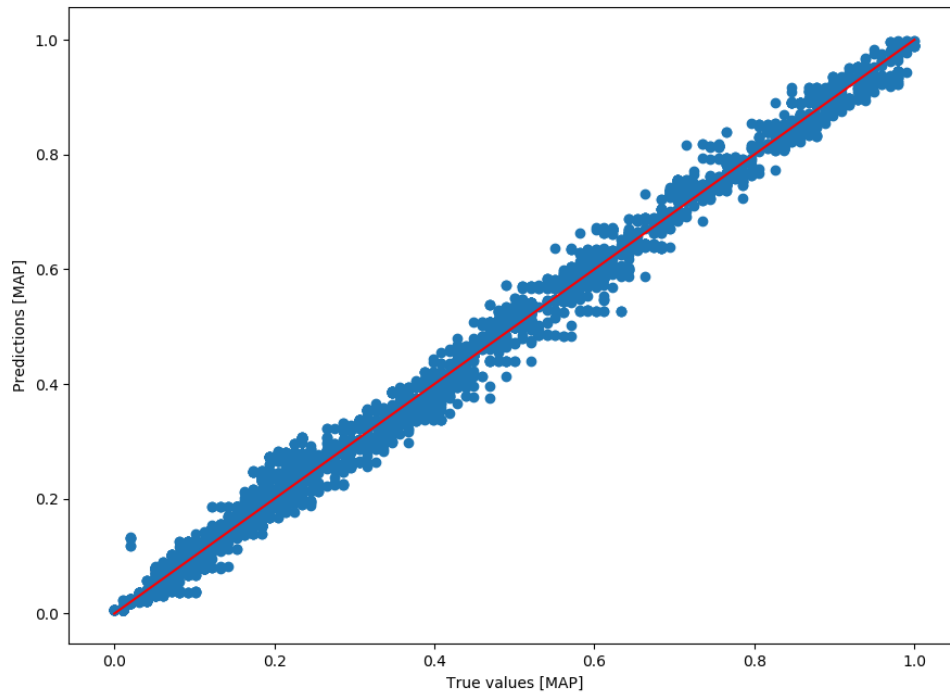
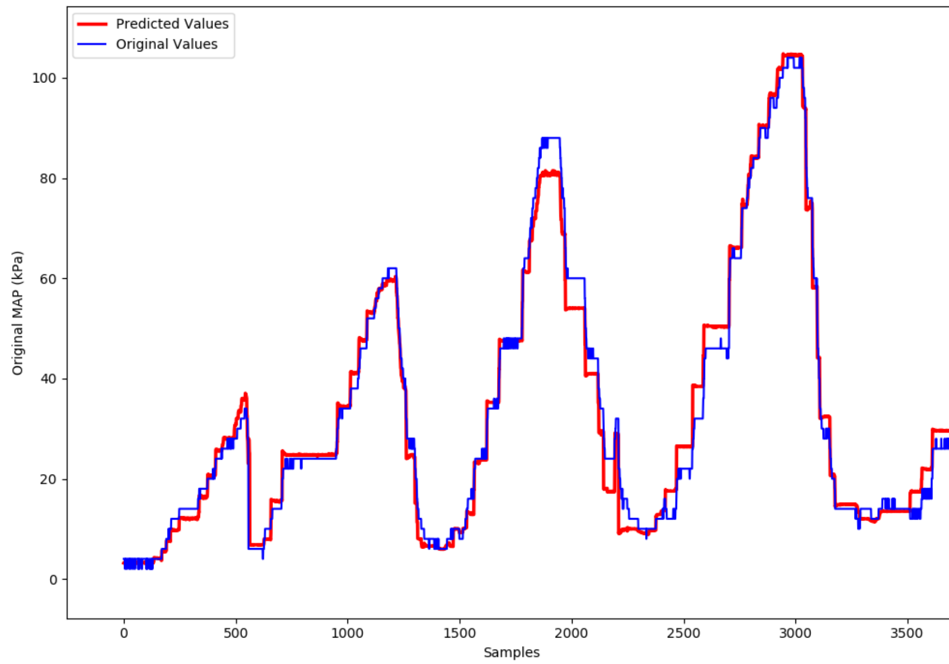


Figure 6.30: Model Testing in Unknown Data: Predicted vs Original Data

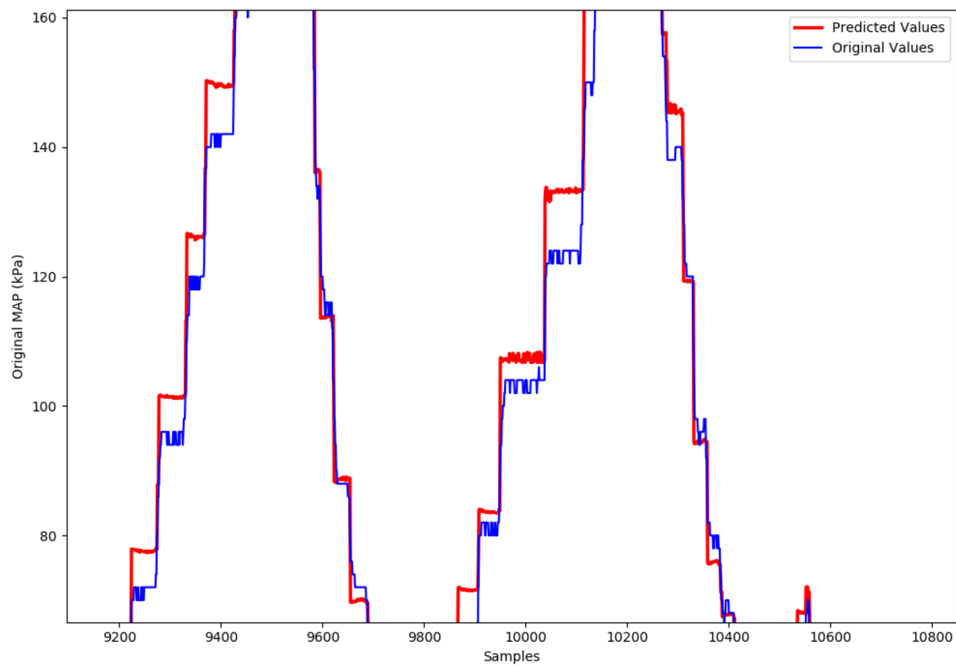
The implemented testing data set was the 30% of the initial whole data set, which functions as the main "data storage" for the problem. The combination of the testing and training data sets can be seen in Figure 6.31a, which seem to converge almost perfectly. There are only some vertical deviations, for example in Figure 6.31b, which are less than 5 kPa and does not bother the model to follow the MAP dynamics. Moreover, even the highest (peaks) or lowest (valleys) values, which are commonly the hardest to fit, are well predicted.

Testing Errors:

- MSE = 17.6313 kPa^2
- MAE = 0.1781 kPa
- MAPE = 7.3557 %



(a)



(b) Zoomed region showing the "deviation" between the predicted and the original values.

Figure 6.31: Results of MAP predictions.

6.3.1.2 Testing Results

After that, the model is tested on completely unknown data. The measured signals used as the unknown test set are shown in Figure 6.32 and all of them follow the same trend. In fact, they are the same with those of Figure 6.10 because all the training and testing processes should be done using common data sets, in order to be the "common denominator" of the predictions and enable the comparison between the results of various model types.

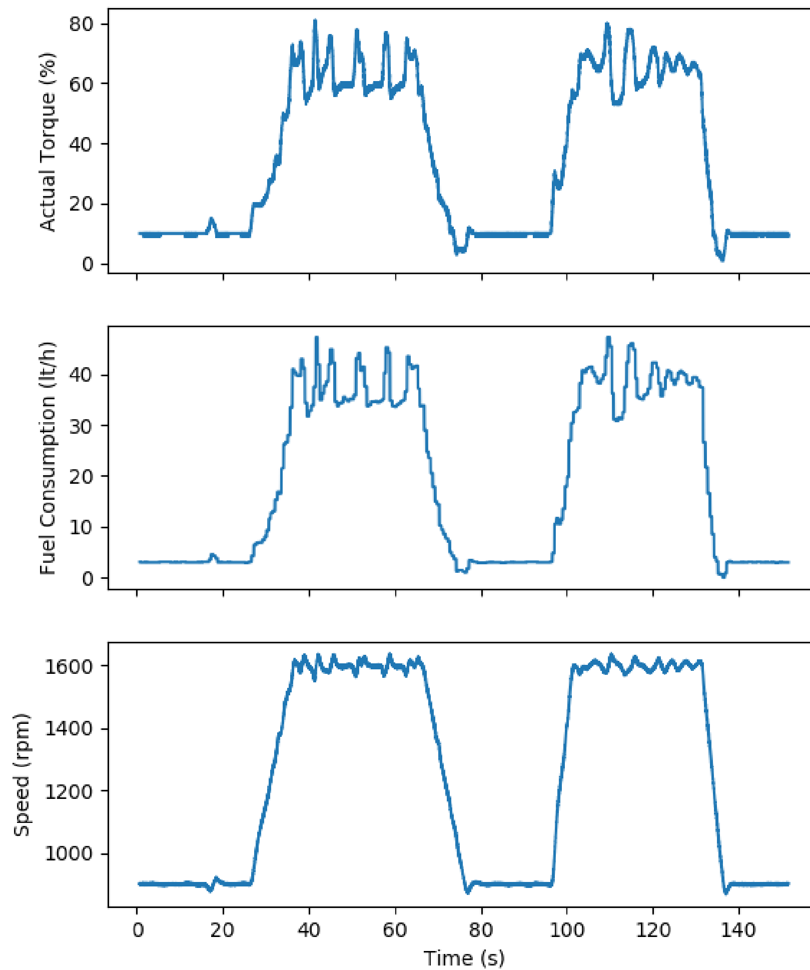


Figure 6.32: Testing Data - Inputs

The fitting results are partially satisfactory as both the following figures and the individual errors imply.

- MSE = 45.1883 $(kPa)^2$
- MAE = 4.3367 kPa

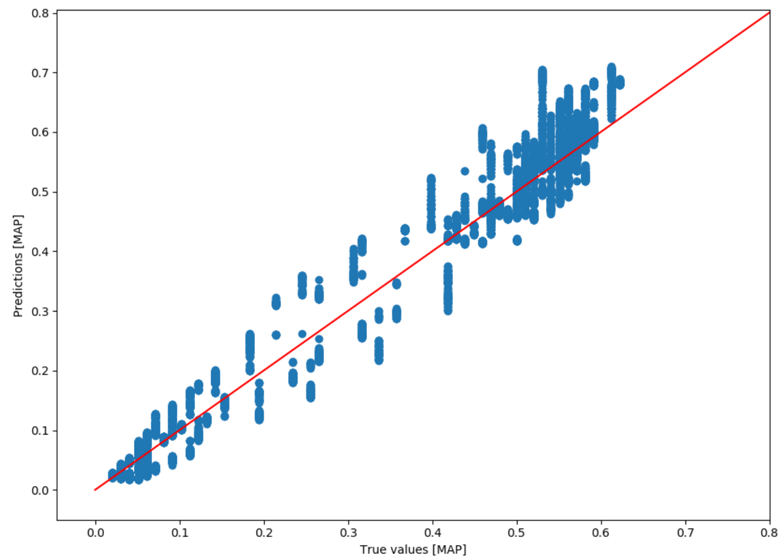


Figure 6.33: Model Testing in Unknown Data: Predicted vs Original Data

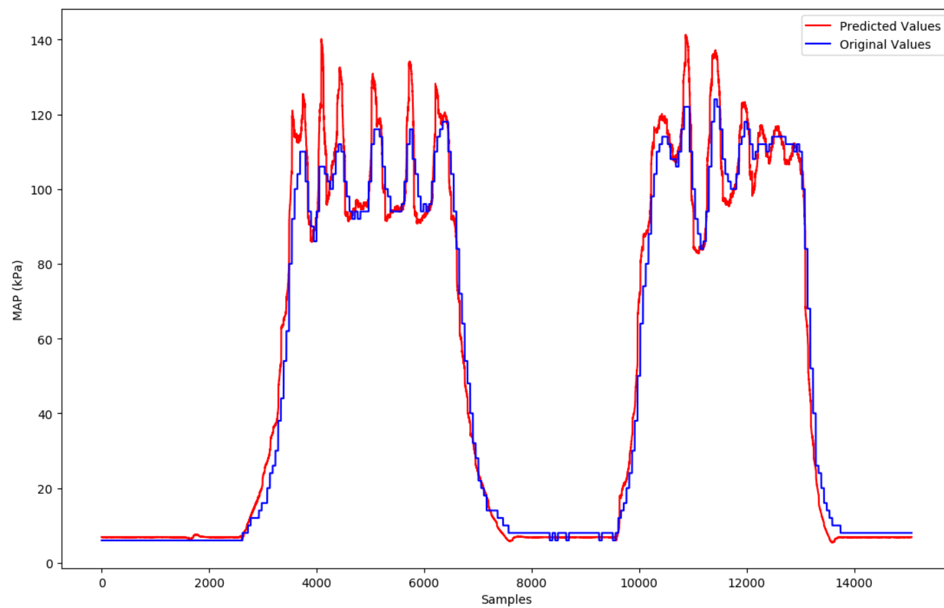


Figure 6.34: FFNN - Trajectories of measured and predicted MAP values, for the unknown data.

Looking at the Figure 6.34, the predicted values are too close to the original at low loads (horizontal parts of the curve) and in the transient regions where the pressure starts to increase/decrease. However, at higher loads, where the original curve gets noisy, the predictions overestimate the real values. This happens especially between samples 4,000 and 6,000 and it is the main reason for the increased values of MSE (45.1883 $(kPa)^2$). The

high values, even a single one, lead to higher means, so the MSE is influenced by large deviations or outliers.

6.3.2 TDNN Model

6.3.2.1 Training Results

By the same token, for the prediction of MAP with a TDNN model, the input values are extracted from the same data set but this time each input has a delay of 0.1, 0.2 and 0.3 seconds. This model is more complicated than the FFNN model, because of the 4 times bigger input *DataFrame*, so a simpler layout with less neurons per hidden layer was chosen. Accordingly, 3 fully connected (Dense) hidden layers have been also used. There are 2 neurons in the first two layers, with a sigmoid activation function and 1 neuron in the last layer, with no activation, so the returned value will be the same with the output. It should be underlined that the different activation functions or optimizers used in FFNN and TDNN are chosen after several attempts, in order to achieve the optimum fitting. The main concept is to experiment with hyperparameters of diverse kind types, so there is no demand to use the same hyperparameters. The total trainable parameters are 101 and the model structure is automatically plotted by a function in Keras, with the following format.

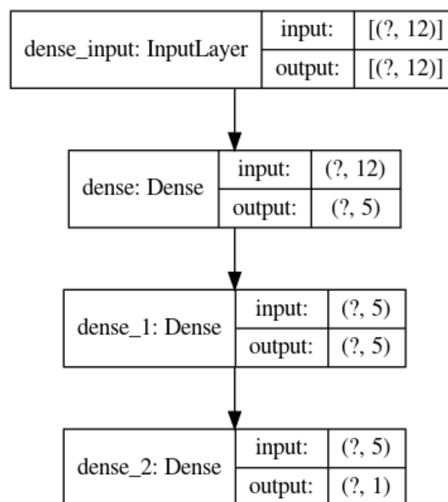


Figure 6.35: Model Layout automated by Keras

Table 6.6: MAP TDNN Hyperparameters & Characteristics

Inputs (×4)	Actual Torque (%) Speed (rpm) Fuel Consumption (lt/h)
Inputs Delays	0.1, 0.2 & 0.3 sec
Hidden Layers	3
Nodes Per Hidden Layer	5-5-1
Total Trainable Parameters	101 (65-30-6)
Activation Function	Sigmoid-Sigmoid-Linear
Optimizer	AdaMax (Learning Rate = 0.0005)

The training process for the TDNN model of MAP was iterated for almost 150 epochs, that is threefold times the iterations of FFNN. Also, the curves of both MAE and MSE are smoother and more convergent than those of FFNN Model, hence a better fitting result is expected. As a matter of fact, the pair plot of True values and Predictions in Figure 6.38 could confirm the latter assumption, but the testing errors in Table 6.7 do not imply the same. However, the functions for error calculations are not always trustworthy, because they may be easily influenced by an outlier or an exception. So, the comparison between the FFNN and TDNN predictions will be more clear when testing in unknown data.

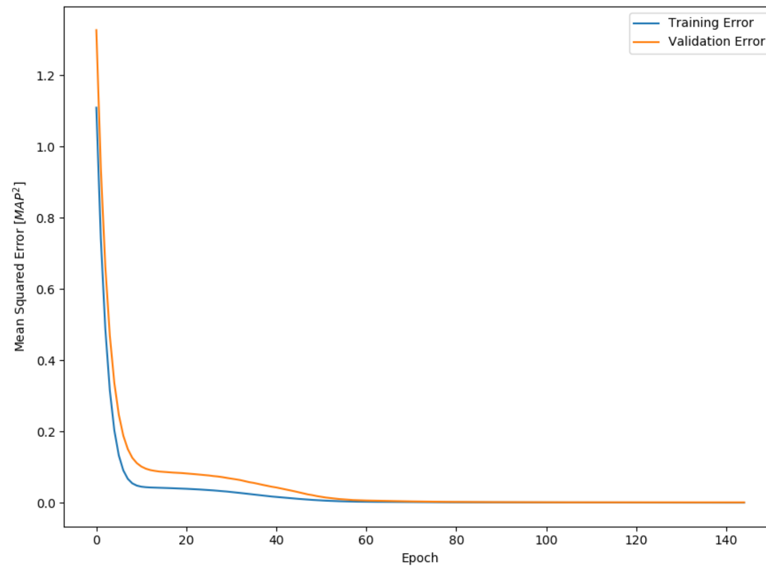


Figure 6.36: Mean Squared Error during training.

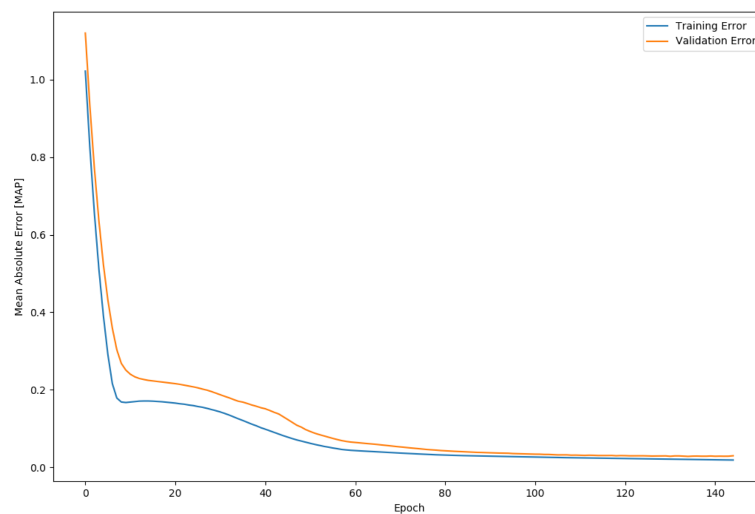


Figure 6.37: Mean Absolute Error during training.

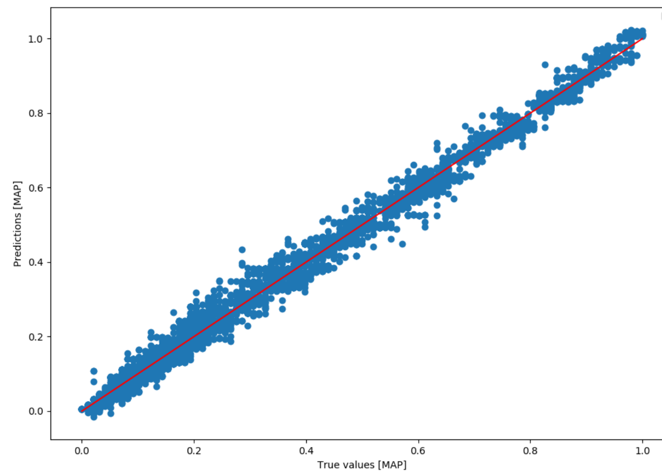


Figure 6.38: Model Testing: Predicted vs Original Data

Table 6.7: Comparison between FFNN & TDNN predictions on testing data from the same data set of training inputs.

	FFNN	TDNN
MSE (kPa^2)	17.6313	25.1919
MAE (kPa)	3.0754	3.9663
MAPE (%)	7.3557	10.8069

Finally, there is nothing really special to mention as regards to Figure 6.39, because the signal measurements have been fitted significantly well. Some vertical deviations at low and high loads are expected and do not have a bad effect on the overall performance of the network.

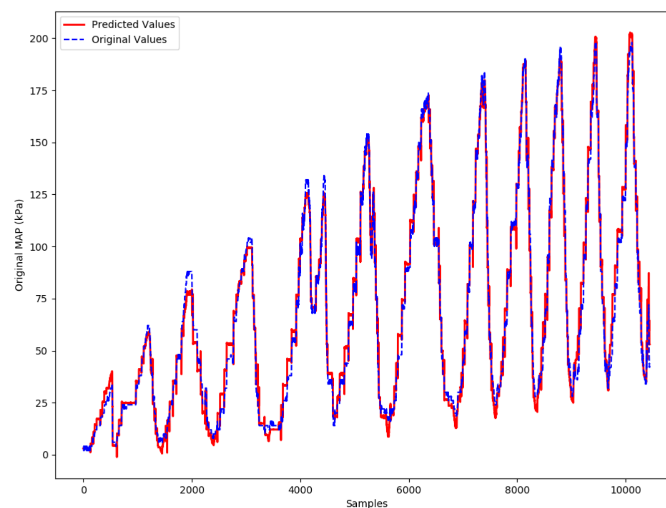


Figure 6.39: Trajectories of measured and predicted MAP values.

6.3.2.2 Testing Results & Comparison with FFNN Performance

After that, the TDNN model is tested on the same unknown data set as FFNN, 6.32, for extra validation and evaluation. The pair plot points in Figure 6.40 are linearly distributed around the $y = x$ line with an acceptable deviation, e.g. ± 0.1. At first glance, the fitting results of Figure 6.41 look very similar to those of the FFNN model, but when looking at details, of course, there are many differences. In this point, it should be mentioned that the "grid" feature in Figure 6.40 was selected in order to see the $\pm$ deviation more clearly. In the rest Figures of this thesis the grid has not included because it would be harder for someone to decode them, especially in the regions where many curves and lines interpolate.

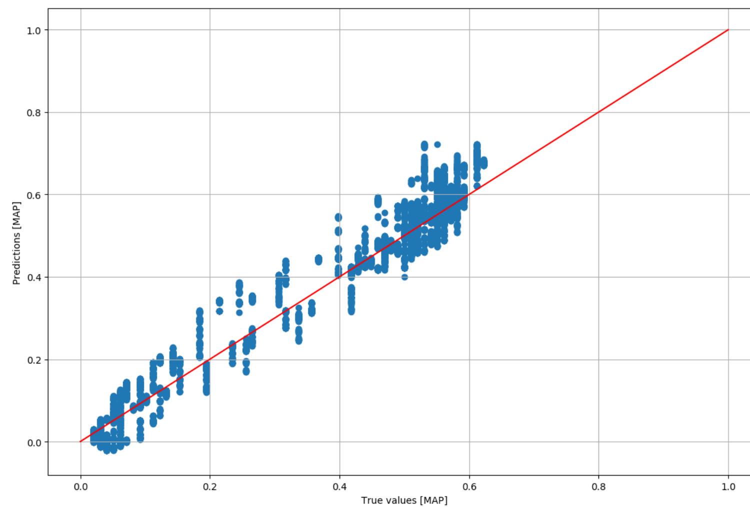


Figure 6.40: Model Testing in Unknown Data: Predicted vs Original Data

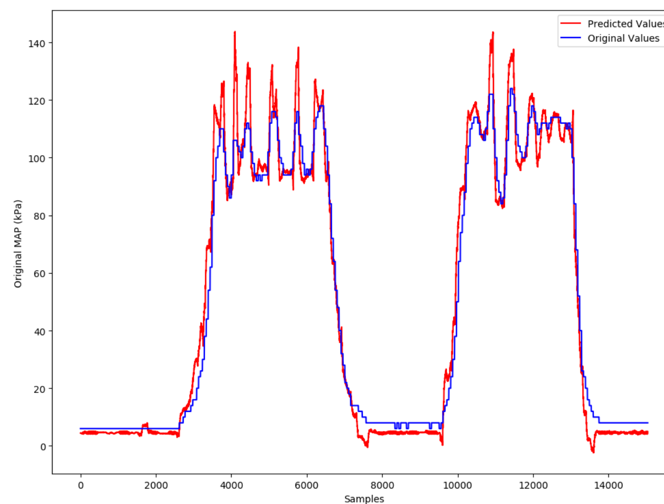


Figure 6.41: TDNN - Trajectories of measured and predicted MAP values, for the unknown data.

Before presenting the concluding graph of the MAP modeling, it is worth mentioning that the TDNN model predictions have larger error values than FFNN, tested with the same input data set, as it can be also seen in Table 6.8.

Table 6.8: Comparison between FFNN & TDNN predictions on unknown testing data

	FFNN	TDNN
MSE $(kPa)^2$	45.1883	52.1849
MAE (kPa)	4.3367	5.0948

In the noisy parts of the graph in Figure 6.42, there is no clear view of which model predictions are more accurate. Both FFNN and TDNN follow the trend and dynamics of the real MAP curve, with larger vertical deviations at high loads-spikes of the graph. Specifically, the highest measurements of MAP are overestimated by both networks. However, the created "valleys" in the noisy regions are obviously better predicted. Having said that, the horizontal and somewhat linear parts of the graph in Figure 6.42 are obviously better predicted by the FFNN model.

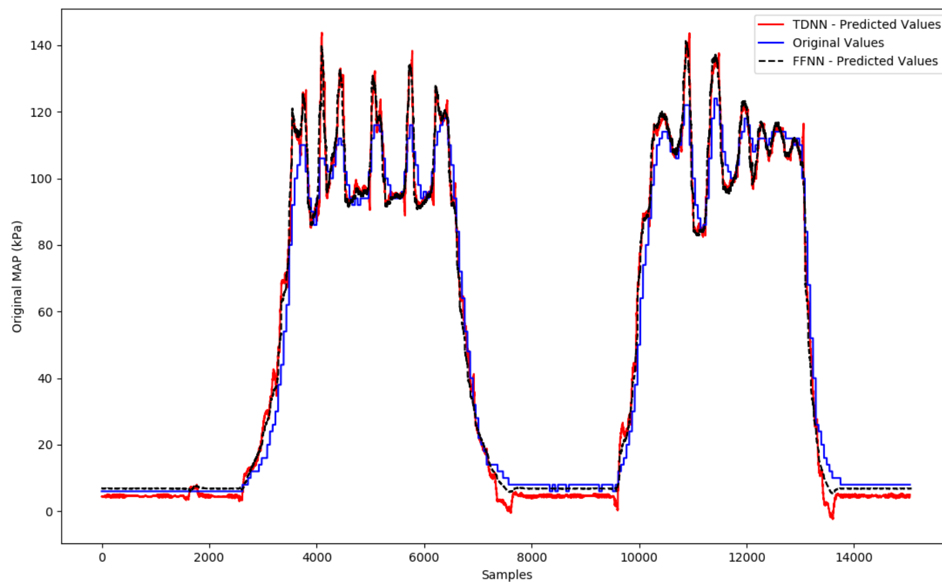


Figure 6.42: Final comparison between experimental MAP (Original Values), FFNN and TDNN trajectories.

6.4 Lambda (λ) Models

6.4.1 FFNN Model

6.4.1.1 Training Results

The unit of λ was not used as an input to the previous neural networks due to its peculiar, and sometimes difficult to handle, distribution. In fact, there is no interest for the extremely high values of λ, because they do not correspond to reality and to real-time values for a diesel engine. Hence, all the input data sets and the output λ were "further" pre-processed in order to keep all the values which confirm the inequality: $\lambda \leq 10$.

In Figure 6.43, the relationship between λ and Fuel Consumption, MAP & Actual Torque, accordingly, is expressed by the reciprocal function, as pair plots form the graph of $y = \frac{1}{x}$. The scatter plot between λ and Speed does not show any function pattern, but it marks the area that the experiment stimulates, as all pair plots which contain the Speed in one axis.

In addition, the Correlation Heatmap in Figure 6.44 verifies what pair plots show about the relationship between the units. The negative correlation coefficient values near -1 mean that all data points lie on a curve where values on y-axis decrease as values on x-axis increase. In simple words, the units are almost inversely proportional.

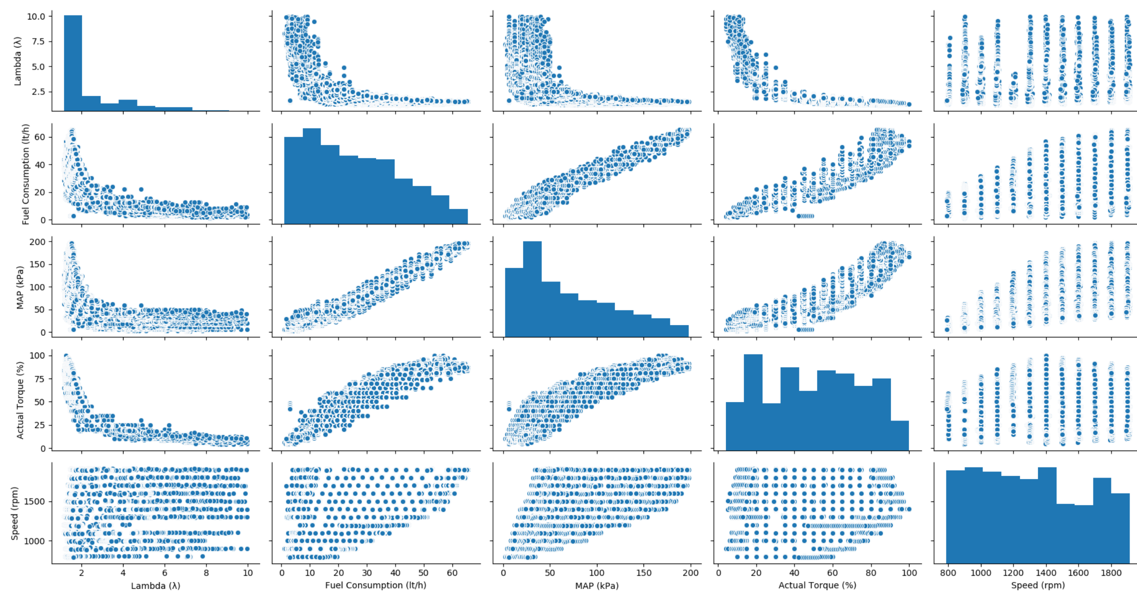


Figure 6.43: Pairwise Distributions

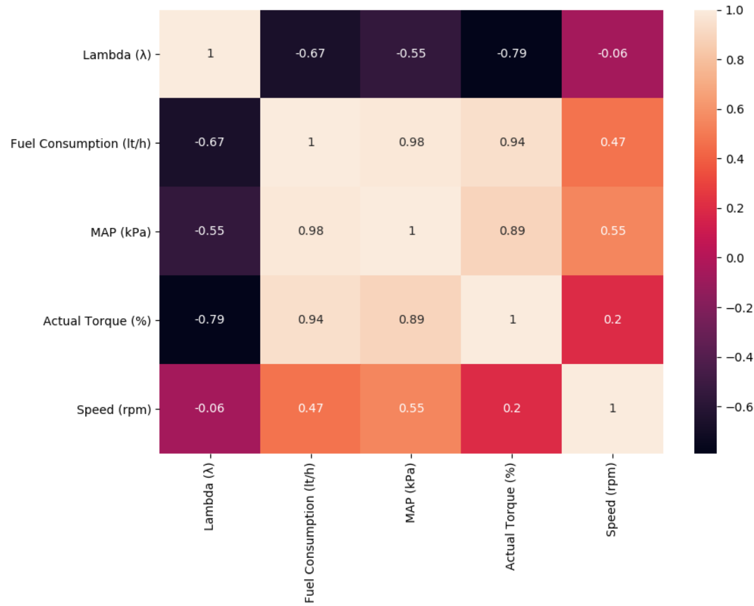


Figure 6.44: Heatmap of Pearson Correlation Coefficient

For the FFNN model of λ prediction 5 fully connected (Dense) hidden layers were used, with 20 neurons in the first three hidden layers and a ReLU activation function, 10 neurons in the fourth hidden layer and a Sigmoid activation function (normalized results between (0,1)) and finally 1 neuron in the last hidden layer with no activation function. This time, more hidden layers with more neurons were used, in comparison with Fuel Consumption and MAP models, because the training process was more demanding and the output value harder to be learnt. In fact, with fewer layers the problem of overfitting came up from the 20th epoch, while with the described structure the model is trained for almost 45 epochs. This number of epochs may sound small and untrustworthy concerning the quality of model training, but it does not mean that the errors will be high or the predicted values will be inadequate. The layout of the model and its basic characteristics and parameters are summarized in the following figures.

Table 6.9: Lambda λ FFNN Hyperparameters & Characteristics

Inputs	Actual Torque (%) Speed (rpm) MAP (kPa) Fuel Consumption (lt/h)
Hidden Layers	5
Nodes Per Hidden Layer	20-20-20-10-1
Total Trainable Parameters	1,161 (100-420-420-210-11)
Activation Function	ReLU-ReLU-ReLU-Sigmoid-Linear
Optimizer	RMSprop (Learning Rate = 0.0005)

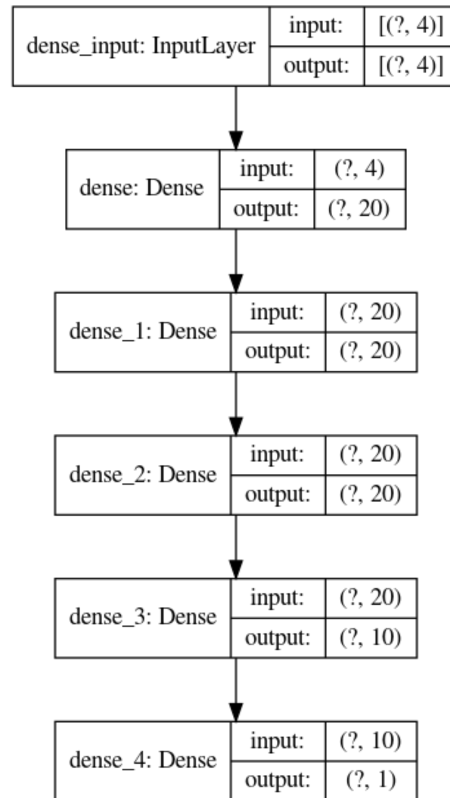


Figure 6.45: Model Layout automated by Keras

As aforementioned, the training process iterated for almost 45 epochs, with RMSprop optimizer and leanring rate equal to 0.0005, producing the below Mean Squared Error (MSE) and Mean Absolute Error (MAE) distributions.

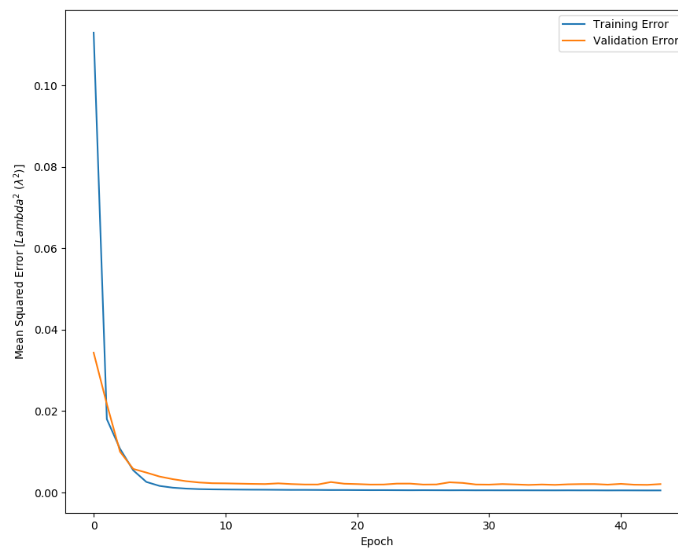


Figure 6.46: Mean Squared Error during training.

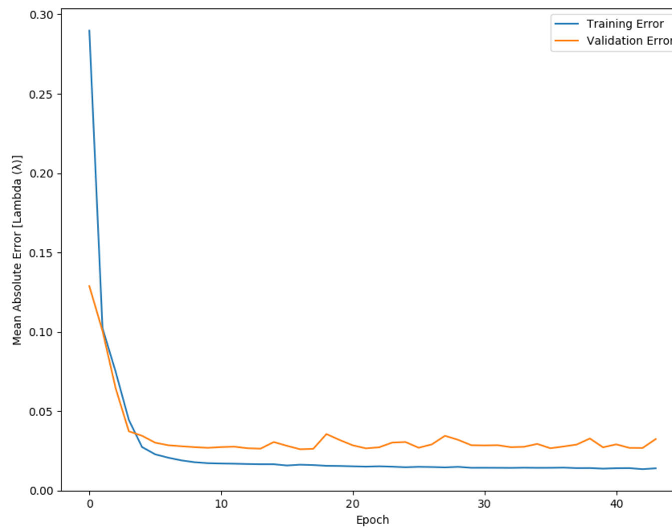


Figure 6.47: Mean Absolute Error during training.

According to Figures 6.46 and 6.47 the training process may not last for too long but that do not cause any augmentation to the the calculated errors.

The problem with overfitting, is that the more specialized the model becomes to training data, the less well it is able to generalize to new data, resulting in an increase in generalization error. This increase in generalization error can be measured by the performance of the model on the validation data set. [29] As in the graphs of previous models, the Training and Validtaion MSE are smoother and get closer to each other than in the case of MAE curves. The Validation MAE seems to be noisy with more "ups-and-downs", which finally lead to the overfitting. It is worthwhile noting that, in the beginning of both graphs the validation error is less than the training one. Some possible reasons for this could be:

- The training loss is measured during each epoch while validation loss is measured after each epoch.
- The validation set may be easier than the training set (or there may be leaks).

However, this "problem" is fixed after the 4th epoch and the curves take a typical trend of a good fit during training. The fitting result is shown below in Figure 6.48, with more outlier points, this time, but not so many in comparison to the whole data points. However, the region between 0 and 0.2, on both axis, is quite dense around the $y = x$ line. If we denormalize the values of the λ according to the equation (5.1), filled-in with the minimum (MIN(λ)=1.15) and maximum (MAX(λ)=9.98) values, the values range between 0 and 2.916 in the square area [(0,0.2),(0,0.2)]. Consequently, it is obvious that in the region of small, thus more representative, values of λ, the model fits adequately. The aforementioned region, highlighted with orange color in Figure 6.48, is the Area of Interest in this study because it represents the engine operation at high loads, i.e. with less lean air-fuel mixtures. For this reason, the prediction of extremely lean mixtures is out of our interest and there is no expectation to fit perfectly.

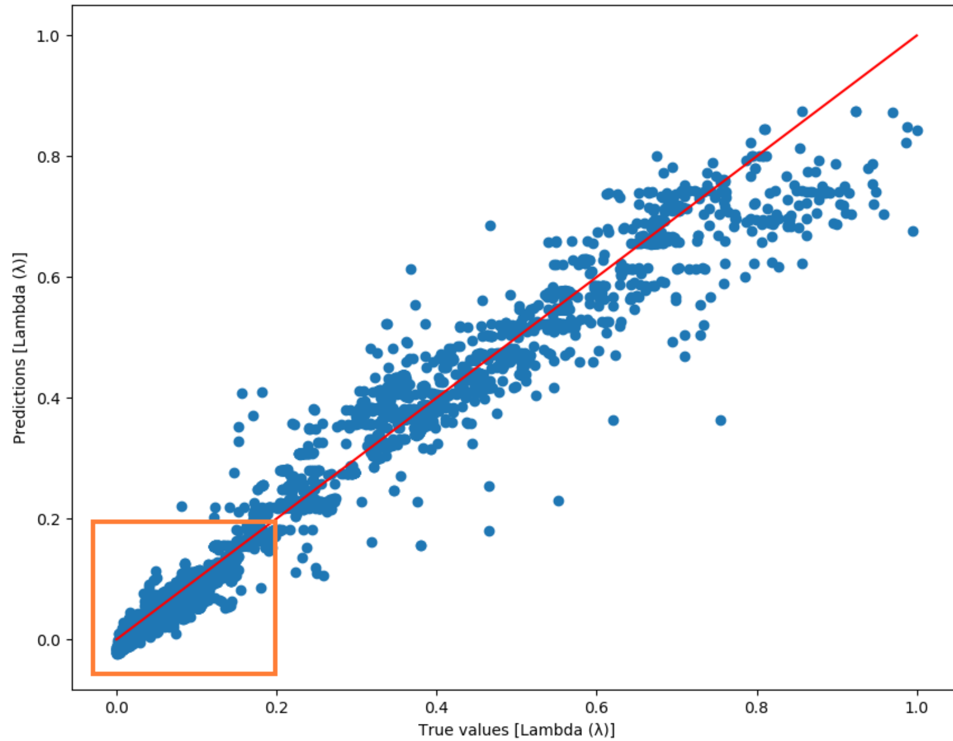


Figure 6.48: Model Testing in Unknown Data: Predicted vs Original Data. Highlighted Area of Interest, i.e. region of higher loads.

The following figures depict the prediction of a testing data set which is extracted from the same available data samples as training data set. In Figure 6.50a there is a small vertical difference between the two curves in low λ measurements, with the predicted values to underestimate the original. The linear parts with the large slopes are ideally fitted and even in Figure 6.50b the really high values are unexpectedly close to the real ones. Accordingly, the error between the predicted and the real values takes the following values:

Testing Errors:

- MSE = 0.0721
- MAE = 0.1781
- MAPE = 8.0377 %

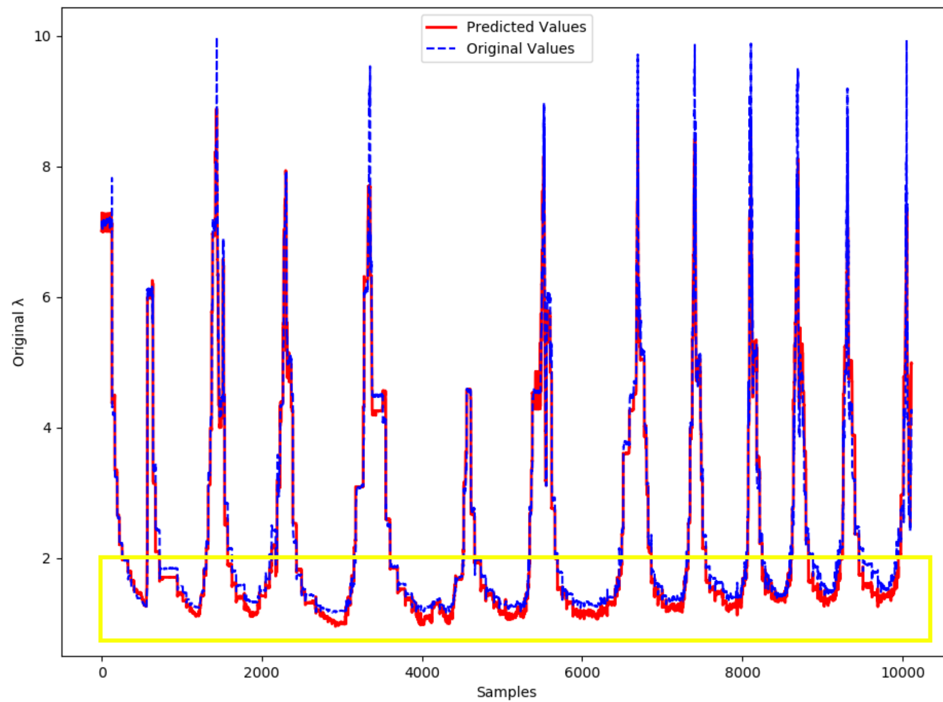
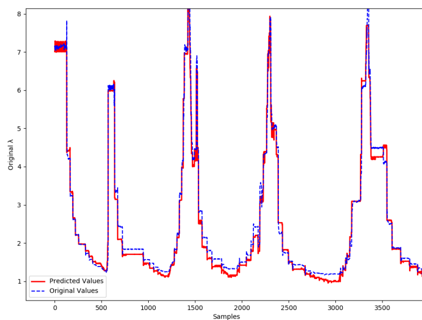
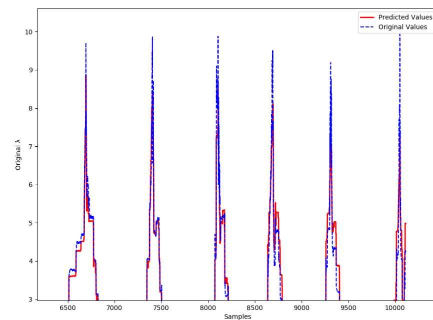


Figure 6.49: Results of λ predictions. Highlighted Area of Interest, i.e. region of lower λ values $\Leftrightarrow$ higher engine loads.



(a)



(b)

Figure 6.50: Various zoomed regions showing the "deviation" between the predicted and the original values.

6.4.1.2 Testing Results

After that, the FFNN model of λ is tested on completely unknown data, which are the same with these of Figure 6.10, but this time without the "Motor Torque (Nm)" and "Exhaust Gas (g/s)", and with the "Fuel Consumption (lt/h)" included in the inputs.

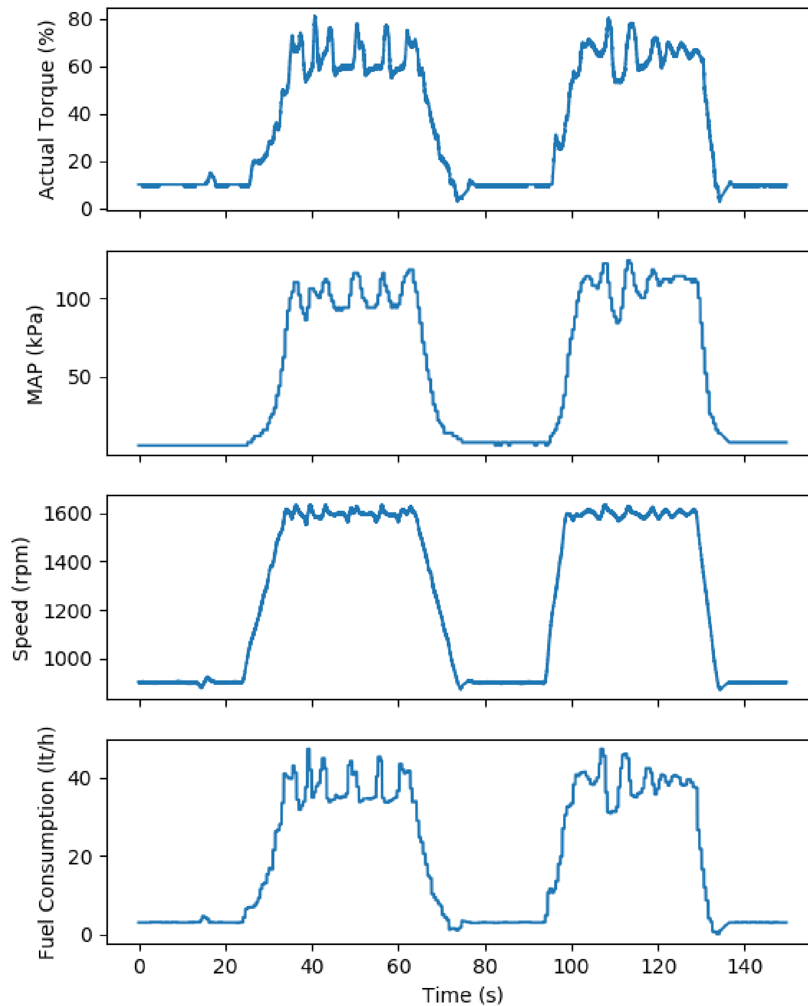


Figure 6.51: Testing Data - Inputs

Again, there may be many outlier points in Figure 6.52, but their total number does not compare to the whole testing data set, which consists of 14,574 data points. Also, as in the previous prediction of λ, the majority of points are concentrated in the "squared" region between 0 and 0.2, symmetrically on both axis, as it can be seen in Figure 6.53.

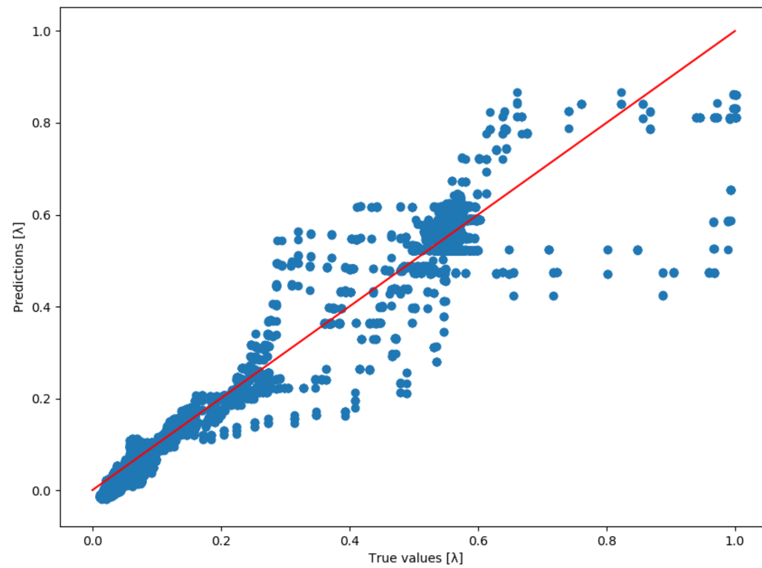


Figure 6.52: Model Testing in Unknown Data: Predicted vs Original Data.

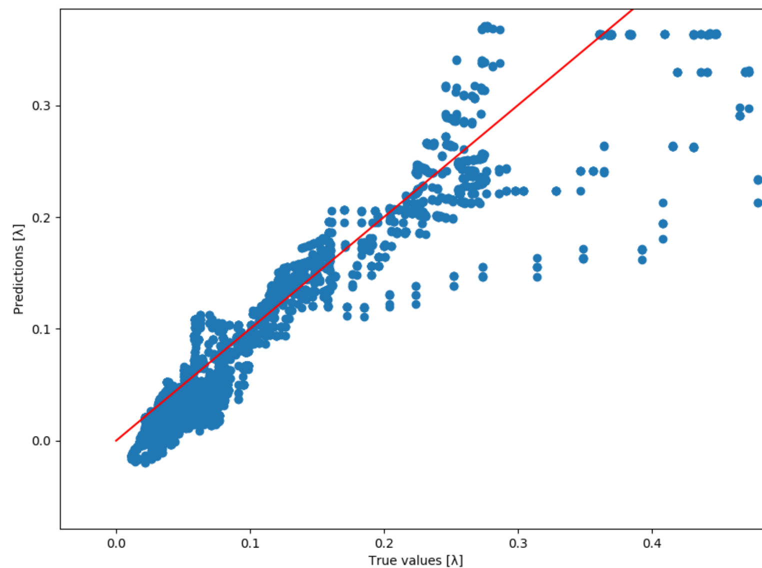


Figure 6.53: Zoomed Area of Interest in ”square” [(0,0.2),(0,0.2)], where many points are densely distributed.

As already mentioned, the region of λ higher values (e.g. $\lambda > 2$) is out of this thesis concern. However, it should be noted that the prediction results of λ near 6-6.5 follow the noisy trend of the experimental values. In more detail, in the range of (0, 2,000) samples the real values are overestimated, whereas in the range of (7,000, 9,000) and after 13,000 samples, approximately, they seem to have the mean of predicted values distribution. Once again the somewhat linear parts of the curve are predicted almost perfectly, even if they

reach really high values (e.g. $\lambda > 8$). Finally, in the range of lower values (Area of Interest), between 0 and 2, there may be a deviation (< 0.5) on the y axis, but the "ups-and-downs" of the original curve are also followed successfully by the predictions.

Testing Error:

- MSE = 0.1561
- MAE = 0.2463

Again, due to some low values of λ between 0 and 1 the result of MAPE is not trustworthy and therefore not included.

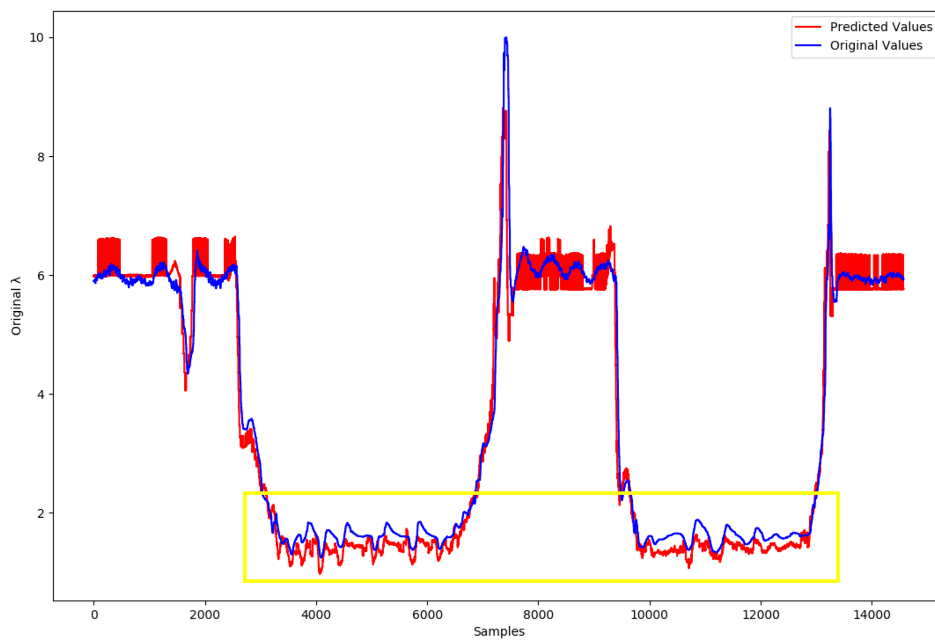


Figure 6.54: FFNN - Trajectories of measured and predicted λ values, for the unknown data.

6.4.2 TDNN Model

6.4.2.1 Training Results

Similarly to the TDNN models for the Fuel Consumption or MAP prediction, the input units remain exactly the same but this time they are also delayed for 0.1, 0.2 and 0.3 seconds. Thus, each input unit is used 4 times, one for the present and three for the past. In that way, the total input data sets for the prediction of λ are $4 \times 4 = 16$. For this reason, a simpler model layout was designed, in order to avoid early overfitting and a time-consuming training process. Specifically, 4 fully connected (Dense) hidden layers have been used, with 10 neurons in the first two and the ReLU activation function, 2 neurons in the third hidden layer with a Sigmoid activation function and 1 neuron in the last hidden layer and no activation function at all. It could be noticed that the "inverse pyramid"

way of selecting neurons per hidden layer is applied in this model, too. The model layout automated plot and the table with all the basic hyperparameters and characteristics are included below.

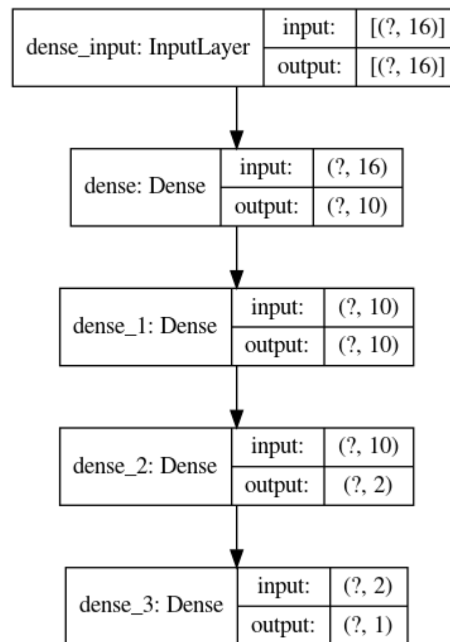


Figure 6.55: Model Layout automated by Keras.

Table 6.10: Lambda λ TDNN Hyperparameters & Characteristics

Inputs (×4)	Actual Torque (%) Speed (rpm) MAP (kPa) Fuel Consumption (lt/h)
Inputs Delays	0.1, 0.2 & 0.3 sec
Hidden Layers	4
Nodes Per Hidden Layer	10-10-2-1
Total Trainable Parameters	305 (170-110-22-3)
Activation Function	ReLU-ReLU-Sigmoid-Linear
Optimizer	RMSprop (Learning Rate = 0.0005)

The training process for the TDNN model of λ was iterated for more epochs than the equivalent FFNN, in specific for 73 epochs. In general, the same described and discussed things for the FFNN error curves apply to these graphs too. The MSE curve has a very good fit once again, whilst the MAE seems more noisy this time and with a biggest gap between the training and validation error in the "steady error" region. This large gap could mean an unrepresentative train data set, which does not provide sufficient information to learn the output. However, this kind of problem does not seem to occur here, taking into account the following fitting results in Figures 6.58 and 6.59.

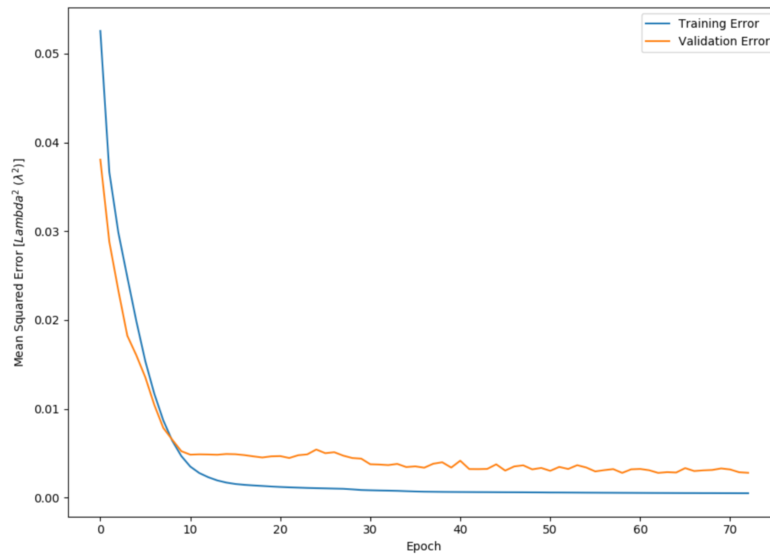


Figure 6.56: Mean Squared Error during training.

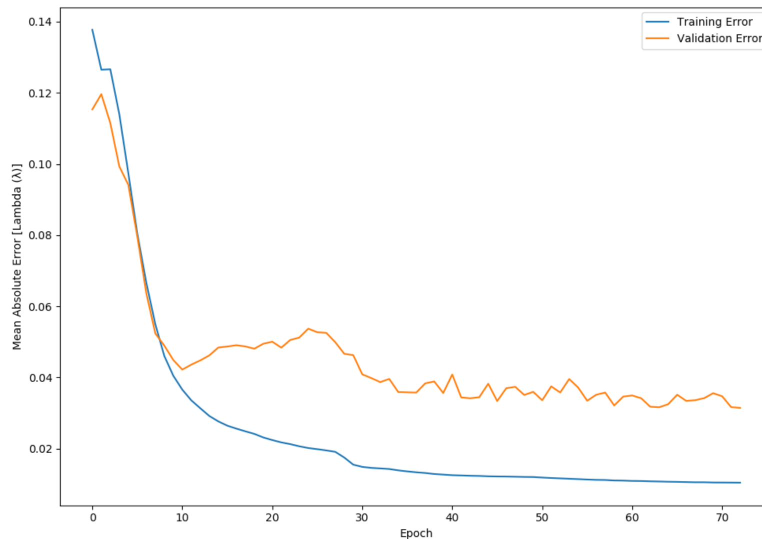


Figure 6.57: Mean Absolute Error during training.

The following figure shows a uniform distribution of predicted and original points around the $y = x$ line, which can be also verified by the impressive convergence between the two different curves even at the smallest, most significant details of Figure 6.59 in the noisy parts with small λ values, or even at the highest values of λ which are out of interest. In this case it is obvious that the TDNN performed much better than the already well fitted FFNN, as the actual error values imply below.

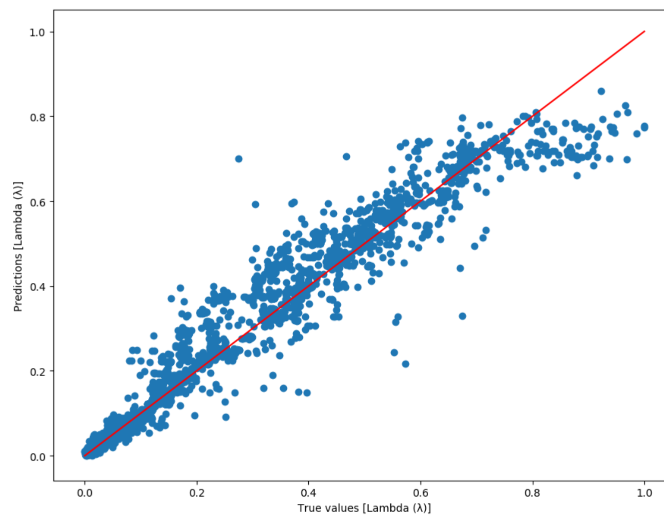


Figure 6.58: Model Testing: Predicted vs Original Data.

Table 6.11: Comparison between FFNN & TDNN predictions on testing data from the same data set of training inputs.

	FFNN	TDNN
MSE	0.0721	0.0805
MAE	0.1781	0.1317
MAPE (%)	8.0377	4.5146

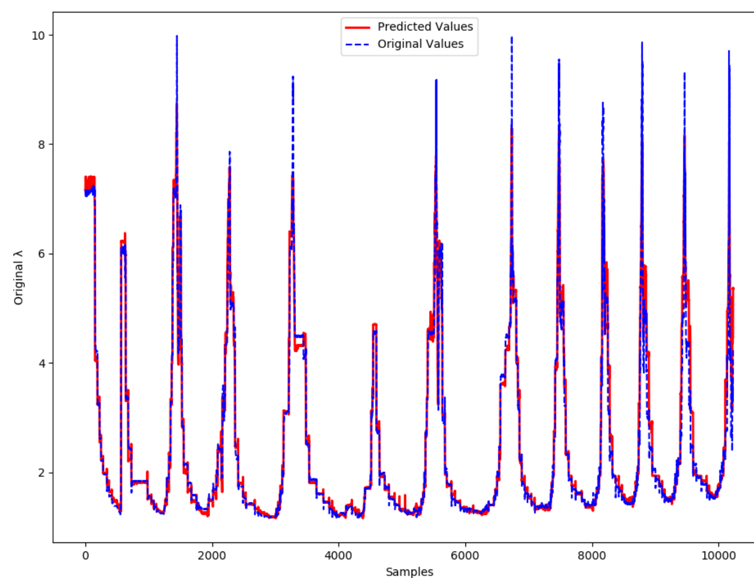


Figure 6.59: Trajectories of measured and predicted λ values.

6.4.2.2 Testing Results & Comparison with FFNN Performance

After the design and training of the TDNN model for λ prediction, the network should be tested on the same unknown data set as FFNN, shown in Figure 6.51. Below, there are the results of this testing process, for the sake of completeness.

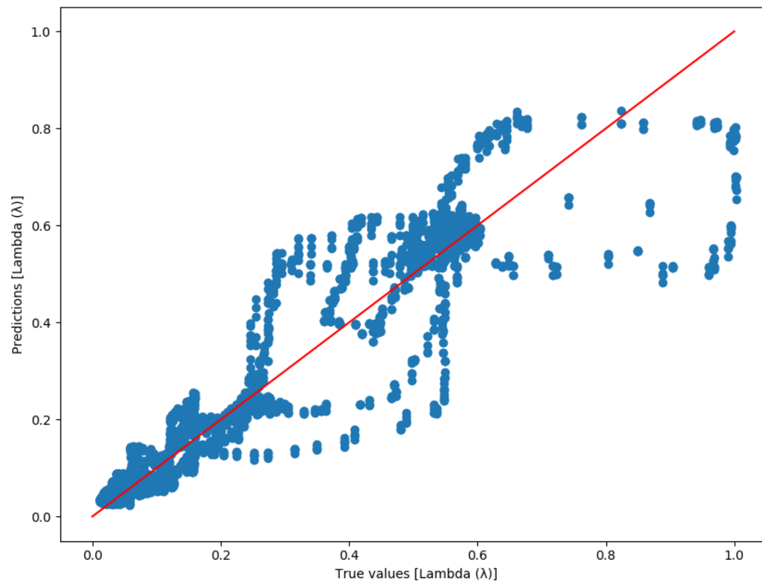


Figure 6.60: Model Testing: Predicted vs Original Data.

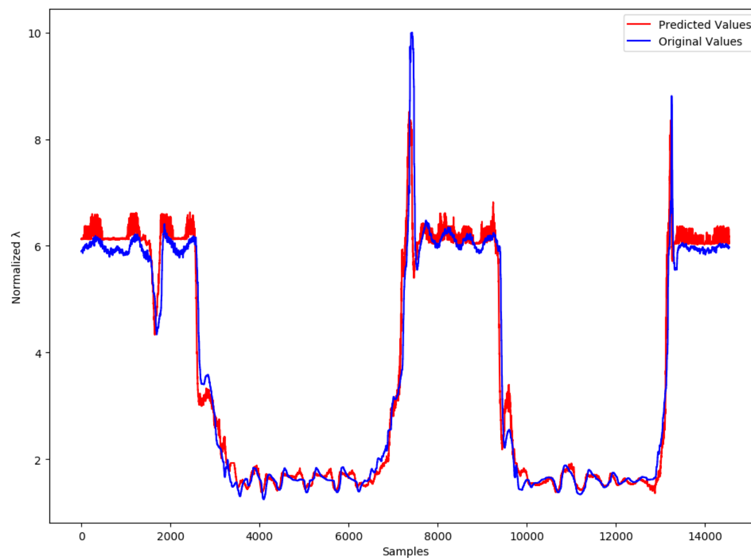


Figure 6.61: TDNN - Trajectories of measured and predicted λ values, for the unknown data.

In this case, the comparison and evaluation of the FFNN & TDNN predictions on the same data set cannot be easily generalized. In fact, the FFNN results seem to get closer to the real values in high ranges of λ, in Figure 6.63a, or in the almost linear regions (i.e. large slopes), but they form a more noisy curve with "rectangular" shapes, which is not representative of the real unit measurements. On the contrary, the TDNN predictions have a more "natural" and realistic pattern, following impressively the original values of λ, especially between samples 10,000 and 13,000 of Figure (6.63b), which is also the *Area of Interest* of this study. Also, it could be noticed that there is an opposition between the errors in Table 6.12, due to the functions handling of data. Finally, when talking about such details and small error values, the selection of the "best" model is up to the researcher's approach, tolerance and point of view. Consequently, in this case, the TDNN model performance could be considered more efficient, because it predicts better the measurements at higher loads (i.e. smaller values of λ).

Table 6.12: Comparison between FFNN & TDNN predictions on unknown testing data

	FFNN	TDNN
MSE	0.1561	0.1805
MAE	0.2463	0.2197

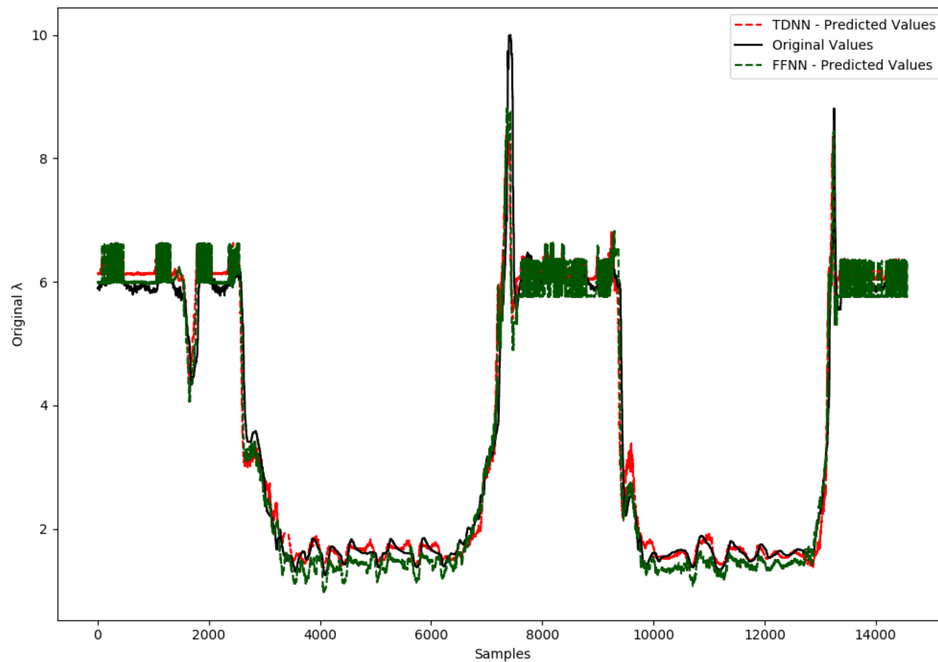
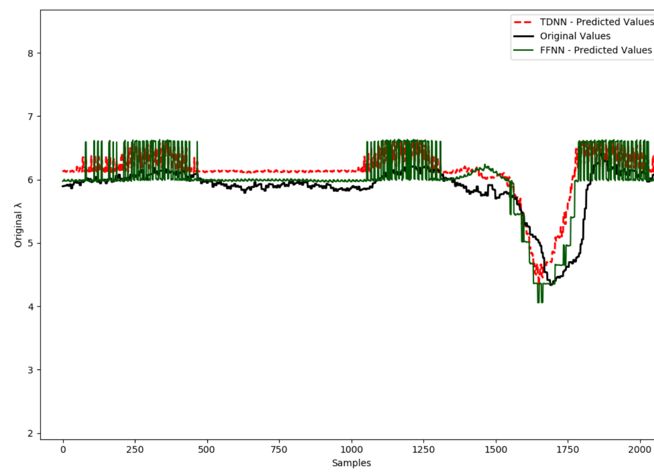
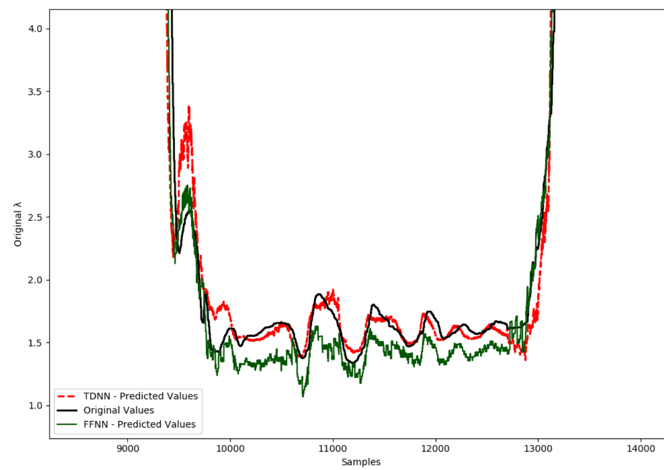


Figure 6.62: Final comparison between experimental λ (Original Values), FFNN and TDNN trajectories.



(a)



(b)

Figure 6.63: Various zoomed regions showing the "deviation" between the FFNN & TDNN predicted and the original values.

6.5 NOx Models

6.5.1 FFNN Model

6.5.1.1 Training Results

In the case of NOx FFNN model training, the things became more difficult than expected. The unit of NOx was more demanding in the training process and the concept of "trial-and-error" was repeated various times until the most decent and trustworthy results were achieved. A possible reason for this could be the "reverse trend" or "mirror" of NOx, in comparison with some of the typical and most common input units (e.g. Actual Torque, MAP, Fuel Consumption, Speed). In fact, as the aforementioned measurements increase, the NOx values decrease, starting with some really high values in the beginning where the EGR valve is closed, and end up to lower but quite noisy values. As already discussed, the NOx emissions are closely-related to λ and EGR Valve Position values. When λ is small or the EGR valve is closed, the NOx emissions reach a high concentration, whereas at really high λ values (e.g. extreme peaks/spikes in Figure 4.9) or a fully open EGR Valve, the NOx is reduced. For these reasons, the NOx distribution appear with many peaks and valleys, which complicate the training process and accordingly the testing. The aforementioned are also verified by the undecoded relationship between NOx and Actual Torque or EGR Valve Position in Figure 6.64 or by the law values of Pearson Correlation Coefficient in Figure 6.65, which imply "weak" correlations or inverse behaviors (e.g. negative value between Speed-NOx).

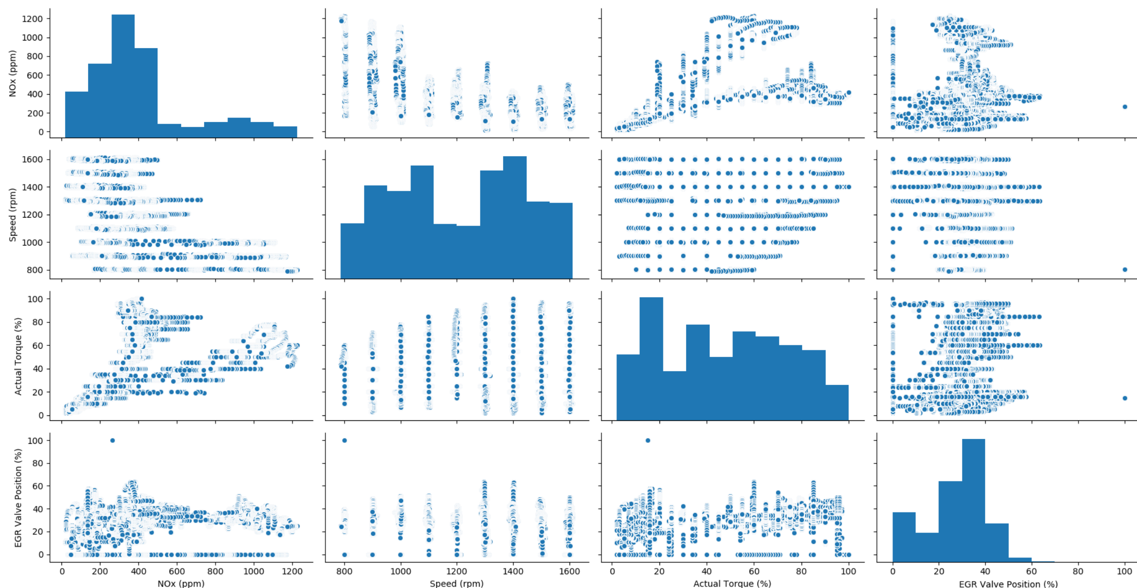


Figure 6.64: Pairwise Distributions

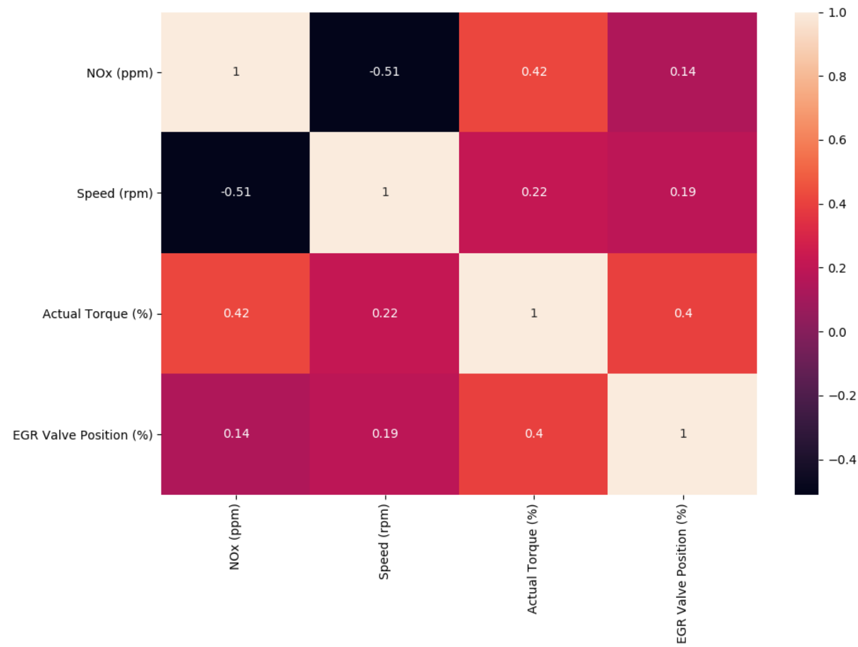


Figure 6.65: Heatmap of Pearson Correlation Coefficient

For the FFNN model of NOx prediction 4 fully connected (Dense) hidden layers were used, with 20 neurons in the first two layers and a ReLU activation function, 10 neurons in the third layer and a Sigmoid activation function (normalized results between (0,1)) and finally 1 neuron in the last hidden layer with no activation function. This network was as the one for the λ prediction, but with 2 hidden layers with 20 neurons and ReLU function, instead of 3. Before settling on this structure many other combinations were tested. The more competitive idea was that of having just one hidden layer but with a large amount of neurons, equal to the 2/3 of the input data sets length. Even if it provided adequate results, the testing errors were higher so this alternative was rejected. Also, an other potential practice was to keep all the data of the input data sets, without keeping the values per 0.1 seconds. This method made the training process too slow and led after 20 epochs to overfitting. So, the final layout of the model and its basic characteristics and hyperparameters are summarized in the following table and figure, accordingly.

Table 6.13: NOx FFNN Hyperparameters & Characteristics

Inputs	Actual Torque (%) Speed (rpm) EGR Valve Position (%)
Hidden Layers	4
Nodes Per Hidden Layer	20-20-10-1
Total Trainable Parameters	721 (80-420-210-11)
Activation Function	ReLU-ReLU-Sigmoid-Linear
Optimizer	AdaMax (Learning Rate = 0.001)

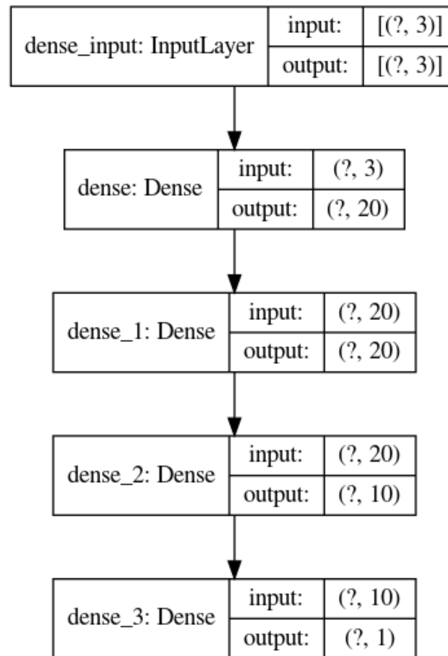


Figure 6.66: Model Layout automated by Keras

This time, the training process lasted for almost 70 epochs before overfitting, with Adamax optimizer and learning rate equal to 0.001. The Figures 6.67 and 6.68 of MSE and MAE metrics functions demonstrate a good fit, as in Fuel Consumption and λ cases, with just a change of validation error direction in the first 10 epochs. Something like this is typical and normal, considering that the network has just started training and the validation set may be harder than the training one to be predicted, as both sets are selected randomly.

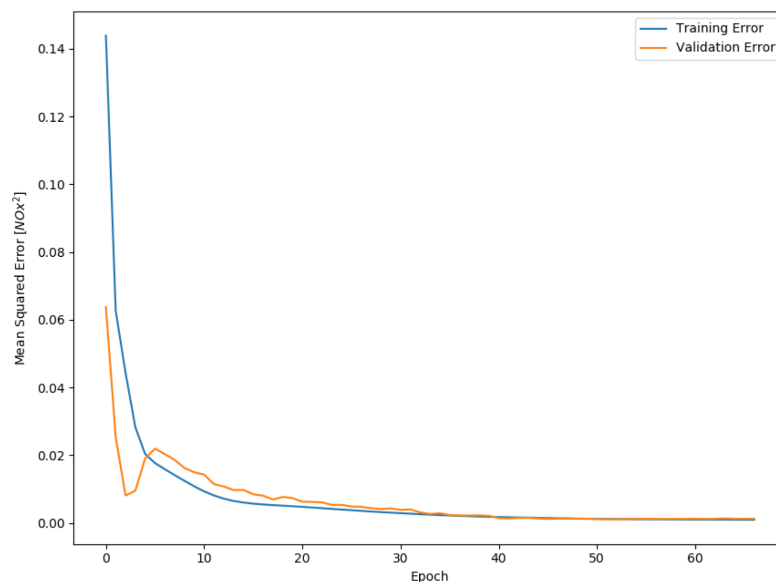


Figure 6.67: Mean Squared Error during training.

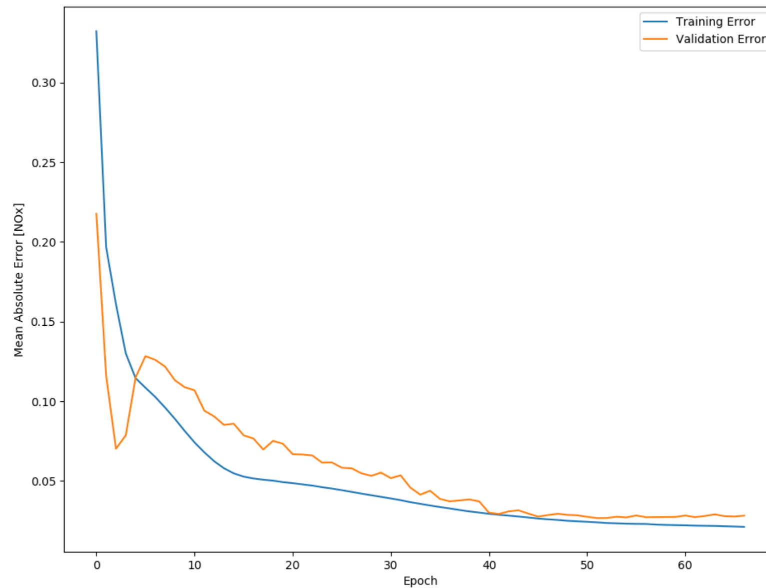


Figure 6.68: Mean Absolute Error during training.

After the training process comes the evaluation in Figure 6.69, where the fitting result seems satisfactory for the most part, but with more uniformly distributed outlier points than in previous model fittings. The interesting thing is that in low (< 0.4) and high (> 0.8) NOx normalized values the model fits quite well, but in the intermediate region, there is a gap around the $y = x$ line. This lack of points becomes more clear and obvious when plotting the original and predicted values of NOx in the same graph, as in Figure 6.70 between samples 4,000 and 6,000. However, this model "failure" is reasonable and unsurprising, since there are approximately 7 rapid up-and-downs of more than 200 ppm in a range of 500 samples (or 50 seconds). Consequently, the model instead of predicting accurately all these peaks and valleys, it reaches a mean between their minimum and maximum values. Unfortunately, this deviations cost, according to the error calculations below, but the model still has predicted the NOx real values quite well, following exactly the trend and direction of the curve.

Testing Errors:

- MSE = 1,624.6502 ppm^2
- MAE = 27.8152 ppm
- MAPE = 10.3316 %

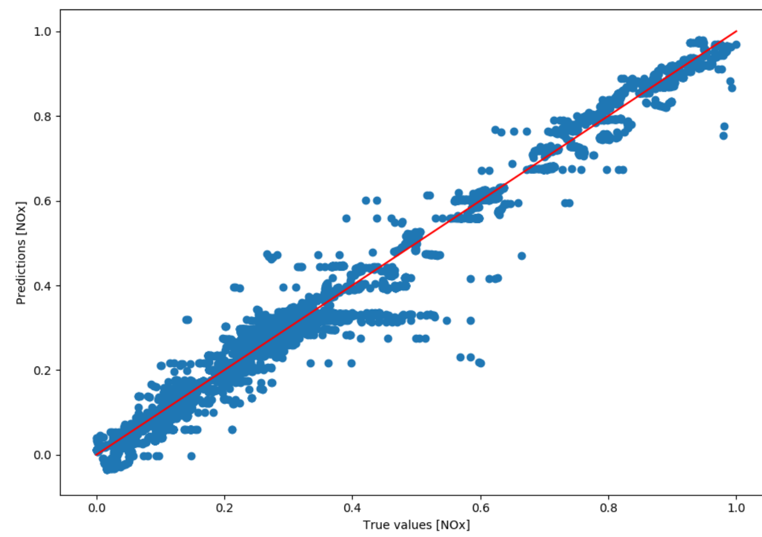


Figure 6.69: Model Testing in Unknown Data: Predicted vs Original Data

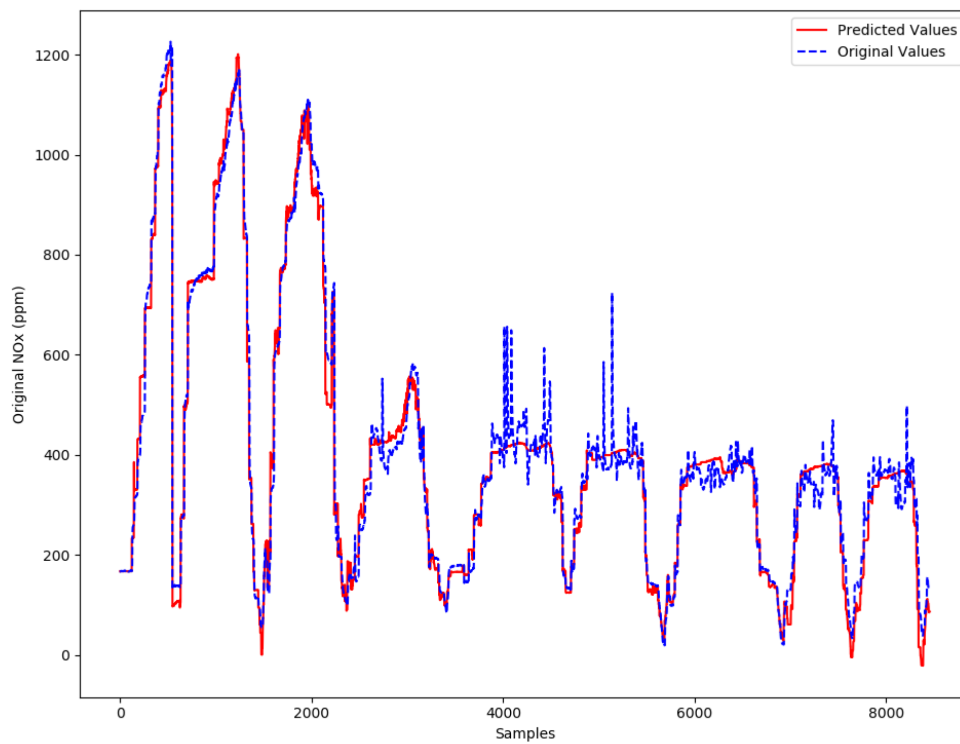


Figure 6.70: Results of NOx predictions.

6.5.1.2 Testing Results

After that, the FFNN model of NOx is tested on completely unknown data and not extracted from the same data set as training points. The extra-testing data was also used as testing inputs in the previous models of Fuel Consumption, MAP and λ.

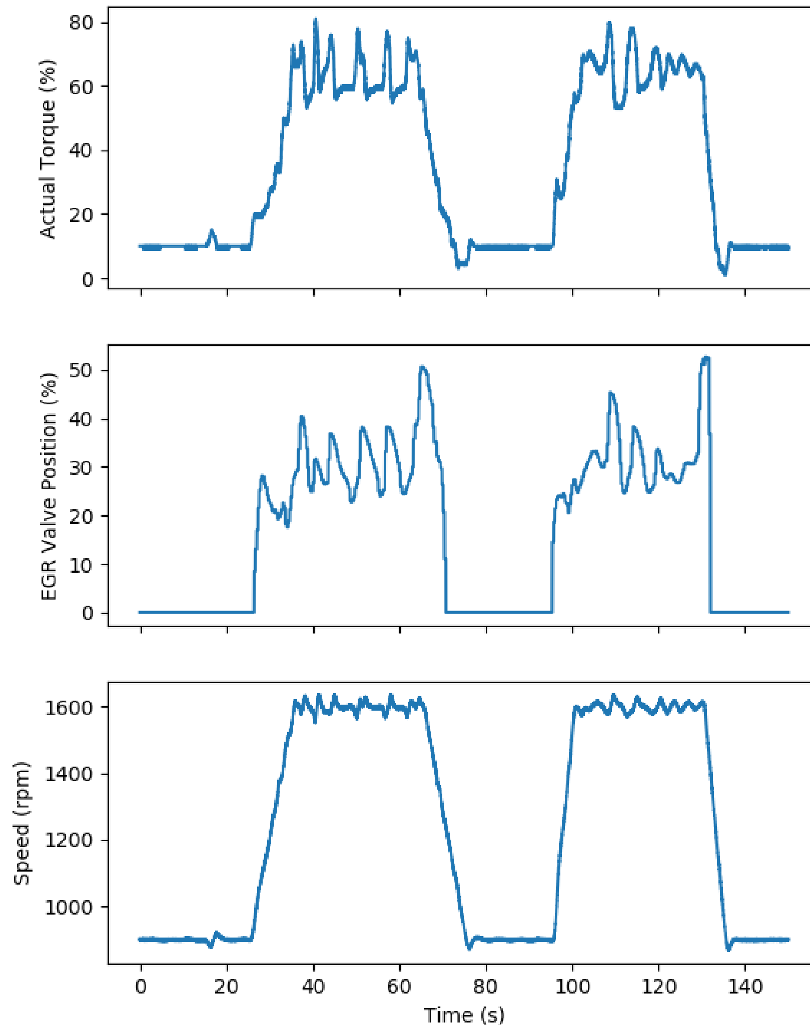


Figure 6.71: Testing Data - Inputs

In this case, the distribution of the real and predicted values around the identity line is unprecedented. The inability of the model to predict the intermediate values of NOx data set, in Section 6.5.1.1, is also confirmed by the following results. Whilst there are some points which are distributed near the $y = x$ line in Figure 6.72, the real normalized values (x-axis) between 0.1 and 0.2 are overestimated by the predictions (y-axis). This overestimation becomes more clear and profound in Figure 6.73, specifically before and after the 2,000 samples, and precisely on sample 10,000, where the model in all three cases

tried to predict a sudden increase of NOx. In general, the predictions follow the dynamics of the real values in steady, transient and noisy regions, as can be seen in Figure 6.73, but there is a delay of 150 samples (i.e. $150 \times 0.1 = 15sec$) between the two signal plots (see Figure 6.74), which causes this misalignment.

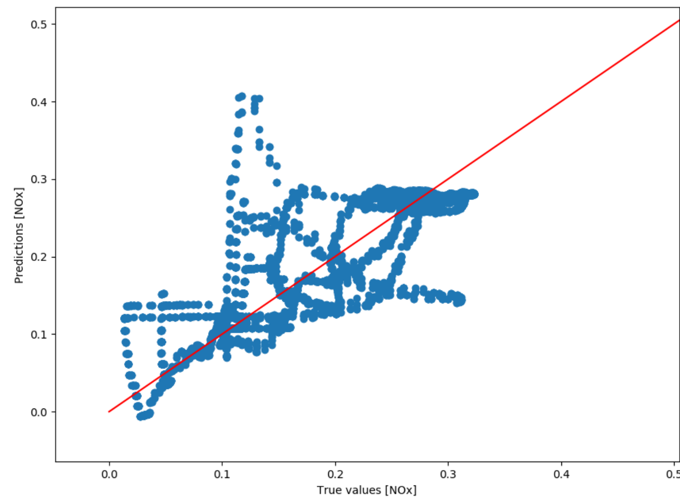


Figure 6.72: Model Testing in Unknown Data: Predicted vs Original Data.

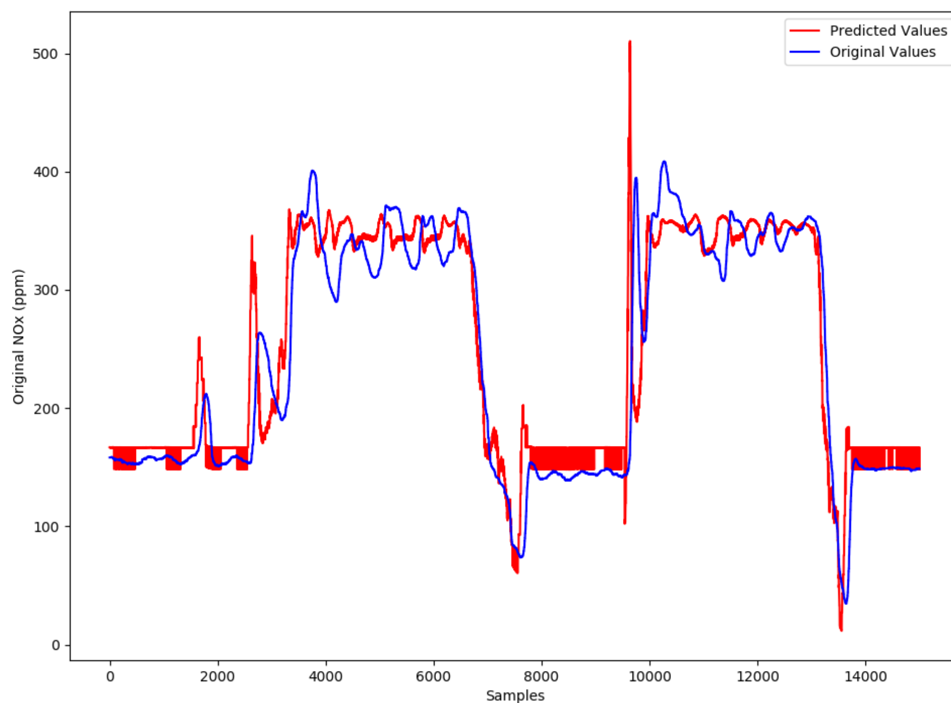


Figure 6.73: FFNN - Trajectories of measured and predicted NOx values, for the unknown data.

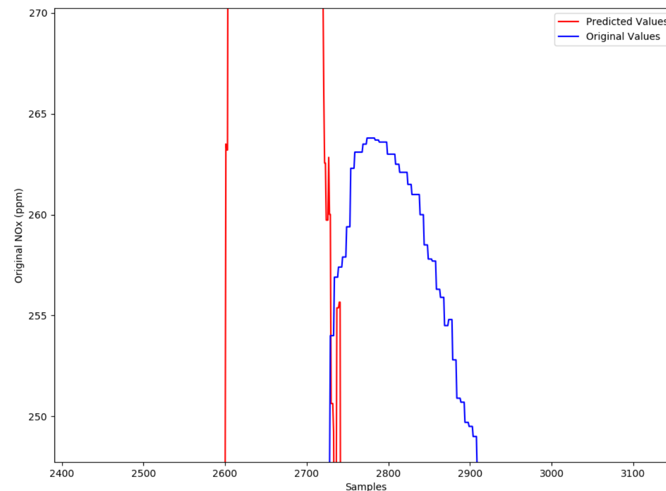


Figure 6.74: Zoomed region of Figure 6.73 showing the delay of 150 samples between the predicted and the original values.

Testing Error:

- MSE = 1,547.0963 ppm^2
- MAE = 0.2463 ppm
- MAPE = 13.3546 %

6.5.2 TDNN Model

6.5.2.1 Training Results

Similarly to the previous TDNN models, the input data sets for the NOx prediction remain exactly the same as with the FFNN model, but delayed for 0.1, 0.2 and 0.3 seconds. In that way, each input unit is used 4 times, one for the present and three for the past, so the total inputs for the TDNN NOx model are $4 \times 3 = 12$. As already discussed, in order to avoid a network with both multiple inputs and complex structure, less neurons are selected per hidden layer, so the number of hyperparameters diminishes. In specific, the table with the selected hyperparameters and the model layout automated by Keras are attached below.

Table 6.14: NOx TDNN Hyperparameters & Characteristics

Inputs (×4)	Actual Torque (%) Speed (rpm) EGR Valve Position (%)
Inputs Delays	0.1, 0.2 & 0.3 sec
Hidden Layers	4
Nodes Per Hidden Layer	5-5-5-1
Total Trainable Parameters	131 (65-30-30-6)
Activation Function	ReLU-ReLU-Sigmoid-Linear
Optimizer	RMSprop (Learning Rate = 0.001)

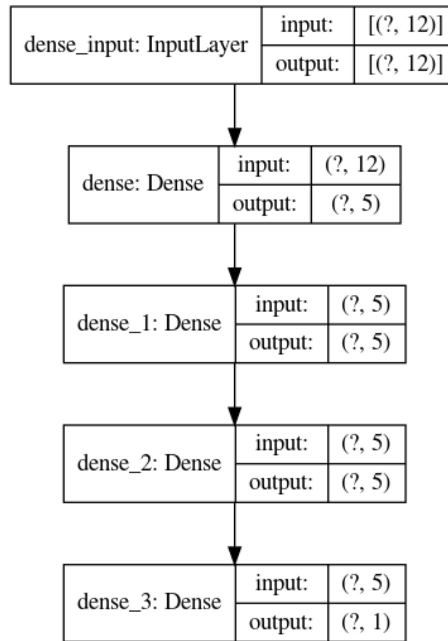


Figure 6.75: Model Layout automated by Keras.

The training process for the TDNN model of NOx lasted for fewer epochs than the relevant FFNN, particularly for almost 35. In MAE graph the validation error is lower than the training one, more possibly due to an ”easier-to-learn” validation data set, or to other potential reasons mentioned in previous sections. As regards to MSE graph, the learning curve for training loss looks like a good fit, whereas the learning curve for validation loss shows noisy movements around the training loss. This may occur if the validation data set has too few examples as compared to the training data set. This kind of validation data set is called ”unrepresentative”, meaning that it does not provide sufficient information to evaluate the ability of the model to generalize.[29] However, these observations do not seem to bother the performance of the network according to the testing errors and scatter plot of experimental vs predicted data.

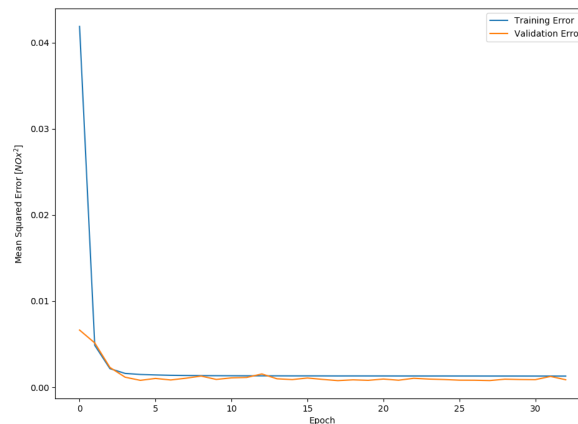


Figure 6.76: Mean Squared Error during training.

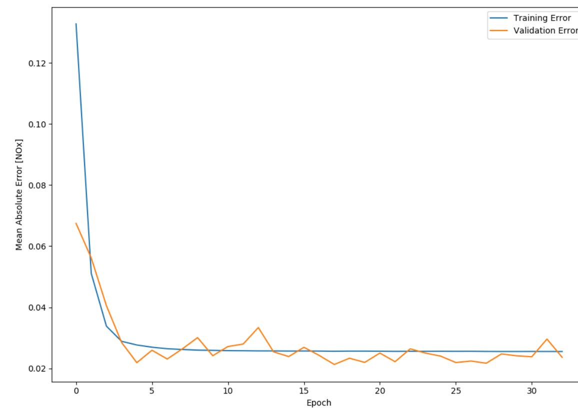


Figure 6.77: Mean Absolute Error during training.

Figure 6.78 shows that the majority of data points are concentrated around the $y = x$ line, especially at the low and high values of NOx, the same as in FFNN model. Again, in the "middle" of the normalized range of original values (e.g. (0.4,0.7)), the predicted values may underestimate or overestimate the real ones, without following a specific pattern. This assumption can also be verified by the Figure 6.80a, where around 30,000 samples the network successfully predicts a peak, but afterwards (see also Figure 6.80b) the demanding "ups-and-downs" of the curve are approached by a mean value, as in the case of the FFNN.

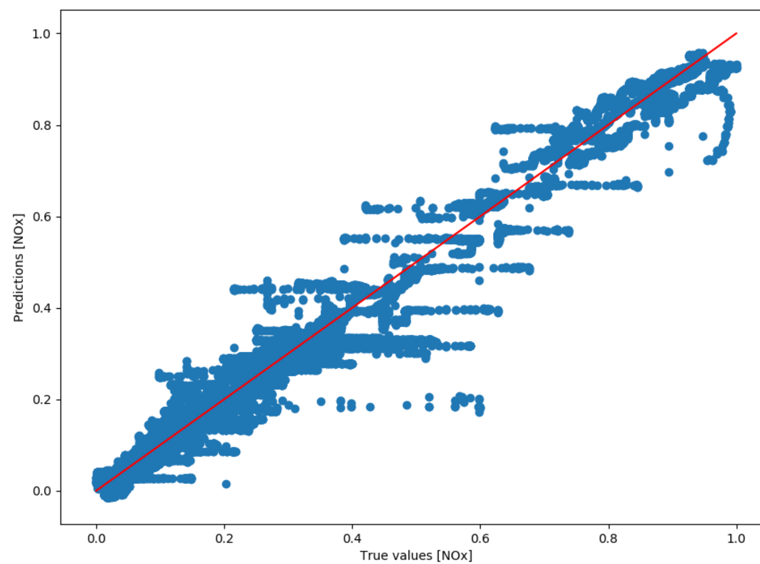


Figure 6.78: Model Testing: Predicted vs Original Data.

Table 6.15: Comparison between FFNN & TDNN predictions on testing data from the same data set of training inputs.

	FFNN	TDNN
MSE (ppm^2)	1,624.6502	1,729.8313
MAE (ppm)	27.8152	29.8030
MAPE (%)	10.3316	9.8782

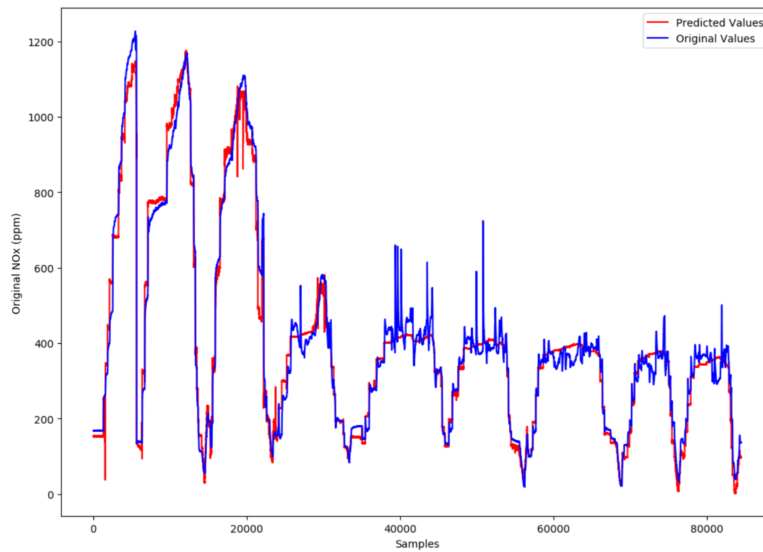
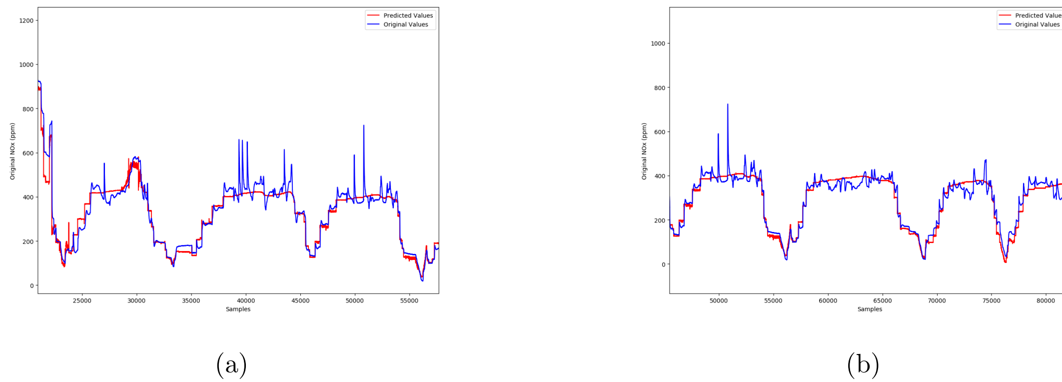


Figure 6.79: Trajectories of measured and predicted NOx values.



(a) (b)

Figure 6.80: Zoomed regions showing the "deviation" between the predicted and the original values.

6.5.2.2 Testing Results & Comparison with FFNN Performance

In the previous section the TDNN model for NOx predictions was tested on a data set which was comparable to the training one. After that, it should also be tested on unknown data sets, that of Figure 6.71, producing the following results.

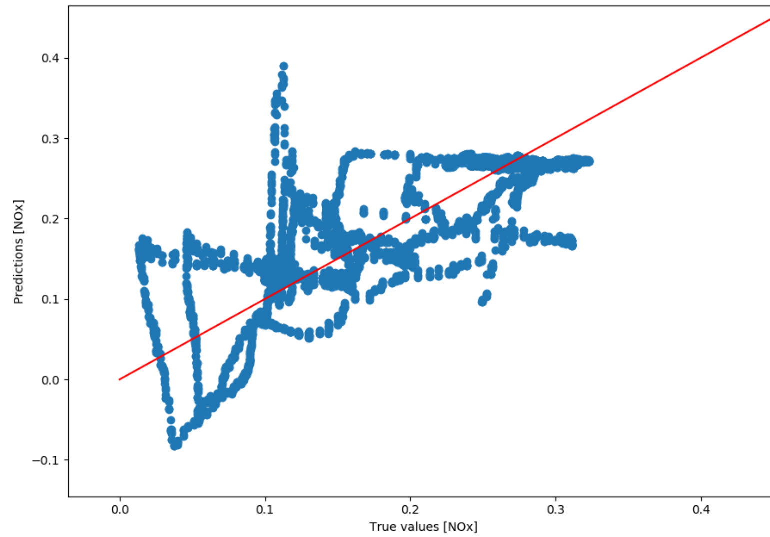


Figure 6.81: Model Testing: Predicted vs Original Data.

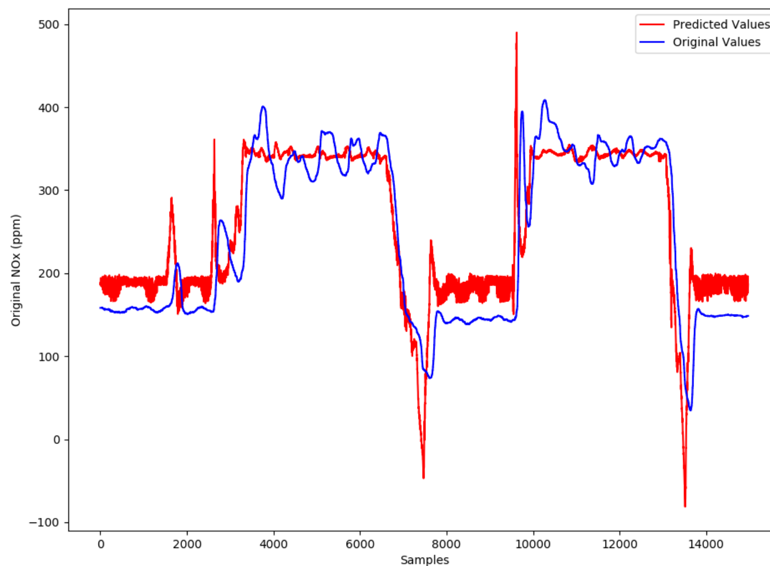


Figure 6.82: TDNN - Trajectories of measured and predicted NOx species, for the unknown data.

In Figure 6.81 the pairs of (True values, Predictions) are not distributed around the identity line but form a random pattern. For instance, near the normalized and true value of 0.1 the predictions tend to a 0.4, that is a 4 times larger value. Except for this extreme case-exception, the majority of original measurements are overestimated, producing 2-3 times bigger or smaller values. This problem becomes more obvious in the Figure 6.82, where only in the noisy regions of ranges: (3,000 , 7,000) & (10,000 , 13,000), or in the almost linear transient parts the results are more representative. In addition, the error values in Table 6.16 and Figure 6.83, which compare the predictions of the FFNN and TDNN models to the experimental results, prove that the FFNN model has performed with larger accuracy, especially in the weakest parts of TDNN. To illustrate, before 8,000 and 14,000 samples, not only does the TDNN fail, but it predicts negative values for an always positive unit (ppm). On the other hand, the FFNN follows exactly the curve of original points.

Table 6.16: Comparison between FFNN & TDNN predictions on unknown testing data

	FFNN	TDNN
MSE (ppm^2)	1,547.0963	2252.2579
MAE (ppm)	24.5868	35.2817

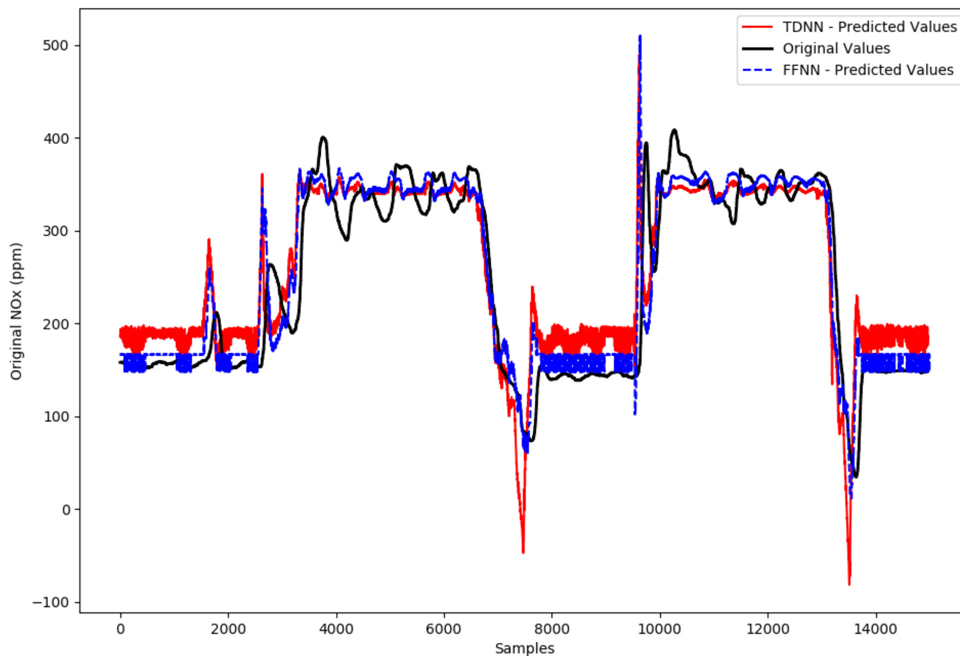


Figure 6.83: Final comparison between experimental NOx (Original Values), FFNN and TDNN trajectories.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, several neural network models were designed in order to predict some fundamental engine parameters. The output estimation was achieved either through static models (FFNN) or neural networks with input history (TDNN). The latter type was selected in order to take into account the previous stages of a system when calculating the target quantity. A comparison of both models results has been carried out in order to evaluate precision and generalization in the various predictions. The models were tested on two different testing data sets; a relevant to the training set and an other completely unknown. The predictions of the relevant testing set (part of the initial data set) were expectedly better, because of their comparable distribution to the training data. As a matter of fact, all networks had a successful performance, with testing errors almost always below 10-15%. In λ prediction models the TDNN model showed a better performance, while in Fuel Consumption, MAP and NOx models the FFNN architecture dominated. However, these results are not generic, but specific to the combination of the selected inputs and hyper-parameters. As far as the TDNN models are concerned, they might need a less complex structure due to their multiple inputs or a larger delay between the inputs of the same quantity. Having said that, it does not mean that the final performance was not satisfactory. In conclusion, all networks succeeded into predicting the dynamics of all quantities, both inside their training range with familiar data patterns and on completely unknown data, even in the most noisy regions or where spikes occurred.

7.2 Suggestions for Future Work

In this work, ANN models are designed for the prediction of *Fuel Consumption*, *MAP*, λ and *NOx*, with already extracted data used for the training process. In a next step, other fundamental engine parameters could be predicted by the already designed models. In addition, the models could be tested during real-time operating conditions, possibly with a RNN architecture, where the outputs from previous steps are fed as inputs to the current step. Also, an other interesting idea would be to design neural network models where some of the inputs are outputs of other models. For instance, some quantities that cannot be measured by a real sensor could be evaluated by NN models, as this thesis and many other researches propose. Afterwards, these values could be used as inputs to other NN models, predicting different engine variables. Of course, alternative model structures with more or

less hidden layers and neurons could be implemented, as well as experimentation on the default structure of Keras built-in activation functions, optimizers and the like.

Bibliography

[1] D. Woodyard, "Chapter ten - man b&w low-speed engines," in *Pounder's Marine Diesel Engines and Gas Turbines (Ninth Edition)* (D. Woodyard, ed.), pp. 289 – 360, Oxford: Butterworth-Heinemann, ninth edition ed., 2009.

[2] I. Arsie, A. Cricchio, M. D. Cesare, F. Lazzarini, C. Pianese, and M. Sorrentino, "Neural network models for virtual sensing of nox emissions in automotive diesel engines with least square-based adaptation," *Control Engineering Practice*, vol. 61, pp. 11 – 20, 2017.

[3] I. Arsie, C. Pianese, and M. Sorrentino, "Development of recurrent neural networks for virtual sensing of nox emissions in internal combustion engines," *SAE International Journal of Fuels and Lubricants*, vol. 2, pp. 354–361, 03 2010.

[4] S. Tasdemir, I. Saritas, M. Ciniviz, and N. Allahverdi, "Artificial neural network and fuzzy expert system comparison for prediction of performance and emission parameters on a gasoline engine," *Expert Syst. Appl.*, vol. 38, pp. 13912–13923, 05 2011.

[5] C. M. Bishop, *Pattern Recognition and Machine Learning.* Cambridge, UK: Springer, 2006.

[6] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow.* 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2019.

[7] H. Li, K. Butts, K. Zaseck, D. Liao-McPherson, and I. Kolmanovsky, "Emissions modeling of a light-duty diesel engine for model-based control design using multi-layer perceptron neural networks," 03 2017.

[8] M. D. Cesare and F. Covassin, "Neural network based models for virtual nox sensing of compression ignition engines," in *SAE Technical Paper*, SAE International, 09 2011.

[9] J. Chen, "Neural network." `https://www.investopedia.com/terms/n/neuralnetwork.asp`, 2020.

[10] M. Caudill, *Neural Network Primer: Part I.* AI Expert, February 1989.

[11] A. Minin, *THE NEURAL-NETWORK ANALYSIS, DATA FILTERS.* JASS, 2006.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning.* Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

[13] Y.LeCun, B.Boser, J.S.Denker, D.Henderson, R.E.Howard, W.Hubbard, and L.D.Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[14] L. NOAKES, "The takens embedding theorem," *International Journal of Bifurcation and Chaos*, vol. 01, no. 04, pp. 867–872, 1991.

[15] M. H. Hassoun, *Fundamentals of Artificial Neural Networks.* Cambridge, MA, USA: MIT Press, 1st ed., 1995.

[16] K. AG, *Flow Measurement: Fuel Consumption Measurement for Diesel Engines.* 10 2017.

[17] L. R. of Shipping, *Emissions of Nitrogen Oxides from Marine Diesel Engines.* 71 Fenchurch Street, London EC3M 4BS, UK: ©Lloyd's Register of Shipping, July 2002.

[18] "Nitrogen oxides (nox) - regulation 13." `http://www.imo.org/en/OurWork/Environment/PollutionPrevention/AirPollution/Pages/Nitrogen-oxides-(NOx)-Regulation-13.aspx`, 2020.

[19] M. Okubo and T. Kuwahara, "Chapter 4 - operation examples of emission control systems," in *New Technologies for Emission Control in Marine Diesel Engines* (M. Okubo and T. Kuwahara, eds.), pp. 145 – 210, Butterworth-Heinemann, 2020.

[20] N. Planakis, G. Papalambrou, and N. Kyrtatos, "Integrated Load-Split Scheme for Hybrid Ship Propulsion Considering Transient Propeller Load and Environmental Disturbance," *Journal of Dynamic Systems, Measurement, and Control*, 09 2020.

[21] J. Martin and C. Elster, "Detecting unusual input to neural networks," 2020.

[22] P. Dimitrakopoulos, "Real-time virtual sensor for nox emisions and stoichiometric air-fuel ratio lambda of a marine diesel engine using neural networks," 2019.

[23] F. Chollet *et al.*, "Keras documentation." `https://keras.io/api/`, 2015.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[25] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[26] M. Waskom, "seaborn: statistical data visualization," 2020.

[27] M. Chester, "Correlating energy data sets: The right way and the wrong way." `https://chesterenergyandpolicy.com/2017/09/21/correlating-energy-data-sets-the-right-way-and-the-wrong-way/`, 09 2017.

[28] Z. Kotulski and W. Szczepiński, *Basic Characteristics of Error Distribution; Histograms*, pp. 1–13. Dordrecht: Springer Netherlands, 2010.

[29] J. Brownlee, "How to use learning curves to diagnose machine learning model performance." `https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/`, 2019.