Estimation of Marine Engine Operating Parameters using LSTM Neural Network

Athanasios Koufoudakis

Diploma Thesis

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

Supervisor:Professor George Papalambrou

Committee Member : Prof. G. Grigoropoulos
Committee Member : Associate Prof. C. Papadopoulos

July 2022

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 Professor George Papalambrou for giving me the chance and motivation to work on this topic. I would also like to thank him for his patience, continuous support and immense knowledge. His guidance helped me in all the time working on this thesis.

Also I would like to thank Professor G. Grigoropoulos and Associate Professor C. Papadopoulos for evaluating my work and being members of my supervisors committee.

Abstract

In this thesis the implementation of Long short term memory neural network models for engines virtual sensors is investigated. Initially, different kinds of models are been presented. Each model is predicting a different engines operating parameter (NOx, Lambda, etc), several models where investigated. The aim is to find models which are accurate and computationally efficient, so that the virtual sensor would be able operate in real time.

LSTM neural network for marine engines, using raw measurement data from laboratory measurements, are developed and verified. These models can be utilised as virtual sensors of engine-out NOx emissions, lambda (λ) and other operation parameters. Investigations for the optimal neural network configuration targeting models were carried so as they can capture the dynamic behaviour of a marine diesel engine, can generalise within the training range and have the minimum complexity due to execution performance and portability reasons.

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. The modeling carried out Julia language environment in particularly using Flux library.

Contents

Chapter 1

Introduction

1.1 Framework

Nowdays the concern about the environmental impact of marine industry and operations is getting bigger pushing the environmental regulations to get restricted over and over again. Marine industry have put great effort to compile with these regulations and find solutions for the sustainability of the industry. Huge amounts of funding have been poured in researches projects in order to develop new and efficient methods to compile with the environmental regulations. Optimized controls, emissions and fuel consumption reduction strategies have been widely accepted in marine industry in order to compile with regulation and reduce the operation cost.

Some of these systems are the exhaust gas recirculation (EGR), consist a highly efficient system to reduce the NOx formation although it is very complex, and the exhaust after treatment systems such as selective catalytic reduction system (SCR) which operate by inducing chemical reactions in the engine's exhaust gases, harmful substances are transformed into ecologically benign constituents. However in the recent year control systems have met significant growth in marine engine industry. Control systems are capable of analyze and processes various data it receives from sub systems such as sensors, which not necessary belong in engine's test-bed, and determine the main factors of engine's operation in order to achieve the most efficient results and respond to demanding situations.

These systems increasingly link to the physical world. Technological advancements and declining unit costs of sensor technology combined with increased connectivity drive the spread and complexity of the Internet of Things or so-called cyber-physical systems. Billions of sensors feed information systems (IS) with data describing physical phenomena – such as temperature, pressure, humidity, velocity, chemical components, or material composition – across many areas ranging from industrial applications (e.g., smart factories) to consumer applications (e.g., smart watches). They form a key foundation for AI-based information systems that apply machine learning and generate analytics-based solutions. In particular, sensor data represents an essential building block of digital twins. As digital duplicates of real assets in the physical world, they rely on sensor technology for continuous data acquisition, may be used to optimize the production process by means of simulation or to develop predictive maintenance services.[1]

Physical sensors tend to be completely replaced by victuals, advantages of virtual sensor in comparison with the physical expanding to many fields from cost up to efficiency and accuracy. For example in marine industry, in which physical environment in the exhaust

system of engine is very harsh, and physical sensors have to be replaced sometimes after less than 100 hours of operation. However average vessel voyage duration is about 8,000 hours a year. Additionally the sensors have to be recalibrated after a short period, which is time-consuming and expensive.

1.2 Literature Review

The main goal is to find the way to implement Long-Short Term Memory (LSTM) neural network based sensor. For this reason it is highly important to understand the basic theory of artificial intelligence by the aid of [2] and how to apply it in julia environment, according to [3]. LSTM architecture has been investigated as in [4], and applied to predict various engines operation variables. Since LSTM neural network is a special kind of Recurrent neural network, some researches as [5] and [6] have been studied, giving high ambitions to LSTM project due to excellent results that RNNs have produced. In particular, in [5] , RNN models have been used to predict NO emission, with an estimation error lower than 4% while in [6] real time predictions have been carried out, using a prototype of ECU, for NOx emissions, with accuracy range from $R^2 = 91\%$ up to $R^2 = 99\%$. Finally, all the above perspectives and previous works using RNNs, such as [7] and [8], but LSTM neural networks also, like [9], have been taken into account.

1.3 Thesis structure

In this thesis, LSTM neural network model structure 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). Predicted quantities are: Fuel Consumption (kg/h), Intake Manifold Absolute Pressure MAP (kPa), Lambda (λ), Engine Torque (Nm) and NOx emissions (ppm). LSTM neural network is capable to take into account not only its input signals but also the powertrain's previous states to calculate its predictions, producing accurate results especially when it comes for non-linear quantities such as NOx emissions.

In Chapter 2, the classical theory concerning the artificial neural networks is presented while an overview of RNN and LSTM neural network are introduced. In Chapter 3, the experimental setup and the various engine parameters to be predicted are described along side with experimental data and the preprocessing technique. In Chapter 4, LSTMs model structure requirements are presented while model configuration carried out. Additionally after the model design, input variables and network hyperparameters are selected. Then the training process of the target variables are presented in Chapter 5, along side with the necessary result figures. Finally the conclusions of this work Chapter 6.

Chapter 2

Neural Networks and Deep Learning

2.1 Definition of Neural Networks

A Neural network is nothing more than a series of algorithms, inspired by the biological neural networks of human brain, capable of recognizing the pattern and the underlying relationships between a set of provided data. Neural networks, also referred as artificial neural networks (ANN), can adapt, changing inputs in order to achieve the best outputs accuracy.
The key factor that makes neural networks one of the most significant prediction tools in our time, is that they learn by example. However the selected examples must be carefully chosen and process, if that necessary, otherwise the network will distracted, time cost will increase and even worst the model could disfunctioning.

2.2 Machine Learning Models Categories

All machine learning models are categorized as either supervised or unsupervised. When the model is a supervised model, it's then sub-categorized as either a regression or classification model as presented in figure 2.1.

- **Supervised Learning**: Supervised learning involves learning a function that maps an input to an output based on example input-output pairs:
    1. **Regression:**Regression models predicts continuous values. The neural networks developed in this thesis are considered a subcategory of regression modeling. Some of the most common types are:Linear Regression, Decision Tree, Random Forest.
    2. **Classification:** In this kind of models the output is discrete. The output variables are often called "labels" or "categories" and the mapping function predicts the class or category for a given observation. Some popular types are:Logistic Regression, Support Vector Machine, Naive Bayes.
- **Unsupervised Learning:** Unlike supervised learning, unsupervised learning is used to find patterns from input data without references to labeled outcomes. Two main methods used in unsupervised learning include clustering and dimensionality reduction.

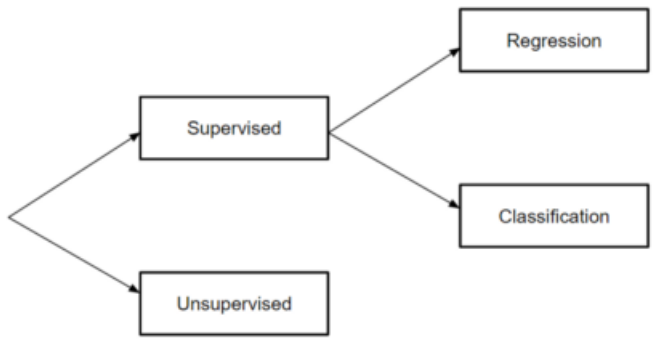


Figure 2.1: Categories of machine learning models.

2.3 Linear Basis Function Models

Before getting into the details of deep neural networks, the basics of neural network training have to be covered. This task is focused on the training process, including defining simple neural network architectures, handling data, specifying a loss function, and training the model. Fortunately, classic statistical learning techniques such as linear and softmax regression can be cast as linear neural networks.

2.3.1 Linear Regression and Basis Elements

Regression refers to a set of methods for modeling the relationship between one or more independent variables and a dependent variable. In the natural sciences and social sciences, the purpose of regression is most often to characterize the relationship between the inputs and outputs. Machine learning, on the other hand, is most often concerned with prediction.

Linear Model

Linear regression is one of the most popular tools to regression. The fundamental assumption of linear regression is the relationship between the independent variables $\mathbf{x}$ and the dependent variable y to be linear. That means that variables $\mathbf{y}$ could be expressed as a weighted sum of the elements in x, given some noise of the observations.
Given a dataset, the goal is to choose the weights $\mathbf{w}$ and the bias b such that on average, the predictions made according to model best fit the true quantities observed in the data. Models whose output prediction is determined by the affine transformation of input features are linear models, where the affine transformation is specified by the chosen weights and bias.
In machine learning, high-dimensional datasets are often used, so it is more convenient to employ linear algebra notation. When inputs consist of d features, prediction expressed as $\hat{y}$ (in general the "hat" symbol denotes estimates) :

$$\hat{y} = w_1 x_1 + ... + w_d x_d + b. \tag{2.3.1}$$

Summarizing all features into a vector $\mathbf{x} \in \mathbb{R}^d$ and all weights into a vector $\mathbf{w} \in \mathbb{R}^d$ the above model could be expressed as:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b. \tag{2.3.2}$$

Before start searching for the best parameters $\mathbf{w}$ and $\mathbf{b}$, two more steps will be needed, first a quality measure for some given model and second a procedure for updating the model to improve its quality.

Loss Function

Loss Function is measured tool in order to quantifies the distance between the real and the predicted value of the target. The loss will usually be a non-negative number where smaller values are better and perfect predictions incur a loss of 0. The most popular loss function in regression problems is the squared error. When prediction for element i is $\hat{y}^{(i)}$ and the corresponding true value is $y^{(i)}$, the squared error is given by:

$$l^{(i)}(\mathbf{w}, b) = (\hat{y}^{(i)} - y^{(i)})^2. \tag{2.3.3}$$

The empirical error for a give dataset considered to be only a function of the model parameters. In figure 2.2 a regression problem for one dimensional case is shown.

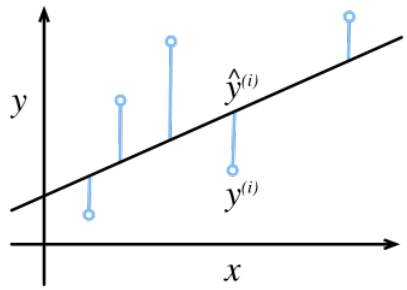


Figure 2.2: Fit data with a linear model.

Note that large differences between estimates $\hat{y}^{(i)}$ and observations $y^{(i)}$ lead to even larger contributions to the loss, due to the quadratic dependence. The quality of a model on the entire dataset of n examples, is measured by simply average (or equivalently, sum) the losses on the training set:

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^T\mathbf{x}^{(i)} + b - y^{(i)})^2. \tag{2.3.4}$$

When training the model, the goal is to find the parameters $(\mathbf{w}^*, b^*)$ that minimize the total loss across all training examples:

$$\boldsymbol{w}^*, b^* = argmin_{w,b} L(\mathbf{w}, b). \tag{2.3.5}$$

The linear relationship between the input variable $\mathbf{x}$ and the predicted variable $\mathbf{y}$ however imposes significant limitations on the model.[10]

Linear Basis Function Models

An alternative form to linear model for regression could be constructed by considering linear combinations of fixed nonlinear functions of the input variables, of the form (the bias b could be considered as w_0):

$$\hat{y}(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{n} w_i \phi_i(\mathbf{x}) \tag{2.3.6}$$

where the $\phi_i(x)$ are known as basis functions.
it is often convenient to define an additional dummy basis function $\phi_0(x) = 1$ so that

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \sum_{i=0}^{n} w_i \phi_i(\mathbf{x}) \tag{2.3.7}$$

where $\mathbf{w} = (w_0, ..., w_n)^T$ and $\phi = (\phi_0, ..., \phi_n)^T$. In many practical applications of pattern recognition, fixed preprocessing forms are applied or feature extraction to the original data variables. If the original variables comprise the vector x, then the features can be expressed in terms of the basis functions $\phi_i(\mathbf{x})$.
By using nonlinear basis function, the function $\hat{y}(\mathbf{x}, \mathbf{w})$ is allowed to be nonlinear function of the input vector $\mathbf{x}$. Functions of the form 2.3.6 are called linear models, however, because this function is linear in w. This linearity in the parameters will significantly simplify the analysis of this class of models.
The example of polynomial regression model is a particular example of this model in which there is a single input variable x, and the basis functions take the form of power of x so that $\phi_i(x) = x^i$. 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.
There are many other possible choices for the basis functions for example

$$\phi_i(x) = exp\left\{ -\frac{(x - \mu_i)^2}{2s^2} \right\} \tag{2.3.8}$$

where the μ_i govern the locations of the basis functions in input space, and the parameter s governs their spatial scale. These are usually referred to as Gaussian basis functions, although it should be noted that they are not required to have a probabilistic interpretation, and in particular the normalization coefficient is unimportant because these basis functions will be multiplied by adaptive parameters w_i. Another possibility is the sigmoidal basis function of the form:

$$\phi_i(x) = \sigma\left(\frac{x - \mu_i}{s} \right) \tag{2.3.9}$$

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

$$\sigma(a) = \frac{1}{1 + exp(-a)}. \tag{2.3.10}$$

Equivalently, the 'tanh' function could be also used 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.

2.3.2 Limitations of Fixed Basis Functions

In previous task have been focused on models comprising a linear combination of fixed nonlinear basis functions. 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 nonlinearities of model could 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 focused on more complex models such as support vector machines and neural networks. The difficulty stems from the assumption that the basis functions $\phi_i(x)$ 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 could be exploited to help alleviate this problem. First of all, the data vectors 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 feed-forward neural network, also known as the multilayer perceptron, operates by fixing the number of basis functions in advance but at the same time allow them to be adaptive, in other words to use parametric forms for the basis functions in which the parameter values are adapted during training, therefore feed-forward neural networks could be considered as the most successful model of this type in the context of pattern recognition. In fact, multilayer perceptron is really a misnomer, because the model comprises multiple layers of logistic regression models (with continuous nonlinearities) rather than multiple perceptrons (with discontinuous nonlinearities).
The linear models for regression and classification are based on linear combinations of fixed nonlinear basis functions $\phi_j(x)$ and take the form

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right) \tag{2.4.1}$$

where $f(\cdot)$ is a nonlinear activation function in the case of classification and is the identity in the case of regression. The main objective is to ectend this model by making the basis functions $\phi_j(x)$ depend on parameters and the 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 2.4.1, 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

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.4.2}$$

where i=1,...,M and the superscript (1) indicates that the corresponding parameters are in the first layer of the network. The parameters $w_{ji}^{(1)}$ should be reffered as weights and the parameters $w_{j0}^{(1)}$ as biases, as mentioned before.
The quantities a_j are known as activations. Each of the is then transformed using a differentiable, nonlinear activation function h(·) to give

$$z_j = h(a_j) \tag{2.4.3}$$

These quantities correspond to the outputs of the basis functions in 2.4.1 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 2.4.1, these values are again linearly combined to give output unit activations

$$a_k = \sum_{j=1}^{n} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{2.4.4}$$

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 $y_k = a_k$.
These various stages could be combine in order to give the overall network function that, for sigmoidal output unit activation functions, takes the form

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{2.4.5}$$

where the set of all weight and bias have been grouped together into a vector 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 w of adjustable parameters.

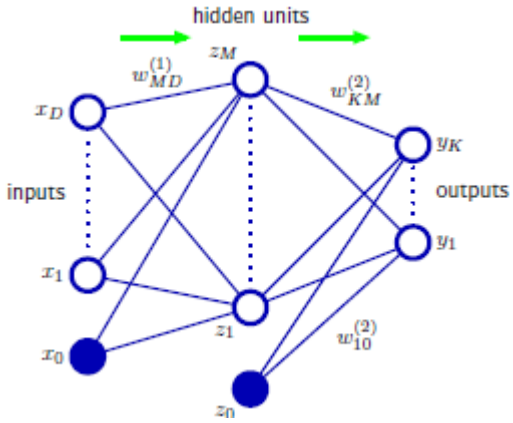


Figure 2.3: Network diagram for the two-layer neural network corresponding to 2.4.5. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and 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 2.3. The process of evaluating 2.4.5 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 2.4.2 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 2.4.2 takes the form

$$a_j = \sum_{j-0}^{D} w_{ji}^{(1)} x_i. \tag{2.4.6}$$

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

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right) \tag{2.4.7}$$

As can be seen from Figure 2.3, the neural network model comprises two stages of processing, and for this reason the neural network is also known as the multilayer perceptron, 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 could easily 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 2.3 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 2.4.4 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 2.3 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).

2.4.1 Network Training

In the previous tasks, neural networks have been viewed as a general class of parametric nonlinear functions from a vector x of input variables to a vector 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 x_n, where $n = 1, ..., N$, together with a corresponding set of target vectors t_n, the error function is minimized

$$E(\mathbf{w} = \frac{1}{2}\sum_{n=1}^{N} \|y(x_n, w) - t_n\|^2. \tag{2.4.8}$$

However, a much more general view of network training could be provided by first giving a probabilistic interpretation to the network outputs. Here the probabilistic predictions will also provide a clearer motivation both for the choice of output unit nonlinearity and the choice of error function.

Starting with regression problems and considering a single target value t that can take any real value, it is also assumed that t has a Gaussian distribution with an x-dependent mean, which is given by the output of the neural netwok so that

$$p(t|x, \mathbf{w}) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}) \tag{2.4.9}$$

where β is the precision (inverse variance) of the Gaussian noise. For the conditional distribution given by 2.4.9, it is sufficient to take the output unit activation function to be the identity, because such a network can approximate any continuous function from x to y. Given a data set of N independent, identically distributed observations $X = x - 1, ..., x - N$, along with corresponding target values $t = t_1, ..., t - N$, the corresponding likelihood function could be constructed

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{n} p(t_n|x_n, w, \beta). \tag{2.4.10}$$

Taking the negative logarithm, the error function is obtained

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 - \frac{N}{2}\ln\beta + \frac{N}{2}\ln(2\pi) \tag{2.4.11}$$

which can be used to learn the parameters 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 w. Maximizing the likelihood function is equivalent to minimizing the sum-of-squares error function given by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 \tag{2.4.12}$$

where additive and multiplicative constants have been discarded. The value of w found by minimizing $E(w)$ will be denoted w_{ML} because it corresponds to the maximum likelihood solution. In practice, the nonlinearity of the network function $y(x_n, w)$ causes the error $E(w)$ to be nonconvex, and so in practice local maxima of the likelihood may be found, corresponding to local minima of the error function.
Having found $w - ML$, the value of β can be found by minimizing the negative log likelihood to give

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(\mathbf{x_n}, \mathbf{w_{ML}}) - t_n\}^2. \tag{2.4.13}$$

Note that this can be evaluated once the iterative optimization required to find 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

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{2.4.14}$$

which could be used when discussing error backpropagation.

2.4.2 Parameter Optimization

The next task is focused on finding a weight vector W which minimizes the chosen function $E(w)$. At this point, it is useful to have a geometrical picture of the error function, which is presented as a surface sitting over weight space as shown in Figure 2.4.

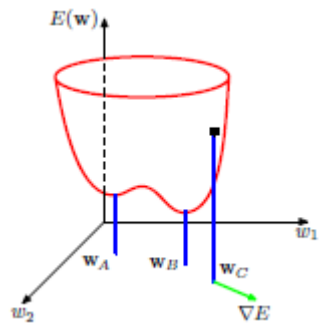


Figure 2.4: Geometrical view of the error function (w) as a surface sitting over weight space. Point w_A is a local minimum and w_B is the global minimum. At any point w_C, the local gradient of the error surface is given by the vector ∇E.

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

$$\nabla E(\mathbf{w}) = 0 \tag{2.4.15}$$

as otherwise making a small step 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 objective 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 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 2.3, 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 $\nabla E(w) = 0$ an 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 $w^{(0)}$ for the weight vector and then moving through weight space in a succession of steps of the form

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \boldsymbol{\Delta}\mathbf{w}^{(\tau)} \tag{2.4.16}$$

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

2.4.3 Gradient descent optimization

The simplest approach to using gradient information is to choose the weight update in 2.4.16 to comprise a small step in the direction of the negative gradient, so that

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E(\mathbf{w}^{(\tau)} \tag{2.4.17}$$

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. Error functions based on maximum likelihood for a set of independent observations comprise a sum of terms, one for each data

point

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{2.4.18}$$

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

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}. \tag{2.4.19}$$

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.

2.4.4 Error Backpropagation

The main goal in this section is to find an efficient technique for evaluating the gradient of an error function $E(w)$ for a feed-forward neural network. 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 stages could 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.

Evaluation of error-function derivatives

In this task 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

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}) \tag{2.4.20}$$

Here the problem of evaluating $E - n(\mathbf{w})$ 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 combinations of the input variables x_i so that

$$y_k = \sum_k (y_{nk} - t_{nk})^2 \tag{2.4.21}$$

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

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj})x_{ni} \tag{2.4.22}$$

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} with the input end of the link.[2]

2.5 LSTM Neural Network

2.5.1 Recurrent Neural Networks

Human mind has the capability to save information coming from the environment and processing them based on previous memories and experiences, in other case it would be very difficult and time consuming to handle these information and reach a thesis or a opinion. The ability described above, is not applicably on traditional neural networks, and this fact is one of the most major issues of traditional neural networks. Lets consider a case which someone wants to classify the different kinds of actions taking place in a theatrical performance, traditional neural networks are incapable of using, in direct way, previous events (information) to improve and describe the later ones, that's why complicated and confused methods are being used expanding the demanded memory and time needed to complete the process. This issue can be resolved by using Recurrent neural networks. This kind of networks are using loops to save and recall the information producing a better outcome.

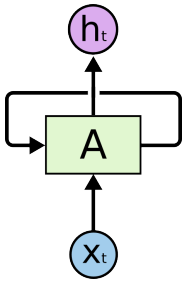


Figure 2.5: Recurrent neural networks.

In the above diagram the construction of the recurrent neural network is being shown. Some input variable $\boldsymbol{x_t}$ produces an output variable $\boldsymbol{h_t}$ using a loop which allows the information to flow from one step, of the network, to the next one.

A proper way to understand the structure of a RNN is to consider it as the multiple copies of the same network in which each one provides the information to the next one. In the diagram that follows, the unroll loop is presented:

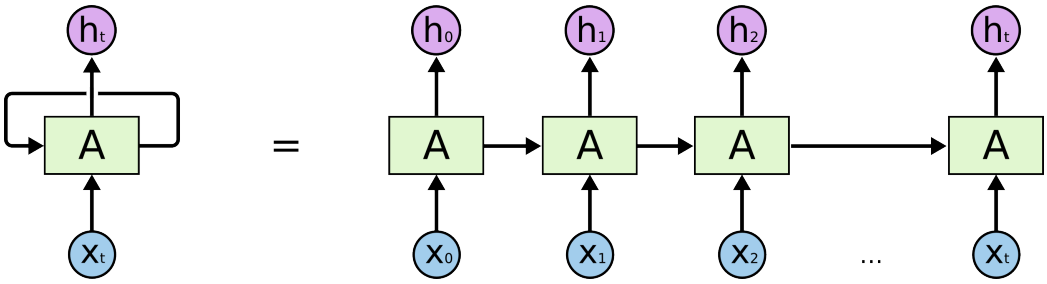


Figure 2.6: Unroll RNN.

The chain form of RNN architecture provides a great advantage, to the network, of handling and processing lists and sequences. In the last years there have been remarkable progress and success applying RNNs to different kind of problems such as speech recognition, language modeling, translation, image captioning and so on.

LSTMs has a key role to this success. Long-Short Term Memory consist a unique and very effective type of recurrent neural network. The vast part of the progress, which has taken place in rnns, has been achieved using LSTMs.

2.5.2 Long Term Dependencies

One of the fundamentals ideas behind the RNNs is the capability of recall and connect previous information to the present task, in order to achieve better and faster results. In some cases only the latest information can be used to perform the present task and in some others only the very earliest.

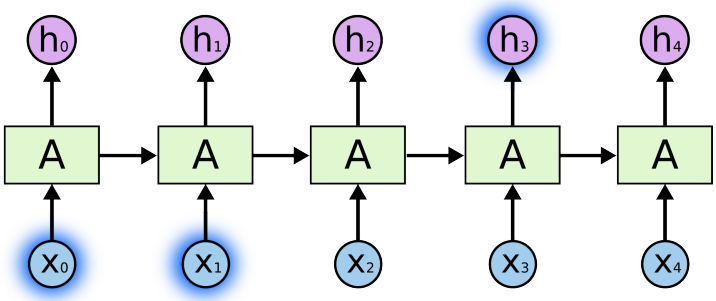


Figure 2.7: RNN short term dependency.

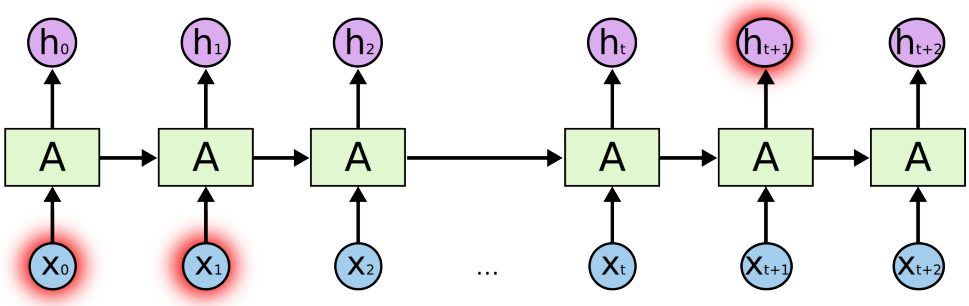


Figure 2.8: RNN long term dependency.

In theory RNNs are completely capably of dealing with that kind of long term dependencies but in practice it seems that as the gap between the necessary information and the point where it is needed become very large RNNs capability of handling these data, reduces dramatically and the network becoming unable to connect and use the previous information.
One of the reasons why the LSTM networks finds a wide range of applications is that LSTMs are absolute sufficient of dealing and handling these huge gaps data-sets.

2.5.3 LSTM Networks

Long Short Term Memory networks first introduced by **Hochreiter & Schmidhuber (1997)** [4]. LSTM is nothing more than a special type of RNN, specialized dealing with long term dependencies. LSTMs design to remember and recall information for long time periods as default.
All RNNs can be described as a chain of repeating modules of neural networks. Common RNNs repeating modules consist of a simply structure such as a single **tanh** layer which presenting in the below diagram:

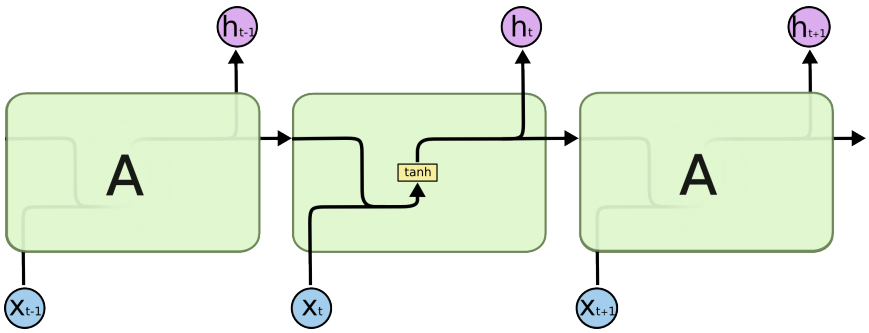


Figure 2.9: The repeating module in a standard RNN contains a single layer.

The repeating module of LSTMs is similar to RNNs but instead of a single layer there are four, connecting to each other with a very innovative and unique way.

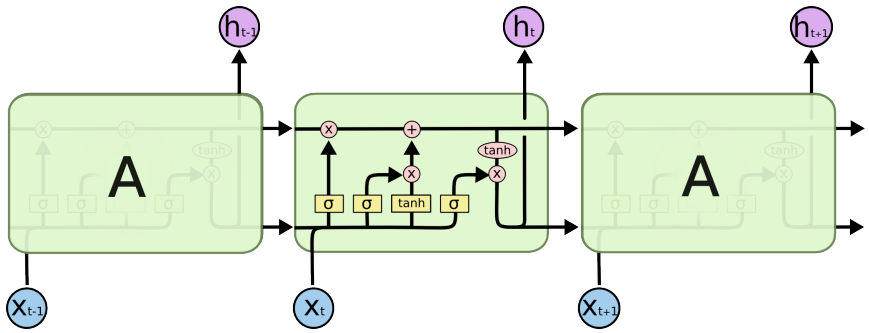
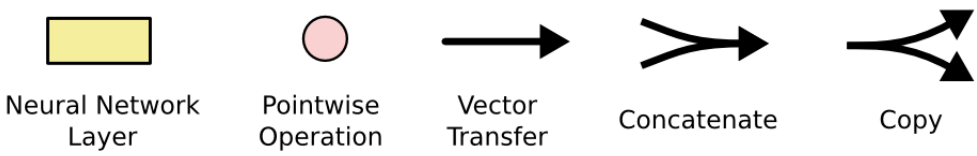
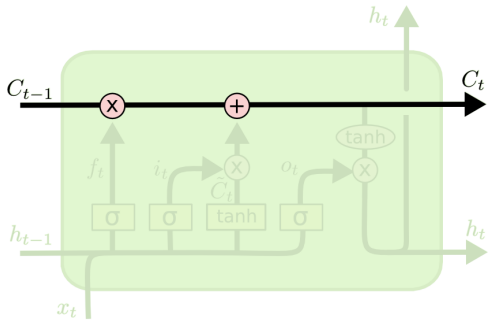


Figure 2.10: The repeating module in LSTM contains four interacting layers.

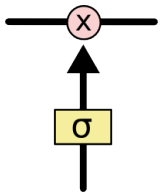
In the above diagram each line carries an entire vector, the pink circles represent pointwise operations and the yellow boxes are learned neural network layers.



One of the majors ideas behind LSTMs is the **cell state**, the horizontal line shown in the diagram.



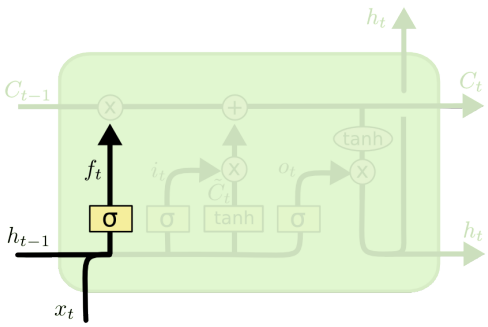
The cell state passes thought the entire chain with only some minor linear interactions, allowing information flow across the chain unchanged.
Also the LSTMs could remove or add information to the cell state, the structures which give LSTMs this ability is called **gates**. Gates allow optionally information to get through and they are composed out of a sigmoid neural net layer and a pointwise multication operation.



The sigmoid layer operates like a information valve, that means that regulate the flow of information passing through, a value of zero means that nothing gets through while a value of one means everything gets through. LSTMs consist of three gates to control the cell state.

2.5.4 LSTMs Operation

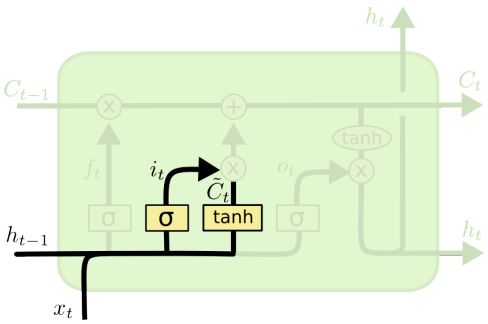
Operation of LSTMs begins with the choice which part of information would be allowed to pass through to cell state and which to throw away from it. The gate which makes that decision is called **forget gate layer**. Receiving as inputs h_{t-1} and x_t, produces an output number between 0 and 1 for each number in cell state C_{t-1}. As mentioned before value of 1 means absorb all the information while 0 throw it all away.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

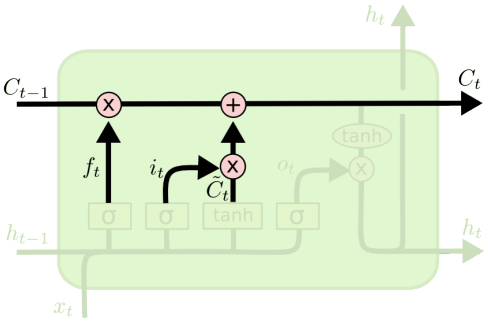
While the process continues another decision is to be made, in this case is all about what new information is going to saved in the cell state, this kind of choice composed out of two parts. The first part is taking place when a gate called **input gate layer** decides which layers would be updated and the second part when a simple *tanh* layer creates a vector

of new candidates values or predicted values $\tilde{C}_t$ that might be added to the state. These two different values are combined, creating an update to the state.



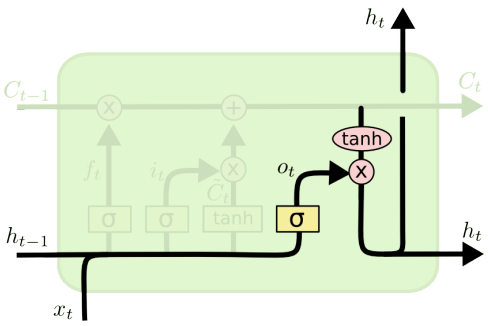
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

The next step is to update the previous cell state C_{t-1} into the new one C_t. The decision has already been made in the previous step, so it just materialized it.
The updated cell state calculated as the multiplication of the old state C_{t-1} by f_t, ignoring the information which has been forgotten in the first step, then added the product of $i_t * \tilde{C_{t-1}}$. The updated cell state is the new candidate value, new prediction, scaled by the decision which took place in the second step of the process, how much each state value is to be updated.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In the final step one more decision is to be made, what kind of information, what output is going to be extract to be used in the next module. This output will be nothing more than a filtered version of the cell state. A sigmoid layer defines the parts of the cell state is going to extracted and then the updated cell state passing throw a tanh, so the new cell state to be compressed between 0 and 1, and finally these two values multiplied together and produce only the parts of output which have been decided earlier to be promoted to the next module.[11]



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

2.5.5 LSTMs Discussion

As previously mentioned in the paper written by **Hochreiter & Schmidhuber (1997)** a detailed research has taken place revealing the limitations and the advantages of traditional LSTM, and others variants of it, in comparison with others neural networks types such as Real Time Recurrent Learning (RTRL), Backpropagation through time (BPTT) etc. [4]

Limitations of LSTM:

- Each memory cell block needs two additional units, one input and one output gate. This fact increases the number of weights up to 9 times in comparison with traditional recurrent networks, due to the fact that each conventional hidden unit (there are three) is replaced by at most 3 units in the LSTMs architecture, increasing the number of weights by 3^2 at fully connected case.
- LSTM seems to dealing with the similar problems to those of feedforward nets, receiving the entire input string at once. These problems caused by the constant error flow through Conventional Excitation Control System (CECS).
- LSTM doesn't appeal to dealing with recency problems that go beyond of other approaches. On the other hand all gradient-based approaches are shown high inefficiency and difficulty to precisely count discrete time steps. If it makes a difference whether a certain signal, appealed 99 or 100 steps ago, necessary accounting mechanism has to be a developed. Simplest tasks however doesn't seems to create any serious problems to LSTM.

Advantages of LSTM:

- LSTM is highly efficient when dealing with very long time lags in case of problems with the notion of recency due to constant error backpropagation within the memory cells.
- When it comes to long time lag problems LSTM could successfully deal with noise, distributed representations and continuous values.
- LSTM doesn't need parametric fine tuning, it operates efficiently over a wide range of parameters like learning rate, input and output gate bias. Even in case of a large leaning rate this fact cause the outputs gates to be decreased and take values near to zero, automatically compensating the negative effects mentioned before.
- Finally LSTM algorithm's update complexity per weight and time step is similar to BPTT but is also local in space and time. In comparison with Real-Time Recurrent Learning (RTRL), Back Propagation Through Time (BPTT), Recurrent Cascade Correlation (RCC) and Neural Sequence Chunking (NSC), LSTM produces more accurate and faster results. [4]

Chapter 3

Operating Parameters and Data Preparation

3.1 Operating Parameters of Marine Diesel Engine

There are different kinds of parameters that define the operation of a Diesel engine. Neural networks are developing to predict some of those variables in order to characterize engine functionality and the efficiency. However in recent years neural networks developed this way, also detecting problems and issues that engine might have. Some of those parameters will be presented below.

3.1.1 Nitrogen Oxides (NOx)

As the marine industry around the world continually expanding, the concerns about air pollution caused by merchant ships is grown. Nitrogen oxides (NOx) and Sulphur oxides (SOx) consist the two main pollutants, caused by ships operations, these gases have adverse effects on ozone layer in the troposphere area of earth's atmosphere causing the green house effect and global warming. International Maritime Organization and other maritime organization have put forward rigorous legal requirements in order to reduce marine diesel engine emissions. That's one of the reasons why the research about reducing NOx emissions has attracted great interest of marine industry.
Among others, regulations about NOx emissions keep getting tougher and more difficult to apply. As it referred in the following table, according to International Maritime Organization (IMO) for all diesel engines of over 130 kW output power.[5]

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 ≥ 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 3.1: IMO restrictions about NOx emissions [12]

3.1.2 Fuel Consumption

In recent years fuel consumption of marine Diesel engine has been on of the most popular fields, where research is taking place. Solutions about ship emissions and voyages costs have already been given by developing Ship Energy Efficiency Management Plan, also known as SEEMP.
The primary objective of this plan is to improve ship's efficiency during a voyage by enforcing correct and optimized methods in order to achieve reduction of fuel consumption and to avoid unnecessary air pollution.

3.1.3 Air-fuel equivalence ratio and Lambda (λ)

Diesel engines needs fuel and oxygen (from air) to produce energy through combustion. To succeed a efficient combustion certain quantities of fuel and air have to be supplied to engine. A complete combustion performed only when all the given fuel is burnt and no quantities of unburnt fuel consist in exhaust gas.
Air fuel ratio is nothing more that the ratio of air and fuel contained in a mixture prepared for combustion. Approximately the air fuel ratio in order to succeed a complete combustion in a diesel engine is 14.4 to 1, and this analogy is called stoichiometric air fuel ratio. In practice the AFR are usually maintained higher that 25 to 1, to avoid excessive smoke formation.
The air fuel ration or AFR is calculated as the ration between air mass m_a and fuel mass m_f used during engine's operation:

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

lambda air fuel ration is considered the ration between the actual air fuel ration and the stoichiometric one:

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

3.1.4 Exhaust Gas Recirculation System (EGR)

Exhaust Gas Recirculation is a system that leads the exhaust gases back into the intake manifold in order to achieve reduction in Nitrogen Oxides emissions. There are two different types of EGR:

- **internal exhaust gas recirculation(iEGR)**:Exhaust gases are driven back in the cylinder by overlapping the opening time of intake and exhaust valves.
- **external exhaust gas recirculation EGR**:Exhaust gases are recirculated back into intake manifold through the external duct and an additional valve called EGR valve.

On diesel engines external EGR is widely used while significant reduction of NOx emission has been achieved.

3.2 Data preparation

The most important and essential factor in machine learning is the data set which the neural network are trained. Missing or invalid data causes crucial problems in algorithm reducing model's accuracy while in some cases could even lead to confusing results. If information given to the network is noisy or misleading then knowledge discovery could be very difficult during training process. Data preparation is also considered as one of the most difficult steps in any machine learning project due to the fact that each dataset is different and unique for the project. In any case good preparation of the given data is required to produce clean and accurate outcomes and to reduce the computation cost of training.

3.2.1 Data collection LME facility

Laboratory of Marine Engineering at the School of Naval Architecture and Marine Engineering, of the National Technical University of Athens, is providing to the students and the researches the opportunity to extract real data and experiment with different efficiency models on the diesel engines which facilities can dispose.
The HIPPO-2 hybrid diesel-electric power plant consists of a internal combustion engine (ICE) in serial connection to an electric motor (EM) is presented in figure 3.1:

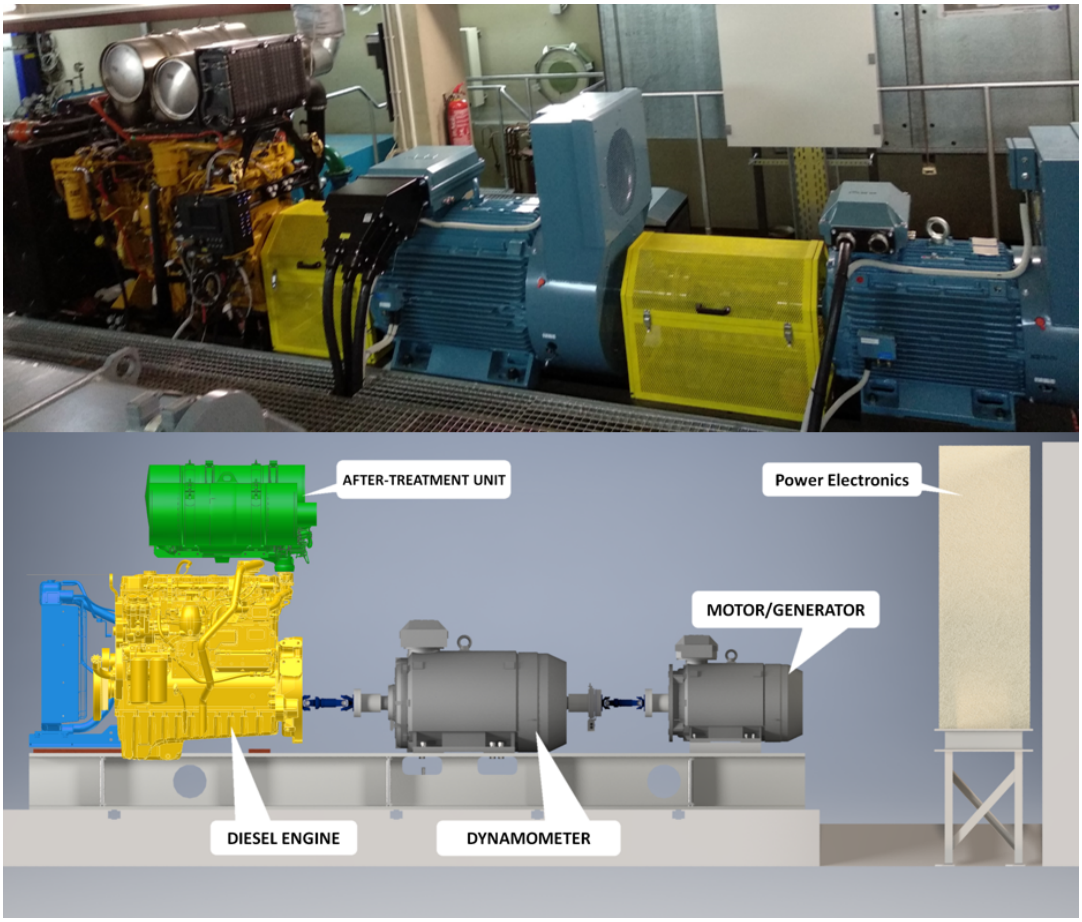


Figure 3.1: HIPPO-2 Engine

3.2.2 Data overview

The Data set used for the purposes of this research, provided by HIPPO-2 sensors, were derived from real time measurements of different engine's loading scenarios. These data were processed and prepared from laboratory personnel and researchers in order to become high quality and accurate input data in case of further analysis.
The range of engine's performance is entirely covered, starting from approximately 800 rpm up to 1300 rpm in a period of 35 minutes 450000 detailed data have been recorded. The variables extracted from the engine are described in table 3.2:

Variable	Mean	Min	Median	Max
NOx [ppm]	368.98	18.40	348.70	1226.90
Fuel Consumption [kg/h]	24.91	0.35	23.35	65.40
lambda [-]	2.76	1.15	1.67	134.63
Exhaust Gas Mass Flow [kg/h]	597.55	237.40	558.60	1318.40
MAP [kPa]	66.90	2.00	54.00	198.00
Torque Reference [%]	49.14	2.00	50.00	100.00
Rot. Speed [rpm]	1309.22	787.42	1206.69	1915.52
Engine Torque [Nm]	627.38	-228.32	633.34	1485.32
EGR Command [%]	22.69	0.00	28.52	100.00
Exhaust Gas Temperature [$°C$]	318.95	145.25	321.75	435.31

Table 3.2: Data overview

In the following figures 3.2, 3.3 the history of the above mentioned measured signals are presented:

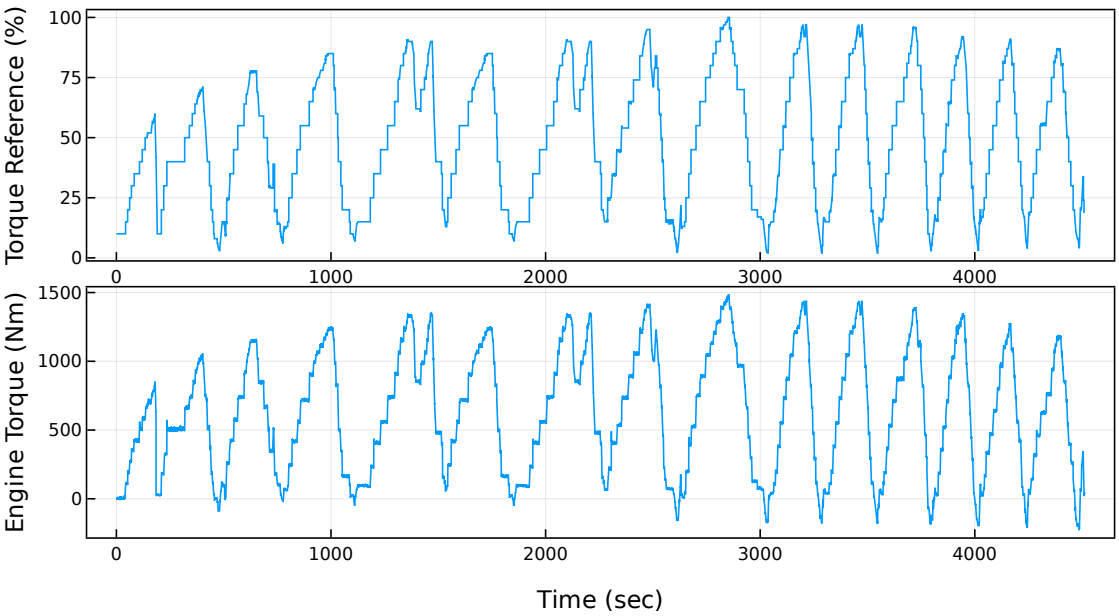
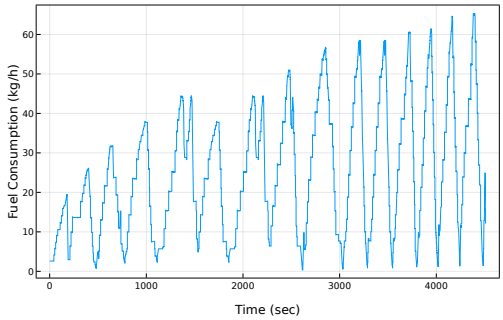
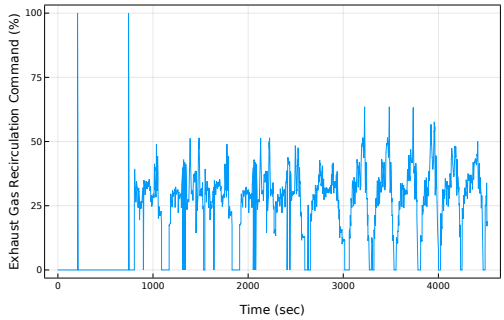


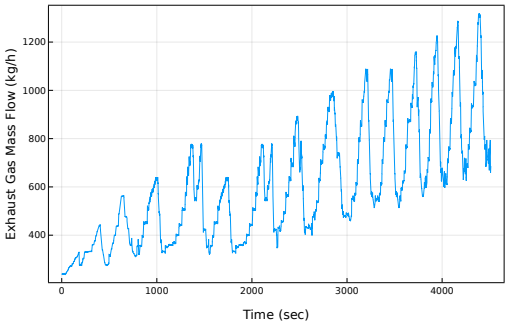
Figure 3.2: Reference Torque (%) and Engine Torque (Nm).



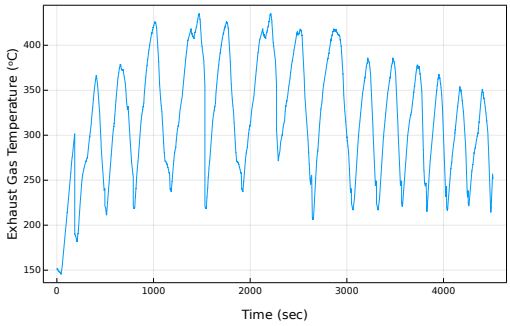
(a) Fuel Oil Consumption (kg/h).



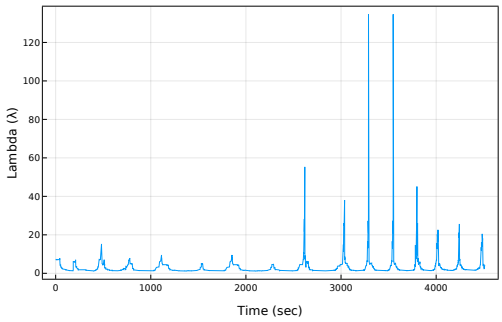
(b) Exhaust Recirculation Gas Command (%).



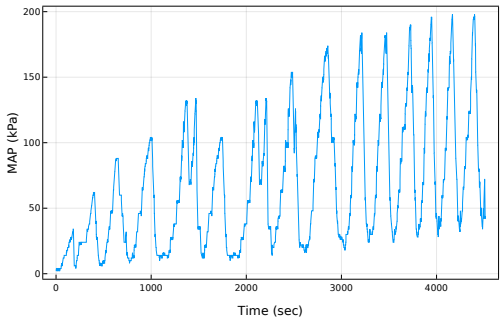
(c) Exhaust Gas Mass Flow (kg/h).



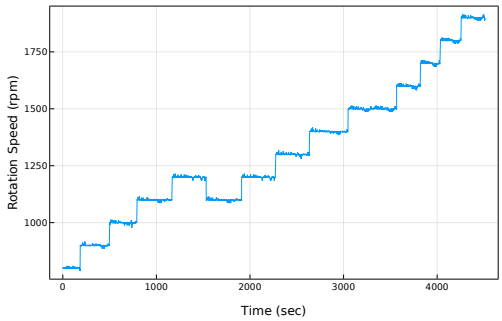
(d) Exhaust Gas Temperature (^oC).



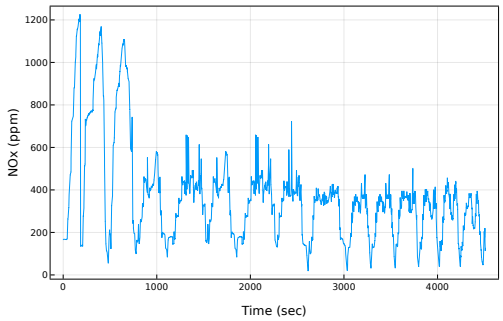
(e) Lambda.



(f) MAP (kPa).



(g) Rotational Speed (rpm).



(h) NOx emissions (ppm).

Figure 3.3: Dataset Variables in comparison with time.

In the following figures 3.4, 3.5 the value's frequencies and the Empirical Distribution (ECD) function are presented. The histogram plots are shown the occurrence frequency of different values in dataset for each feature while the Empirical distribution function is an estimate of the cumulative distribution function that generated the points in the sample.

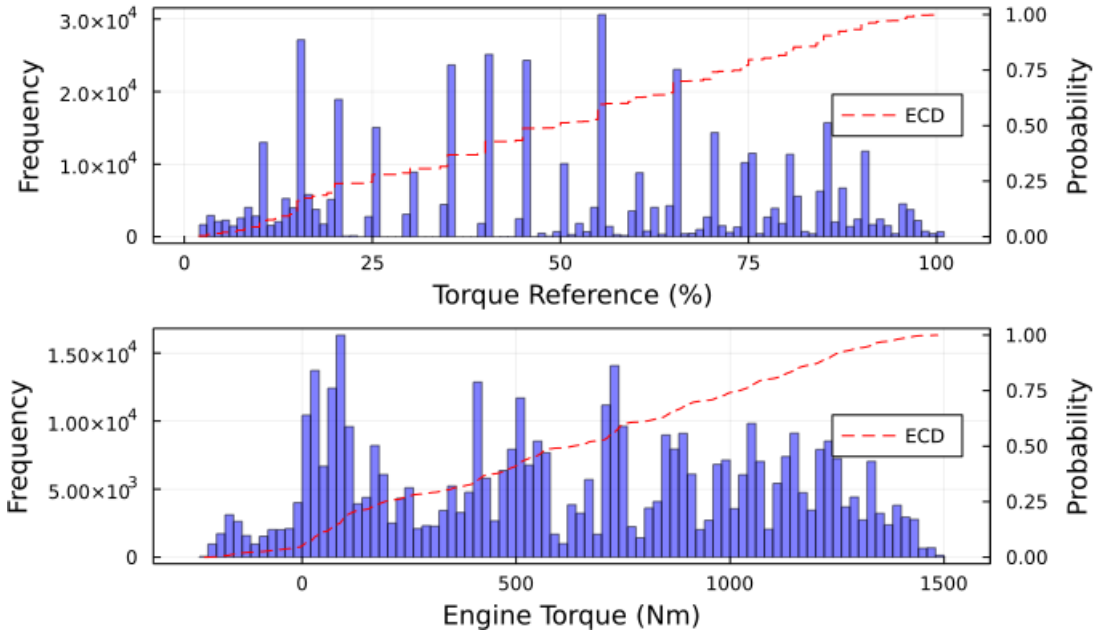


Figure 3.4: Reference Torque (%) and Engine Torque (Nm) frequencies and ECD.

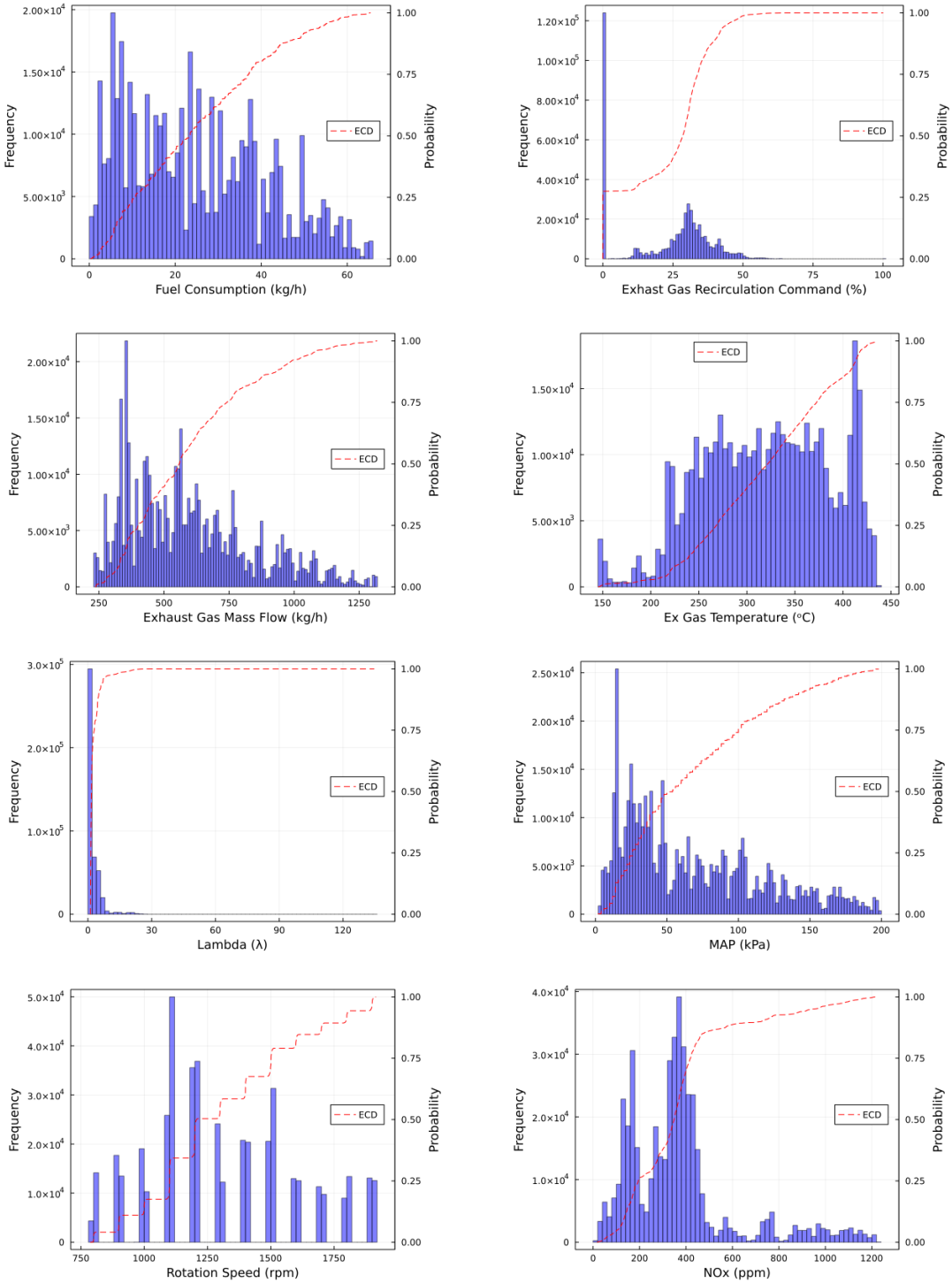


Figure 3.5: Frequencies and ECD of each variable.

3.2.3 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. For example, the large training data set, which covers the whole main engine envelope, is equivalent to 451228 samples of 0.01 seconds, i.e. 75 minutes of running, approximately. As it is obvious, it contains a large amount of samples, which will slow down the training of the neural network and increase its complexity. 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 3.6 demonstrates, even the resampled NOx data set contains more than enough points for training.[13]

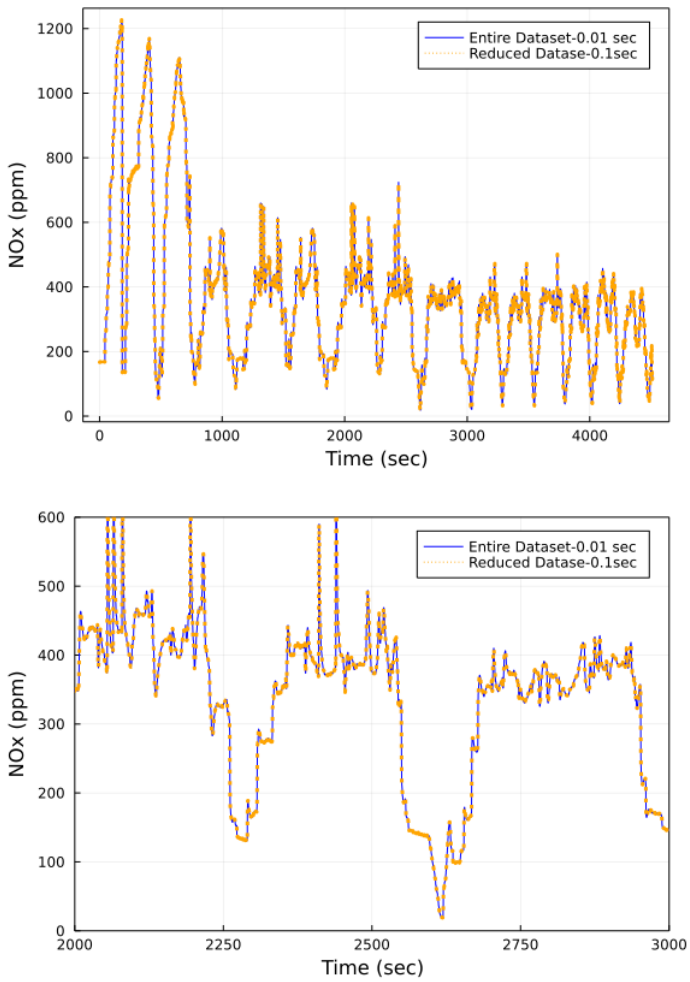


Figure 3.6: Initial (blue line) and Re-sampled (orange dots) NOx Dataset.

Chapter 4

Virtual sensors using LSTM neural network

Unlike the common sensors which widely used in previous years in now days a new technologies have been developed to inform engines operators about the different physical phenomena which occurred during the operation. A virtual sensor is nothing more than a software capable to replace the physical sensor, and produce almost the same results having significant reduced operation and maintenance cost. A virtual sensor learns how the different input variables effects the system and simulated them during the operation.

4.1 Input Data Modification

LSTM network requires the input data to be given in specific form called samples, each sample is composed of a input component and a output component. Input components could consist more than one input variables only on the condition that these variables would be consecutive time steps.[14]
For example, an LSTM model which receives NOx and lambda as input variables and predicts the Fuel Consumption with samples of three consecutive time steps will have the follow form:

	Input variables			Output variables
	(NOx,lambda)			(Fuel Consumption)
Time steps	1	2	3	4
Sample 1	$(\text{NOx}_t, lambda_t)$	$(\text{NOx}_{t+1}), lambda_{t+1})$	$(\text{NOx}_{t+2}, lambda_{t+2})$	Fuel Consumption$_{t+3}$
Sample 2	$(\text{NOx}_{t+1}, lambda_{t+1})$	$(\text{NOx}_{t+2}, lambda_{t+2})$	$(\text{NOx}_{t+3}, lambda_{t+3})$	Fuel Consumption$_{t+4}$

Table 4.1: Example of LSTM network's input-output data.

In training mode samples should be shuffled in order to achieve better training results and accuracy.

4.2 LSTM Model Configuration

The LSTM model could be developed in different ways, in this task different kind of models will be presented and based on results and time-costs, the best model will be chosen for the virtual sensors configuration.
In this configuration process, 12000 from 45000 data were used, from the available Dataset.

4.2.1 Sample Time Steps Configuration

At first a comparison between different number of Time steps, in the samples, will take place. A simple model of neural network composed of a LSTM network as a input layer, a Dense for a hidden layer and finally a Dense of output layer using 16 neurons. As a activation function in two Denses Relu function was chosen. As for the input and output data real scenario of virtual sensor was considered, and the variables of **Fuel Consumption, lambda, EGR Command, MAP** and **Torque reference** were taken as input variables in order to predict the output variable of **NOx** emissions.
The model described above is presented in the following table:

	Model description		
Model Chain	LSTM(5,16)	Dense(16,16,relu)	Dense(16,1)
Parameteres	1440	272	17

The samples configurations which were tested and compared in this task consist of three, six, nine and thirty-two consecutive time steps as shown below:

Model testing samples		
Sample	Input	Output
a	$(\text{Input}_t, Input_{t+1}, Input_{t+2}$	Output_{t+3}
b	$(\text{Input}_t, Input_{t+1}, ...Input_{t+5}$	Output_{t+6}
c	$(\text{Input}_t, Input_{t+1}, ...Input_{t+8}$	Output_{t+9}
d	$(\text{Input}_t, Input_{t+1}, ...Input_{t+31}$	Output_{t+32}

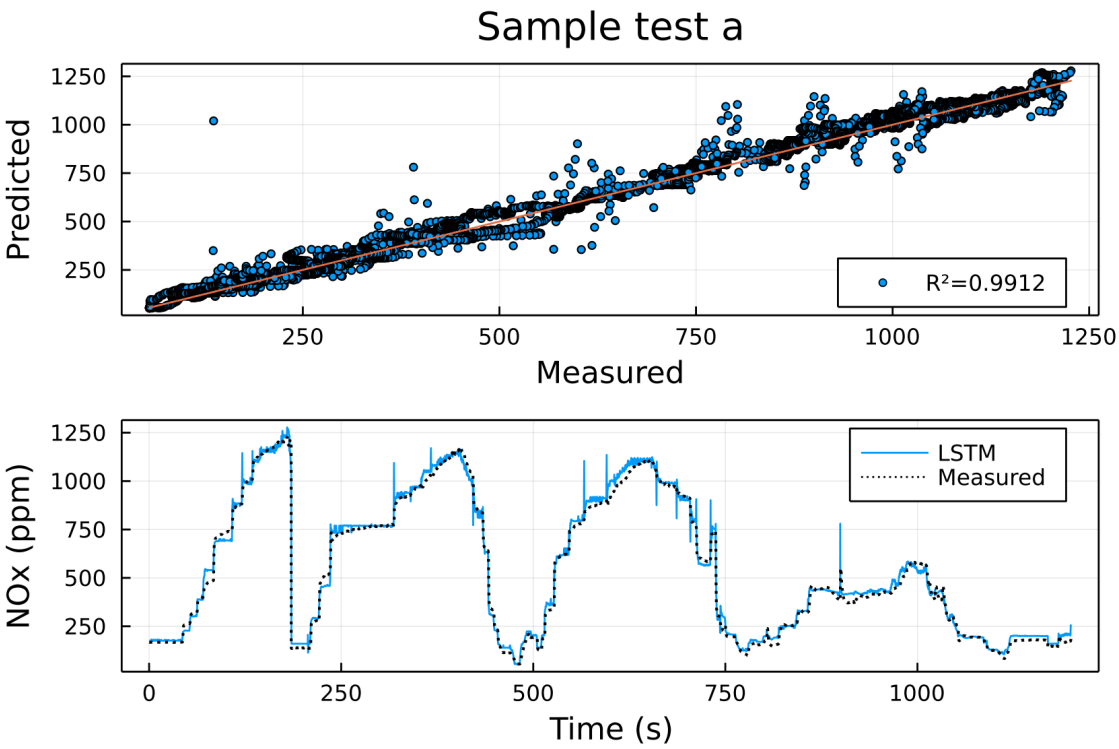
The model trained with ADAM optimizer, of learning rate $\eta = 0.01$, 10 epoch and batch size 32, data were splitted at training & testing data by the rate 70%-30% respectively, organized in the different types of samples which mentioned before and shuffled, the results of different types of samples presenting at table 4.2:

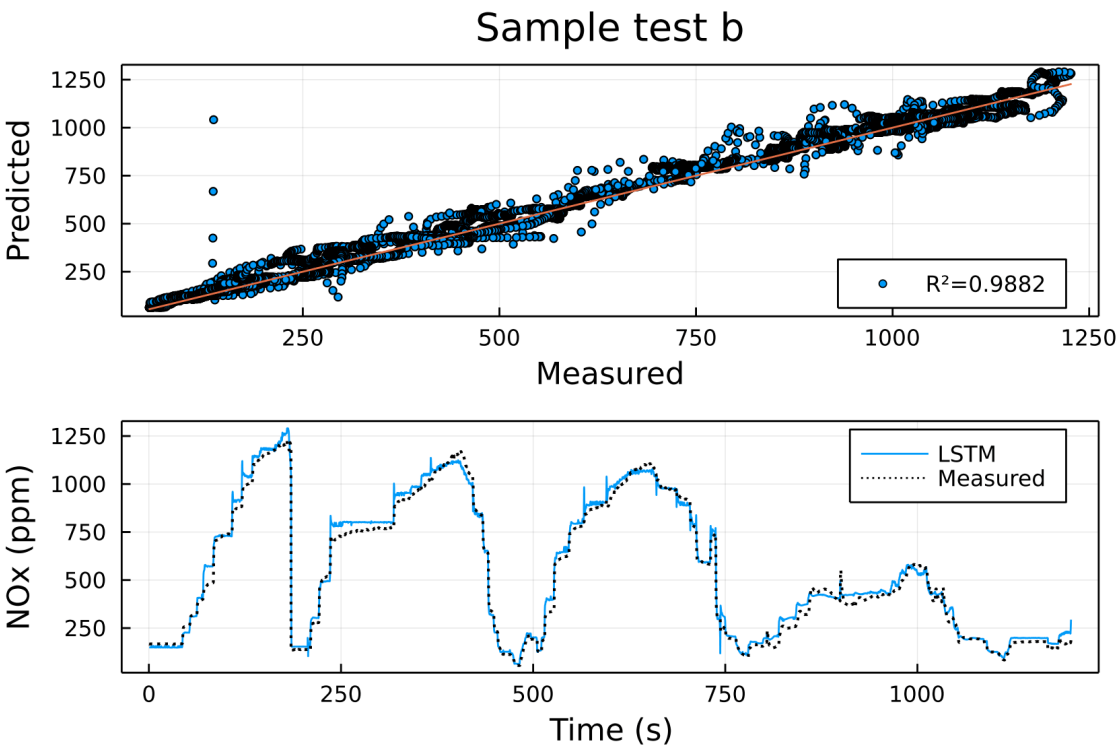
Model testing results					
Samples	Time steps number	Model Accuracy (Training Comp/Test Comp)	Mean Absolute Error (Train data/Test data)	Allocations (Number/Memory)	Time cost (sec)
a	3	99.11% / 99.17%	1.984% / 1.993%	7.45 M /1.529 GiB	4.66
b	6	98.80% / 98.86%	2.463% / 2.470%	12.31 M /3.041 GiB	6.90
c	9	98.52% / 98.51%	2.680% / 2.733%	17.16M /4.609 GiB	8.48
d	32	99.06% / 98.90%	2.218% / 2.258%	54.53M /18.446 GiB	31.52

Table 4.2: Model testing Results for time steps number

Its easy to conclude that the best results taking into consideration the time cost lies between sample a (3 time steps) and sample b (6 time steps). Due to the lower time cost and the accuracy, the sample a will be chosen for further analysis.
The figures of NOx, predicted-real and in comparison with time, are presented for each

Sample type, R^2 which is shown in scatter plots represents the r2 score of all the input data that model receives (both Training and Testing data):





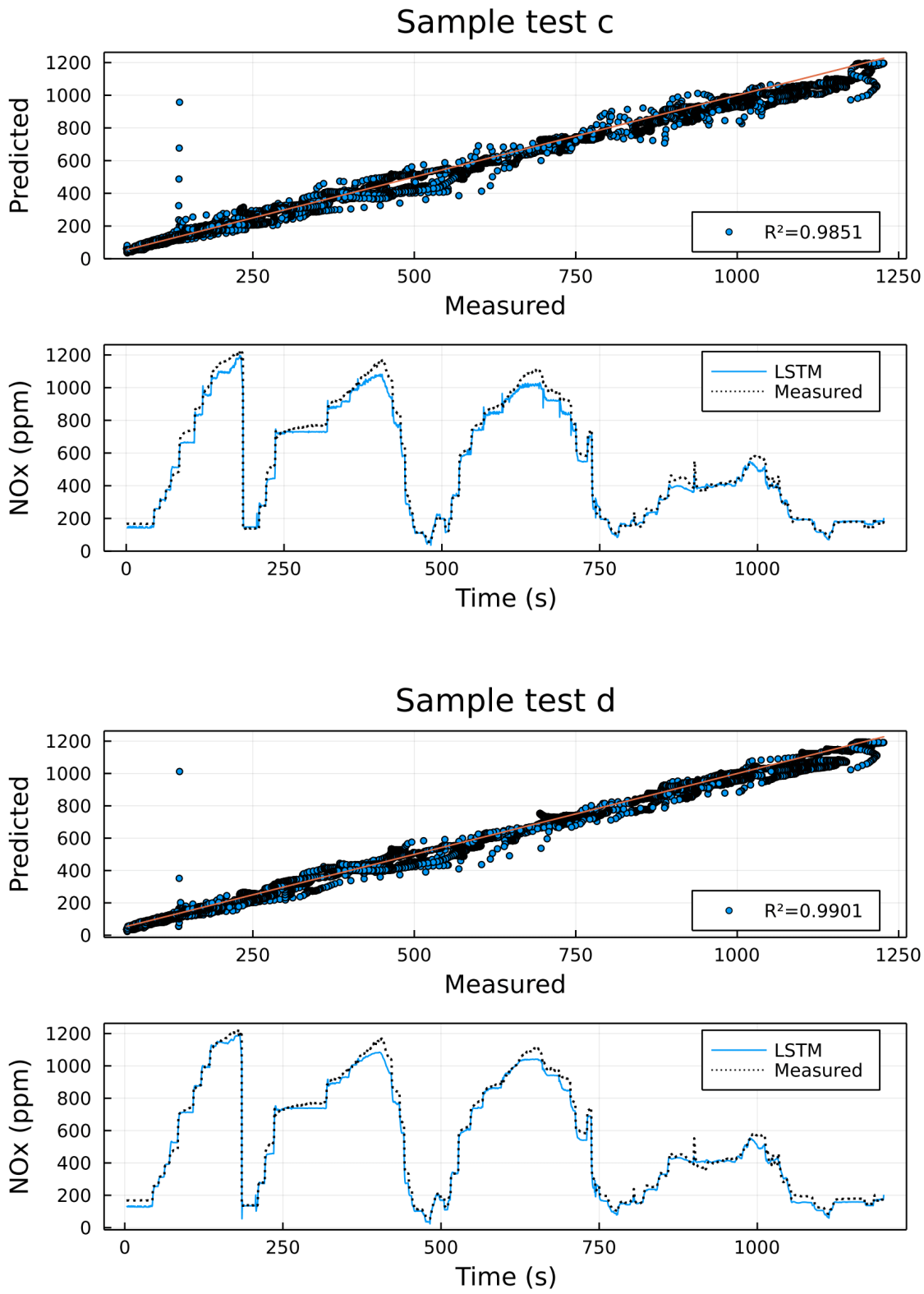


Figure 4.1: Sample tests results.

Finally the number of neurons of model described at section 4.2.1 is to be determined, using the time step chosen before (3 steps), three versions of the same model has been tested, each one with different number of neurons, the different versions and the results are describing in table 4.3:

Model testing results						
Version	Neurons	Trainable Parameters	Model Accuracy (Training Comp/Test Comp	Mean Absolute Error (Train data/Test data)	Allocations (Number/Memory)	Time cost (sec)
a	16	1729	99.11% / 99.17%	1.984% / 1.993%	7.45M /1.529 GiB	4.66
b	32	6017	99.22% / 99.15%	1.856% / 1.798%	7.45M /2.599 GiB	5.40
c	64	22273	99.07% / 98.98%	2.099% / 2.106%	7.50M /5.051 GiB	12.64

Table 4.3: Model testing Results for Neurons

As it seems at the above table, despite the fact that the trainable parameters in **version a** are almost four times more than model **version b**, the time cost has slightly increased.So the model which is going to be developed will be similar to version b.
In conclusion the lstm model which has been configured consist of a three time steps model, with 32 neurons and 3 time steps.

4.3 Model Training

4.3.1 Output Variable Selection

The output Variables selected taking into consideration the increased interest of Marine Industry about the environmental impact of vessels operation, the great effort which taking place in order to reduce Fuel Consumption and the significant variables which could effect Engine's operation. Finally the output variables which are going to been predicted by the models are:

- **Fuel Consumption**
- **MAP**
- **Lambda**
- **Engine Torque**
- **NOx**

4.3.2 Input Variable Selection

The selection of model inputs is considered as one of the most important steps in black-box model developing. The different relationships and dependencies between inputs and outputs have a significant impact at model's performance. Direct relationships between inputs outputs intasosbility of the network. In case of engine emissions modeling, available measurements are very limited. In order to deal with the lack of available measurements, the different inputs to emission models are chosen carefully based on physical insight into pollutant formation mechanisms.

During the operation of the engine, the operating condition is determined by the engine speed and fuel injection quantity, and this fact makes these two variables highly important for the engine emissions. On the other hand EGR plays a dominant part on the NOx and smoke formation, in particular increasing EGR rate leads to higher heat capacity of the in cylinder gas causing lower flame temperatures and reduce the NOx formation. However, excessive amount of EGR in the cylinder could lead to incomplete combustion and increase smoke formation. Another parameter which is highly significant for engine emissions is the air to fuel ratio (AFR) and the lambda parameter.[15]

In order to determine the most efficient Inputs for each model, cross-correlation between the candidate input variables and the chosen output ones has to be studied. For this purpose the *SelectKBest* function of ScikitLearn.jl library was used to visualize the different correlationships between inputs-outputs and as such finally to select the combination which maximises the model's accuracy and reduces its training time cost. The method used from SelectKBest function was f_regression.

F_regression executes univariate linear regression tests returning F-statistic and p-values, and it is a quick linear model for testing the effect of a single regressor, sequentially for many regressors. The procedure follows two steps:

1. The cross correlation between each regressor and the target is computed. For every feature $X[:,i]$ it computes the correlation with y:

$$\rho = \frac{(X[:,i] - mean(X[:,i]) * (y - mean(y)))}{std(X[:,i]) * std(y)}$$

2. Then it computes the F-static:

$$F_i = \frac{\rho_i^2}{1 - \rho_i^2} * (n - 2)$$

   where $n = len(y)$, the number of samples (there is a slight difference if parameter center is False, then it multiplies with $n - 1$). These F-values are then returned, together with the associated p-values. So the result is a tuple (F-values, p-values). Then SelectKBest takes the first component of this tuple (these will be the scores), sorts it, and picks the first k features of X with the highest scores. As k has been set all the available variables of Data set SelectKBest(f_regression, k="all").

It is worth to emphasize that SelectBest could discriminate the correlations between the input features. In case where the information of one feature has already given to model through one of the others, it is expected to mark lower score than different less relevant inputs. For example EGR and Lambda variables, are highly connected to each other, that's why in the NOx model EGR is expected to mark lower score that other less important for NOx formation variables.
Also in this thesis, selected model's inputs variables are to be easily observed by physical sensors and for this reason Engine Torque has been replaced by Engine Torque reference.[16]
Scores for each different output variable are presented below:

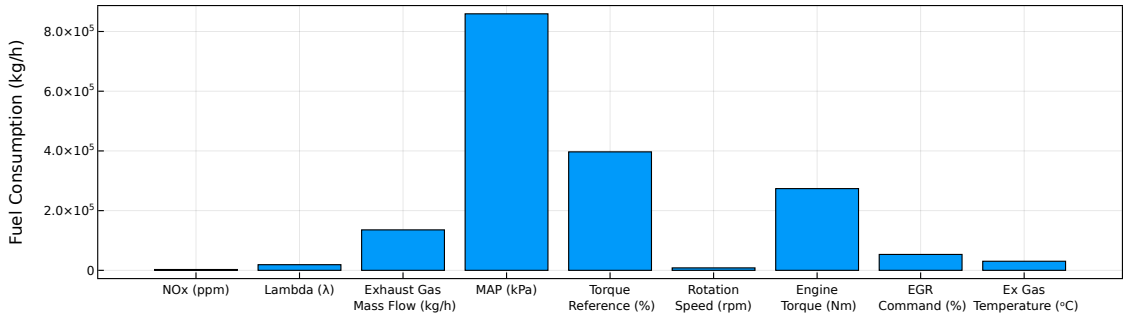


Figure 4.2: Fuel Consumption Model data correlations.

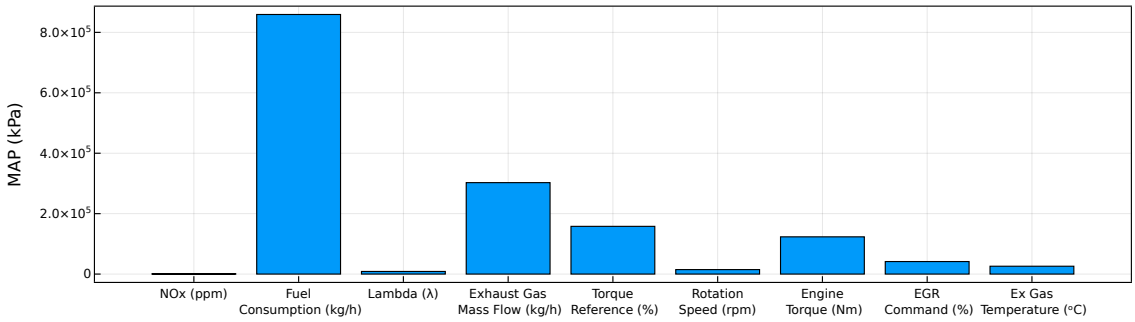


Figure 4.3: MAP Model data correlations.

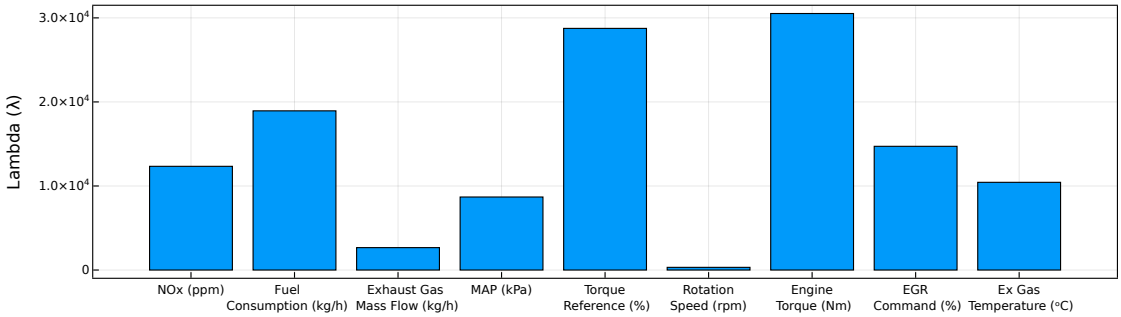


Figure 4.4: Lambda Model data correlations.

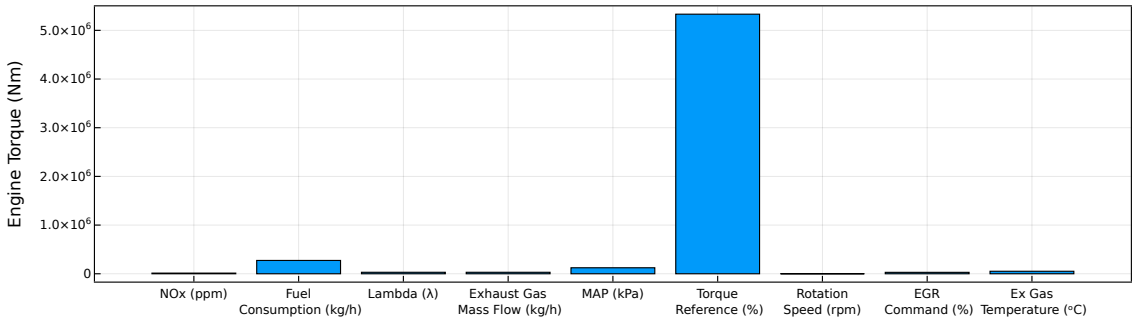


Figure 4.5: Engine Torque Model data correlations.

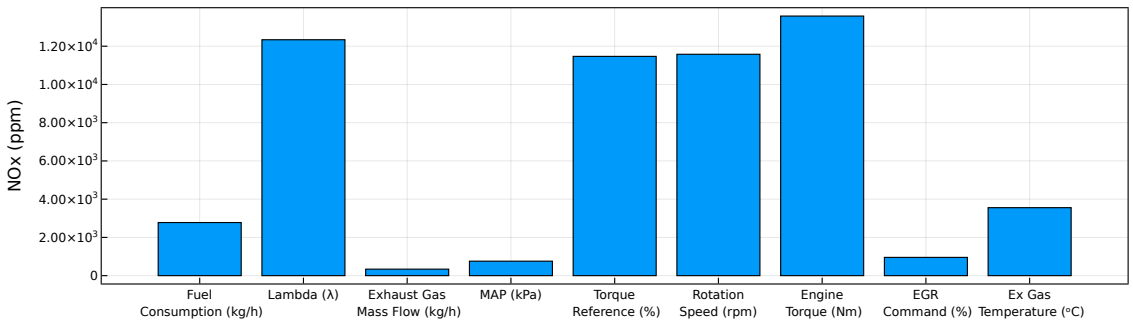


Figure 4.6: NOx Model data correlations.

Taking into consideration the above Score results of SelectKBest function, the four different models have been developed as shown below.

- **Fuel Consumption:** According to the results of SelectKBest function as shown in figure 4.2 input variables of Fuel Consumption Model are selected to be: Exhaust Gas Mass Flow, MAP, Torque Reference.

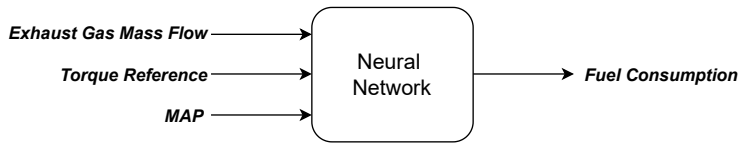


Figure 4.7: Fuel Consumption Model input.

- **MAP:** Similarly, the figure 4.3 input variables of MAP Model are selected to be: Fuel Consumption, Exhaust Gas Mass Flow, Torque Reference.

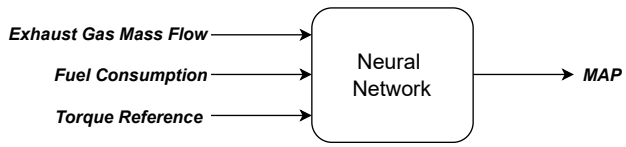


Figure 4.8: MAP Model input.

- **Lambda (λ):** According to the figure 4.4 input variables of Lambda Model are selected to be: Fuel Consumption, Torque reference, EGR Command, Exhaust Gas Temperature.

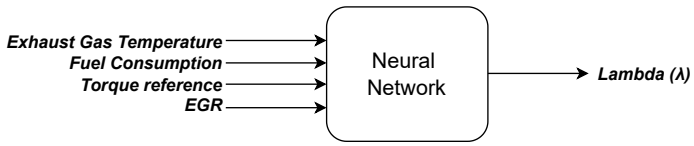


Figure 4.9: Lambda Model input.

- **Engine Torque:** According to figure 4.5 input variables of Engine Torque Model are selected to be: Fuel Consumption, Torque reference, MAP.

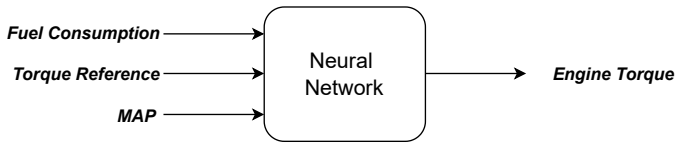


Figure 4.10: Engine Torque Model input.

- **NOx:** According to the results of SelectKBest function as shown in figure 4.6 and also the relation between inputs-outputs described in section 4.3.2, input variables of NOx Model are chosen to be: FuelConsumption, Lambda, Torque Reference, Rotation Speed, EGR Command.

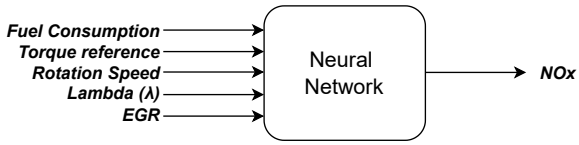


Figure 4.11: NOx Model input.

4.3.3 Training and Testing Dataset

The engine's Dataset was contributed at *.csv* file and inserted in *julia* environment through a structure called DataFrames. Afterwards Data were formed in samples of 3 continues time-steps as configured in subsection 4.2.1 and the using the command *shuffleobs* and *splitobs* data shuffled and separated in rate of 70% training- 30% testing data. This step is essential in order to train the model not only in a percentage of the range but in all of data set and also to review the performance of LSTM model on the given data of HIPPO-2 Diesel Engine. The virtual sensor which going to be developed will be composed of the same neural network architecture of **32 neurons** and **3 time steps** per sample (as configured in 4.2) although the inputs and the outputs variables will be different in each case. As before the Number of epochs will be 10 and the batch sized 32.

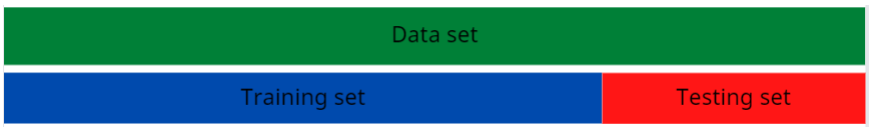


Figure 4.12: Training and Testing data set.

4.3.4 Data Normalization

When features in machine learning have different range Data Normalization is considered to be necessary step. Otherwise non-normalized input variables could result in a slow or unstable learning process. If a Feature in the Dataset is big in scale compared to others then this big scaled feature becomes dominating and as a result of that, Predictions of the Neural Network will not be Accurate. Normalization improves models accuracy dramatically since it gives equal weights to each feature so that no single variable steers model performance in one direction just because they are bigger numbers. Backpropagation of Neural Networks involves the Dot Product of Weights with Input Features. Consequently, Model converges slowly, if the Inputs are not Normalized. In this thesis the package *Stats-Base* was used in julia environment and in particularly the *fit()* command, and through the Unitrange Transform each input-output variable scaled within the range of 0 and 1 according the following equation:

$$\mathbf{X_{normalized}} = \frac{\mathbf{x - min}}{\mathbf{max - min}} \tag{4.3.1}$$

4.3.5 Activation Function

Activation function is the function which decides whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. For a given node the inputs are multiplied by the weights and then summed together in order to transformed via an activation function and finally define the specif output. Linear Activation function considered as the simplest activation function since no transform is applied at all. A neural network composed of only linear activation function it is easy to train but it is not capable to learn and perform complex tasks. On the other hand nonlinear activation function are widely used and preferred as they allow the network to learn more complex patterns in the data. In this thesis the output values calculated using two types of activation function, according to the output structure and the trial-and-error attempts.

ReLU (Rectified Linear Unit)

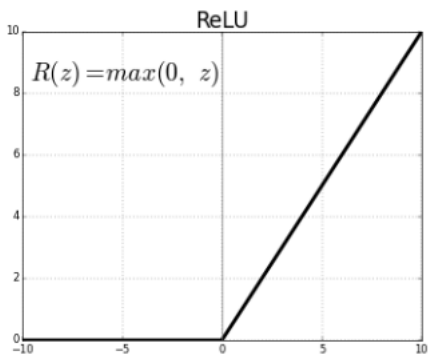


Figure 4.13: ReLU Function

The rectified linear activation function has rapidly become the default activation function when developing most types of neural networks. ReLU provides computational simplicity, there is no need for computing the exponential function in activations, so the time cost is significant reduced in compered with sigmoid or tanh function. Also an important benefit of ReLU activation function is that it is capable of outputting true zero value and not

an approximation of zero output like sigmoid and tanh function. This means that negative inputs can output true zero values allowing the activation of hidden layers in neural networks to contain one or more true zero values, accelerating the learning process and simplify the model. Finally rectified linear units are based on the principle that models are easier to optimize if their behavior is closer to linear. So due to this linearity, gradients flow well on the active paths of neurons (there is no gradient vanishing effect due to activation non-linearities of sigmoid or tanh units).

Sigmoid function

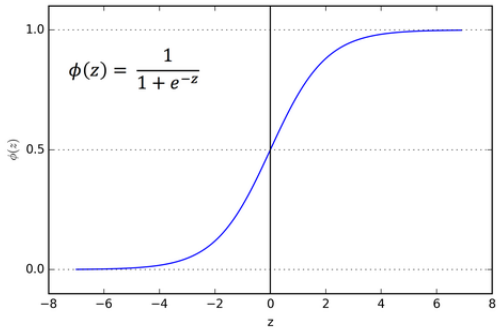


Figure 4.14: Sigmoid Function.

The Sigmoid function is the most frequently widely used activation function in the beginning of deep learning. It is a smoothing function that is easy to derive and implement. Sigmoid Function output bound between 0 and 1 normalizing the output of each neuron. Besides that it provides smooth gradient which contributes to preventing "jumps" in output value. Also, around z=0 where z is neither too large or too small, 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.[17]

4.3.6 Optimizer

Training a very large deep neural network could be significant slow procedure. A key factor to speed boost model training comes from using a faster optimizer than the regular Gradient Descent optimizer. Optimizers are algorithms which finds the value of the parameters (weights) that minimize the error when mapping inputs to outputs. Learning rate is a optimizer parameter that provides the model a scale of how much model weights should be updated. In this task the AdaGrad, RMSprop and Adam optimizers will be presented, as implemented in the designed networks.[18]

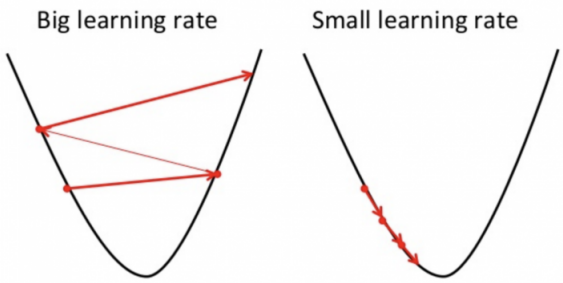


Figure 4.15: The effect of the learning rate in model's behavior.

4.3.7 AdaGrad

The **AdaGrad** algorithm individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the sum of all the historical squared values of the gradient. The parameters with the largest partial derivative of the loss have a correspondingly rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. The net effect is greater progress in the more gently sloped directions of parameter space.

$$\upsilon_t^w = \upsilon_{t-1}^w + (\nabla w_t)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\upsilon_t^w + \epsilon}} * \nabla w_t$$

$$\upsilon_t^b = \upsilon_{t-1}^b + (\nabla b_t)^2$$
$$b_{t+1} = b_t - \frac{\eta}{\sqrt{\upsilon_t^b + \epsilon}} * \nabla b_t$$

4.3.8 RMSProp

The **RMSProp** algorithm modifies AdaGrad to perform better in the nonconvex setting by changing the gradient accumulation into an exponentially weighted moving average. AdaGrad is designed to converge rapidly when applied to a convex function. When applied to a nonconvex function to train a neural network, the learning trajectory may pass through many different structures and eventuallyarrive at a region that is a locally convex bowl. AdaGrad shrinks the learning rate according to the entire history of the squared gradient and may have made the learning rate too small before arriving at such a convex structure. RMSProp uses an exponentially decaying average to discard history from the extreme past so that it can converge rapidly after finding a convex bowl, as if it were an

instance of the AdaGrad algorithm initialized within that bowl.

$$\begin{aligned} \upsilon_t^w &= \beta * \upsilon_{t-1}^w + (1-\beta)(\nabla w_t)^2 \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{\upsilon_t^w + \epsilon}} * \nabla w_t \end{aligned}$$

$$\begin{aligned} \upsilon_t^b &= \beta * \upsilon_{t-1}^b + (1-\beta)(\nabla b_t)^2 \\ b_{t+1} &= b_t - \frac{\eta}{\sqrt{\upsilon_t^b + \epsilon}} * \nabla b_t \end{aligned}$$

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 2.4.17. Sometimes the value of υ^w could be really close to 0. Then, the value of weight 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.

4.3.9 Adam

Adam is yet another adaptive learning rate optimization algorithm. The name "Adam" derives from the phrase "adaptive moments." In the context of the earlier algorithms, it is perhaps best seen as a variant on the combination of RMSProp and momentum with a few important distinctions. First, in Adam, momentum is incorporated directly as an estimate of the first-order moment (with exponential weighting) of the gradient. The most straightforward way to add momentum to RMSProp is to apply momentum to the rescaled gradients. The use of momentum in combination with rescaling does not have a clear theoretical motivation. Second, Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin. RMSProp also incorporates an estimate of the (uncentered) second-order moment; however, it lacks the correction factor. Thus, unlike in Adam, the RMSProp second-order moment estimate may have high bias early in training. Adam is generally regarded as being fairly robust to the choice of hyperparameters, though the learning rate sometimes needs to be changed from the suggested default.

$$\begin{aligned} m_t &= \beta_1 * m_{t-1} + (1-\beta_1) * \nabla w_t \\ \upsilon_t &= \beta_2 * \upsilon_{t-1} + (1-\beta_2)(\nabla w_t)^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \qquad \hat{\upsilon}_t = \frac{\upsilon_t}{1-\beta_2^t} \\ w_{t+1} &= w_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t + \epsilon}} * \hat{m}_t \end{aligned}$$

A similar set of equations are used for b_t. In the above equations, t represents the iteration number, as in 2.4.16, starting at 1. The decay hyperparameter β_1 is initialized to 0.9, while the scaling decay hyperparameter β_2 is initialized to 0.999. As earlier, the smoothing term ϵ is usually initialized to a tiny number such as 10^{-8}.Since Adam is an adaptive learning rate algorithm, it requires less tuning of the learning rate hyperparameter, with 0.001 as a default value.[19]

4.3.10 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 $\hat{Y}$ being the predicted values, then the within-sample MSE of the predictor is computed as

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.3.2}$$

Mean Absolute Error: It measures the average of the absolute errors between the estimated 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 $\hat{Y}$ being the predicted values, then the within-sample MAE of the predictor is computed as:

$$MAE = \frac{\sum_{i=1}^{n}\left|(Y_i - \hat{Y}_i)\right|}{n} \tag{4.3.3}$$

Chapter 5

Training & Testing Results

This section contains both training and test results of the LSTM models designed to predict the following values:

- **Fuel Consumption**
- **MAP**
- **Lambda**
- **Engine Torque**
- **NOx**

In this task the model which configured and analyzed before will be trained and tested with all 45000 data after split them into a rate 70% training data-30% testing data, in order to review the performance of lstm model on the given data of HIPPO-2 Diesel Engine. The virtual sensor which going to be developed will be composed of the same neural network architecture of 32 neurons and 3 time steps per sample although the inputs and the outputs will be different in each case. As before the Number of epochs will be 30 and the batch sized 32.

5.1 Procedure Flowchart

Before presenting the predictions results of the models it could be useful to include a schematic representation of the modeling process. One common tool to visualize the procedure which took place in this thesis is the Flowchart diagram presenting below and summarize the procedure described before.

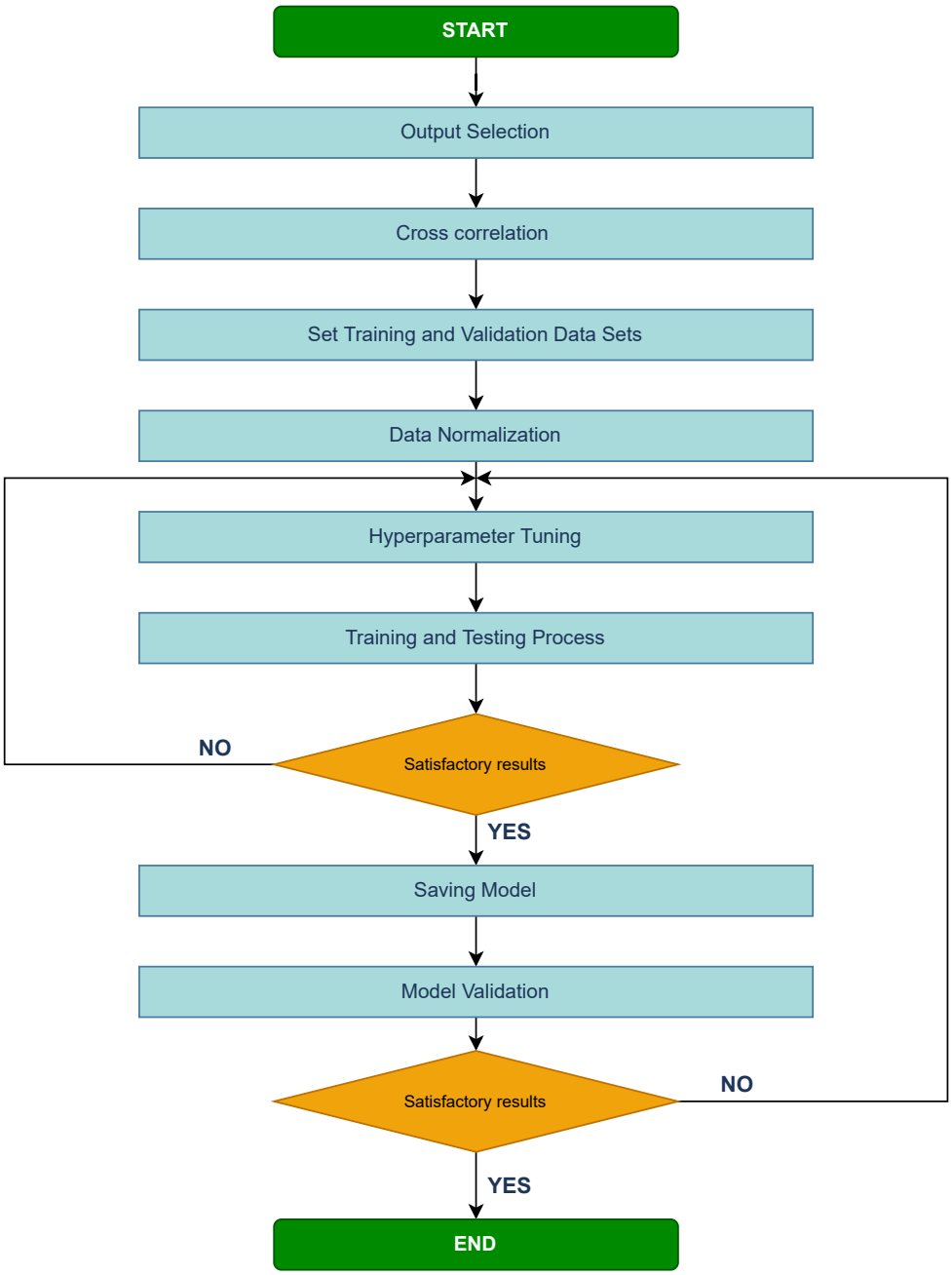


Figure 5.1: Flowchart of the Neural Network Modeling process, applied in this thesis.

5.2 Fuel Consumption Model

5.2.1 Training Results

After the Data preprocessing which carried out in section 3.2.3, the Exploratory Data Analysis (EDA) will be applied on the available Dataset using tools and methods for visualization and further decoding. One of the most common and effective tools are PairGrid function of Seaborn library which visualizes pairwise relationships between the multiple variables.[20] 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 of there) between two variables. The Figure 5.2 presenting below, shows the multiple pairwise bivariate distributions in the data set used for Fuel Consumption prediction.

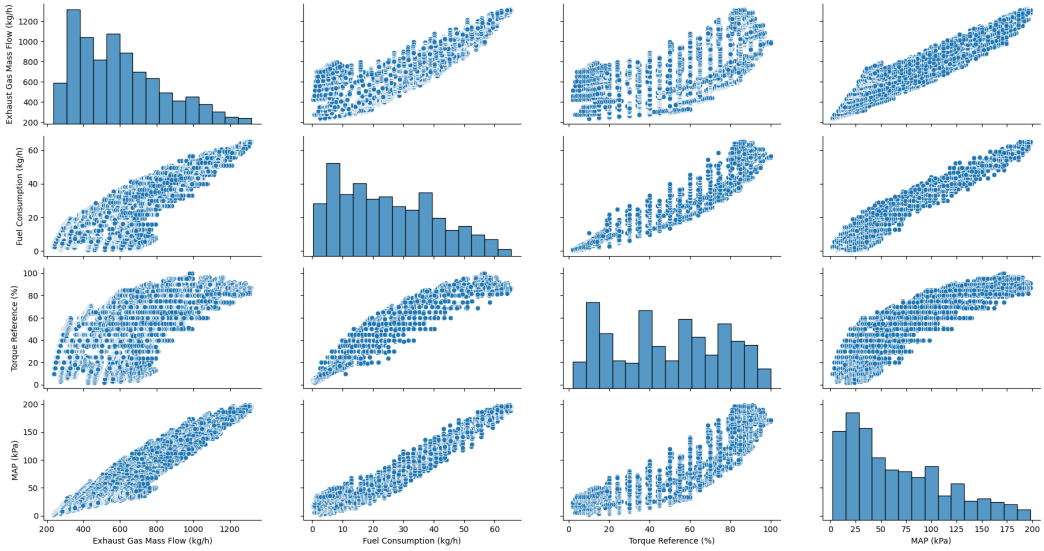


Figure 5.2: Pairwise Distributions.

In figure 5.2 reveals the linear connection between the different inputs and the Fuel Consumption, this factor makes the input variables easy to used for a neural network because there are not complex patterns to adapt.

Another 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. This function was used to visualize the pairwise correlation of variables in the data set. Correlation examines and quantities the relationship between two variables, or sets of data. A typical method of measuring correlation except the one used for chose the input variables at section 4.3.2 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.

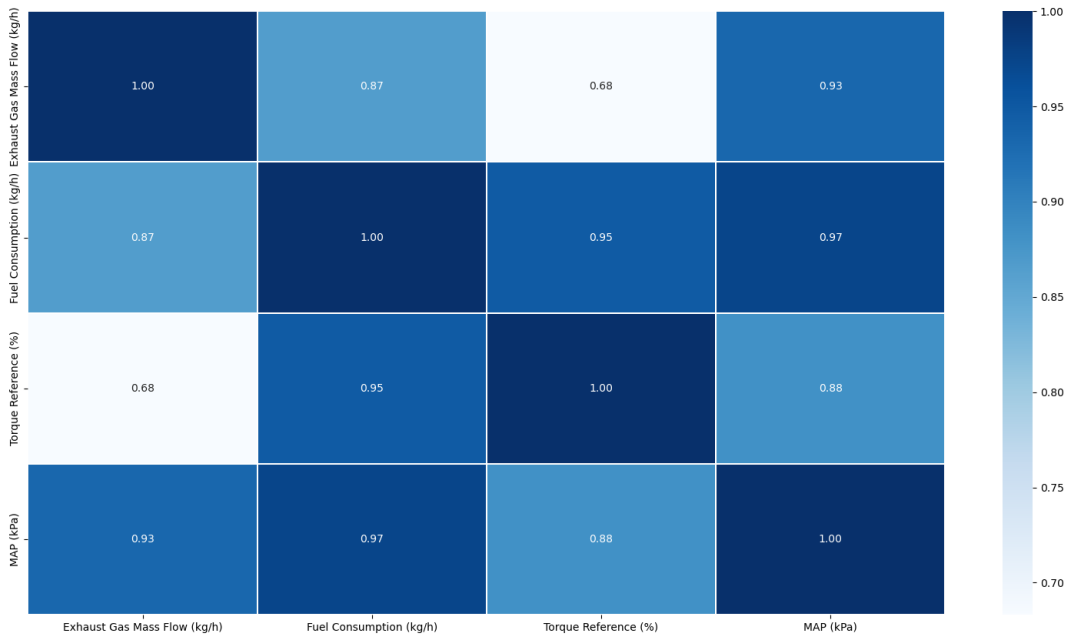


Figure 5.3: Heatmap of Pearson Correlation Coefficient.

In figure 5.3 the significant correlation between inputs variables and Fuel Consumption are revealed, while confirms the SelectBest function results about inputs correlations.

Fuel Consumption model consist of one LSTM input layer using 3 time steps in each sample, one hidden, and one output layer, connected to each other with 32 neurons using ReLU as activation function as configured in section 4.2.

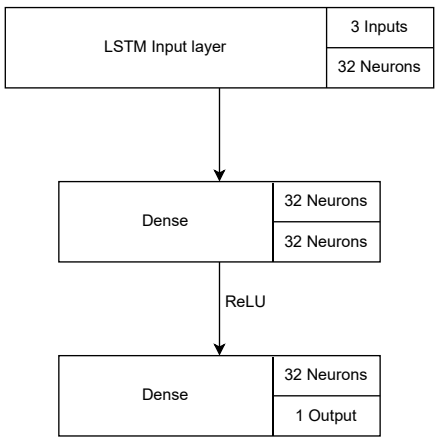


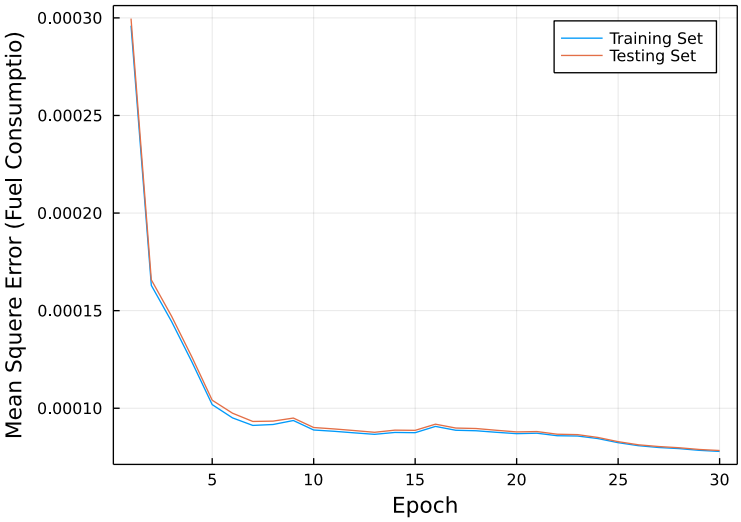
Figure 5.4: Model Chain.

Model is capable to accept inputs of any dimension, that means that the specific model could accept 3 vectors with unknown length. The Fuel Consumption model proceeds with input data of length 45123 (after resampling at section 3.2.3) and is trained with the 70% of them while the rest 30% of them is used for testing the model accuracy and error.

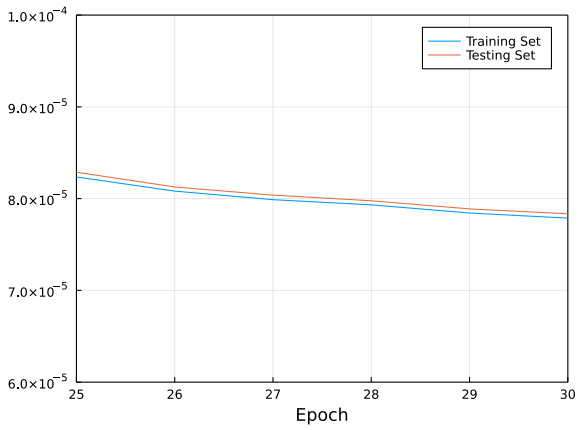
Inputs	Fuel Consumption (kg/h) Torque reference (%) Exhaust gas Mass Flow (kg/h)
Hidden Layers	1
Trainable Parameters	5761 (4672-1056-33)
Optimizer	ADAM (0.001)
Sample Time Step	3

Table 5.1: Fuel Consumption Model Characteristics synopsis.

During the training procedure, 30 epochs were used with ADAM optimizer of learning rate $\eta = 0.001$. In order to visualize the training performance the MSE was saved during both training and testing process in each epoch.In figure 5.5 model's loss function (Mean Squared Error) is presented.



(a) MSE during all epochs.



(b) Zoomed region.

Figure 5.5: Mean Squared Error during Training.

As it can be seen in the above figures, the error from the beginning could be considered as very small. Error was reduced drastically in the first 10 epochs and after this point the error adopts a almost linear distribution while training and testing error is almost

identical during all 30 epochs avoiding overfitting and accomplishing a case of good fit. Model results and accuracy could be visualized in scatter figures, which shown the linearity or the lack of it, between the measured and the predicted values of the training data. The R^2 score, calculated for all given data (training and testing), is also concluded in the figure while the Table 5.2 presents the detailed results of training process for Fuel Consumption Model:

Fuel Consumption Model Results						
Neurons	Input/Output Variables	Trainable Parameters	Model Accuracy (R^2) (Training Comp /Test Comp)	Mean Absolute Error (Train data /Test data)	Allocations (Number/Memory)	Time Cost (sec)
32	3/1	5761	99.87% / 99.87%	0.589% / 0.587%	84.02M /31.158 GiB	52.07

Table 5.2: Summarize Table of Model Results.

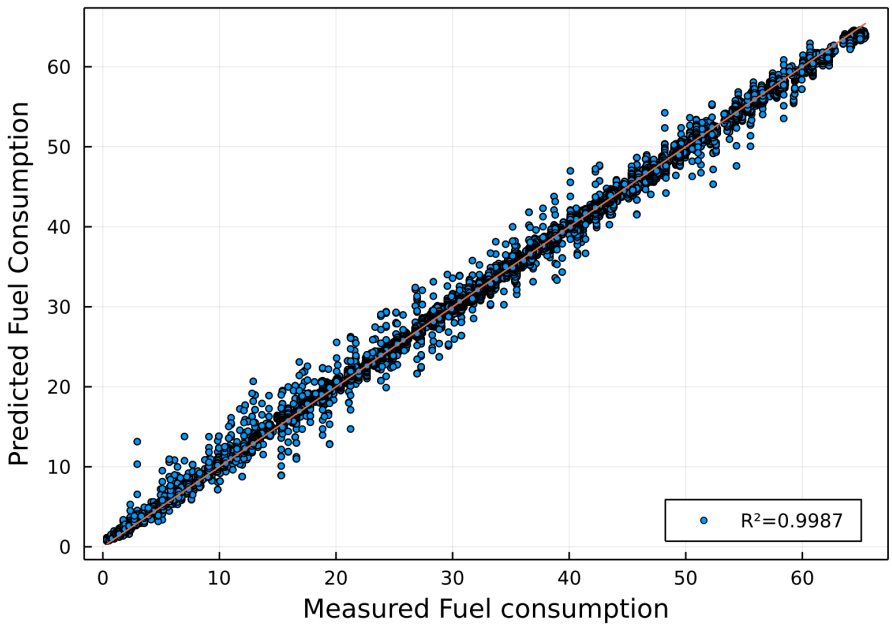
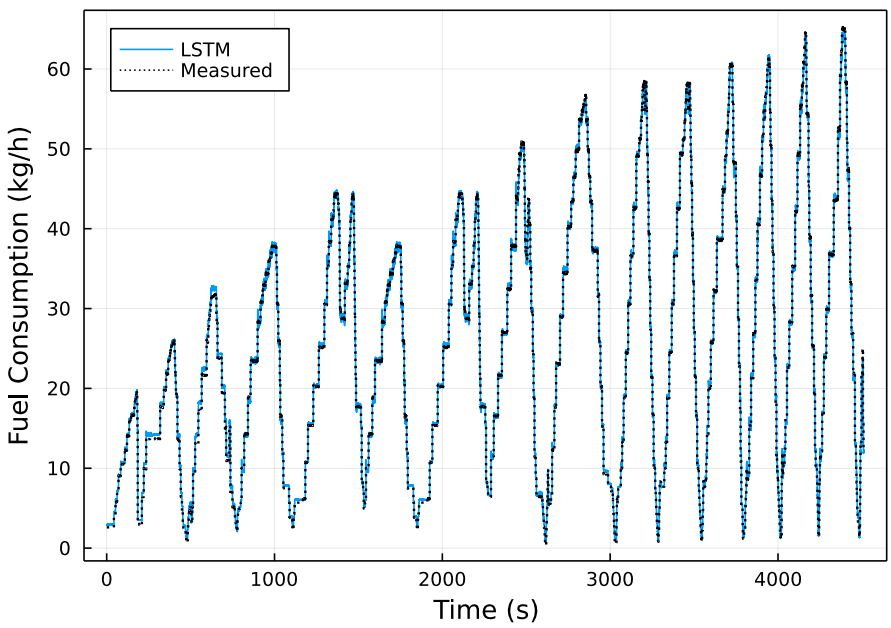
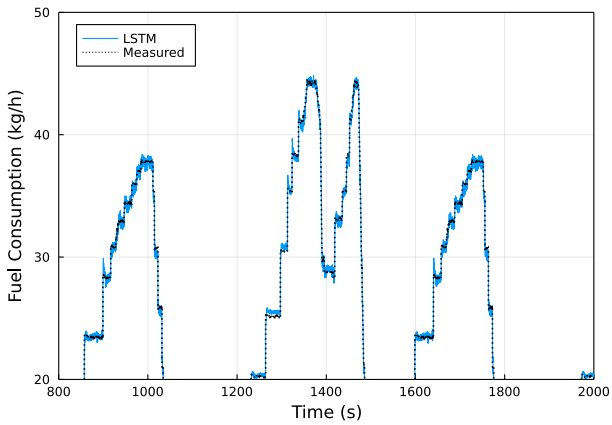


Figure 5.6: Scatter plot predicted-measured values.

In figure 5.6 the high accuracy of the model is presented while these two variables are linear distributed as the red line $y = x$ indicates. Both predicted and measured variables have been reconstructed and plotted in real scale. Some points could be distinguished from the linear distribution without exceeding it overly but still the error remains extremely low.



(a)



(b)

Figure 5.7: Results of Fuel Consumption Prediction in comparison with time.

As the figure 5.7 indicates the model is fitting perfectly in the input data set, even in the testing set which picked randomly and consist the 30% of the input data set. One of the reasons of this high performance in the testing set is that, the data follows the same distribution with the training set of the 70%, so the model is familiar with the data and fits almost perfectly. As seems in the zoomed region the error doesn't exceeds the value of 2 kg/h while in the x-axis the predicted values follows the trend of the measured values without any delay.

5.2.2 Validation Results

To validate the accuracy of the models, the models have to be tested in completely unknown data. For this purpose a new data set was provided by LME. The range which the validation data set covers starts from approximately 900 rpm up to 1660 rpm, in a period of 15 minutes,in total 91168 detailed data has been recorded. As applied before the initial validation data was preprocessed and configured per time step 0.1 sec instead of 0.01 sec which was firstly given, without losing any valuable information. The validation set are described at table 5.3:

Variable	Mean	Min	Median	Max
Fuel Consumption [kg/h]	26.2847	2.7	26.35	49.75
Lambda [-]	2.44	1.26	1.57	6.19
Exhaust Gas Mass Flow [kg/h]	598.19	261.4	549.6	961.6
MAP [kPa]	70.1	6	64.0	136.0
Torque Reference [%]	48.95	9	53	83
Rot. Speed [rpm]	1321.76	894.25	1327.56	1661.0
Engine Torque [Nm]	626.02	14.713	697.58	1143.5
EGR Command [%]	19.79	0.00	24.83	38.72
Exhaust Gas Temperature [$°C$]	358.43	246.69	381.31	397.5

Table 5.3: Validation Data overview.

The Variables from the validation data set needed for the Fuel Consumption Model are shown below in comparison with time:

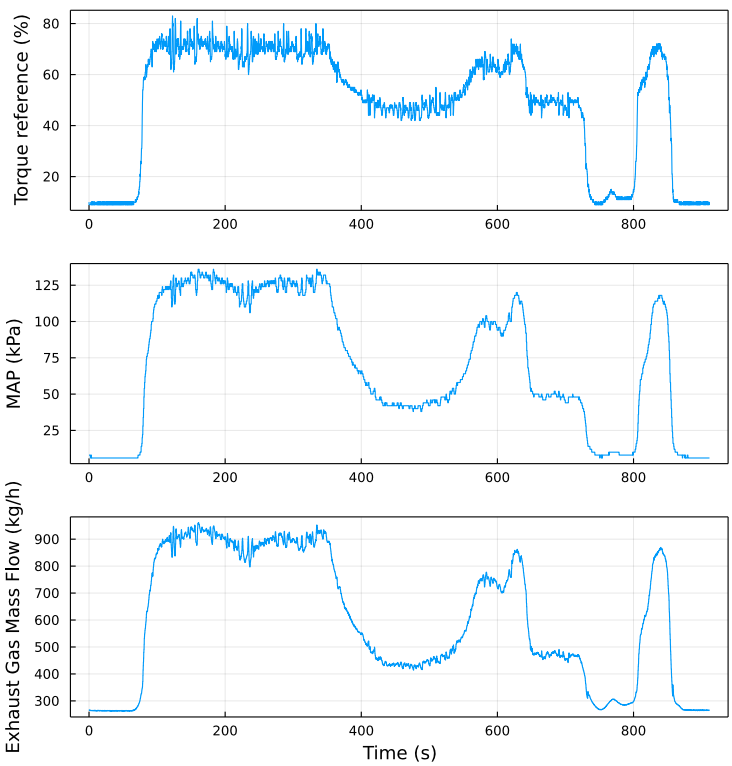


Figure 5.8: Validation data in comparison with time.

The Model's performance in these unknown data set is significant accurate, in particular Mean Absolute and Mean Squared Error calculated as :

- **MAE:**0.38 kg/h
- **MSE:**0.33 $(kg/h)^2$

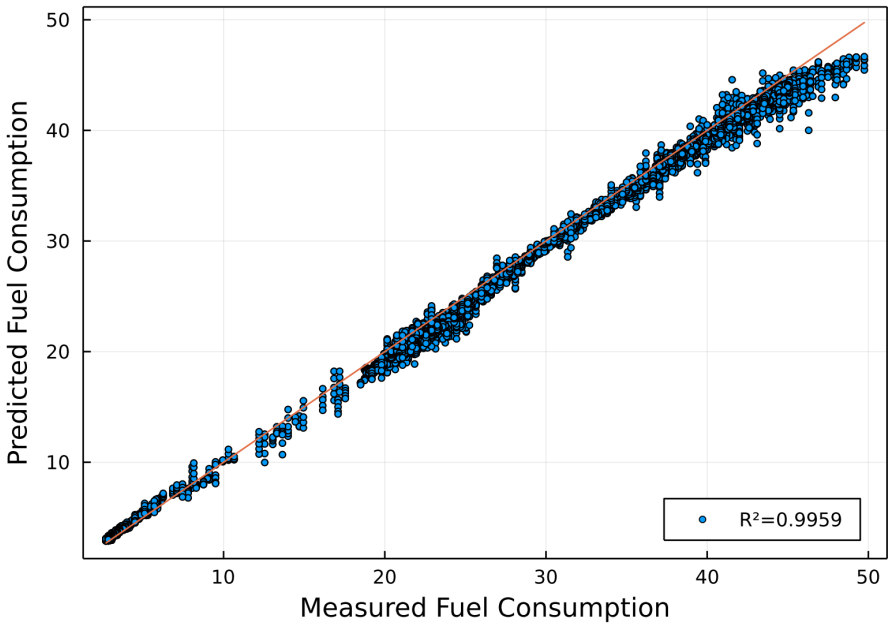
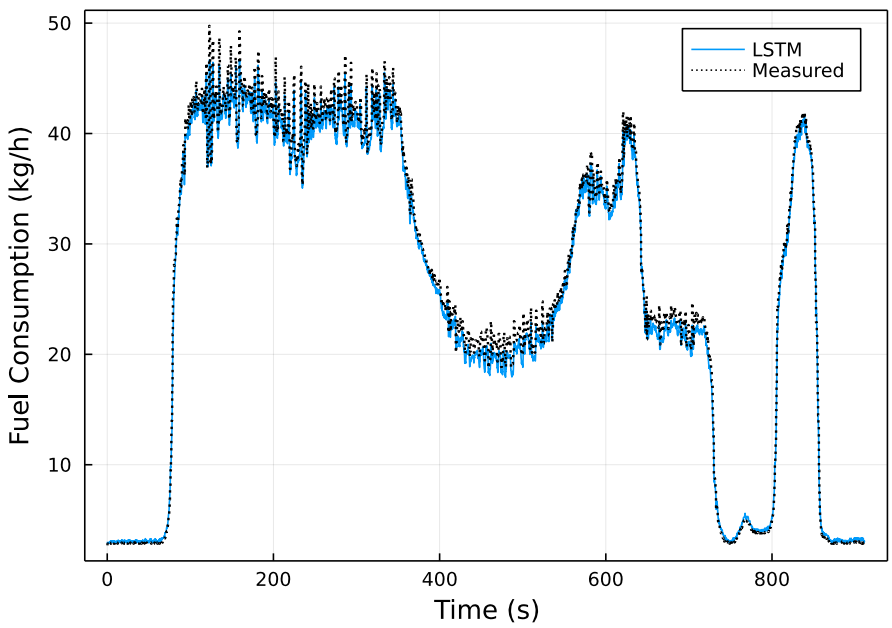
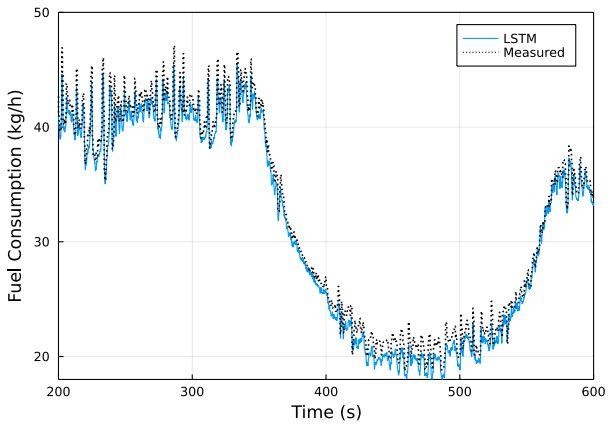


Figure 5.9: Validation Data scatter plot Predicted vs Measured Data.

In the above scatter plot the R^2 score is presented, and calculated as 99.59% accuracy. Besides that the linear distribution of different points confirms the satisfactory results which Model produces.



(a) Validation Data, Predicted and Measured, in comparison with time.



(b) Zoomed region.

Figure 5.10: Validation Data figures.

As it could easily be observed in the above figures, the difference between the Predicted and the Measured values doesn't exceeds the value of 2 kg/h while in the x-axis there is not any delay. The predicted values follows the trend of actual values giving a quite satisfactory result even when it comes to noisy regions.

5.3 MAP Model

5.3.1 Training Results

MAP variable follows the same pattern as the Fuel Consumption but in different scaling as it seems in figures 3.3a and 3.3f. For the MAP model the **Exhaust gas Mass Flow**, **Fuel Consumption** and **Torque reference** have been selected as input variables according to section 4.3.2 and SelectBest function results. The Seaborn function PairGrid was used to visualize the relation between Models inputs.

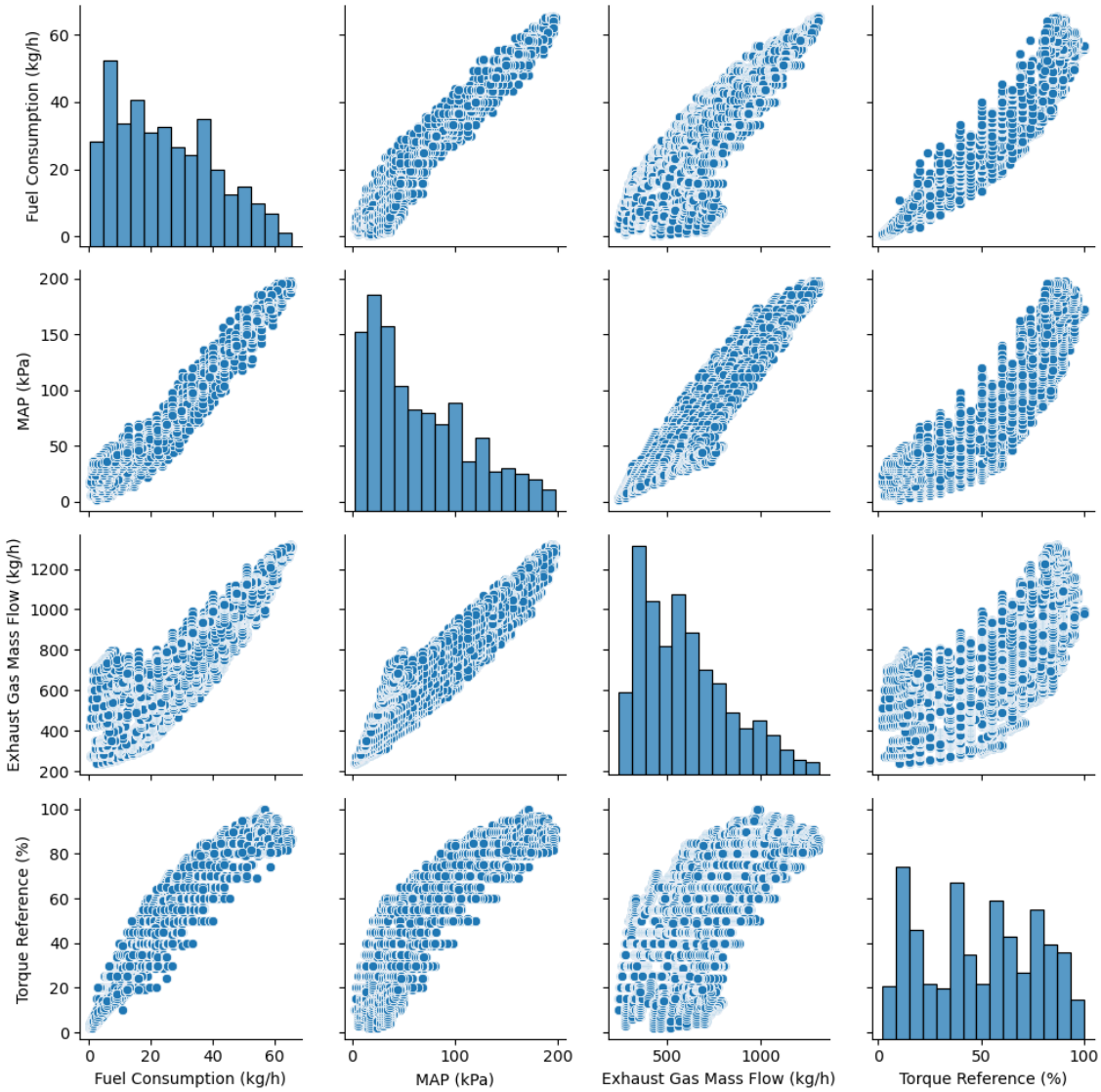


Figure 5.11: Pairwise Distribution.

In figure 5.11 the linear relation between the variables could be observed, even in the case of Torque reference where the relation incline to be parabolic.

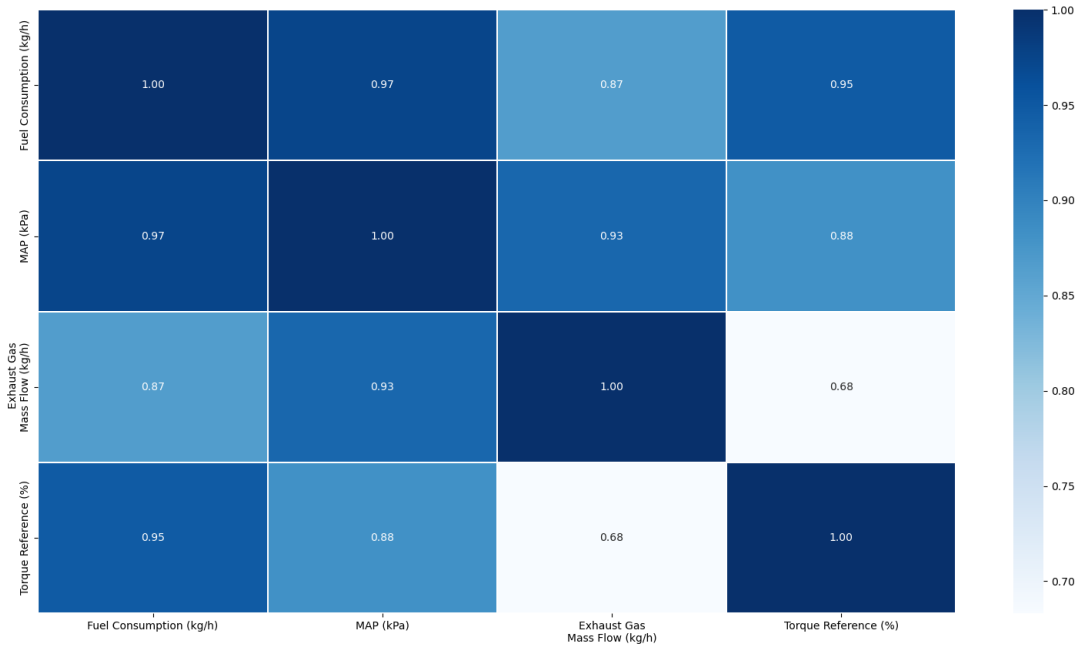


Figure 5.12: Heatmap of Pearson Correlation Coefficient.

As the heatmap indicates, the selected inputs variables are highly correlated with MAP variable. MAP model consist of one LSTM input layer using 3 time steps in each sample, one hidden, and one output layer, connected to each other with 32 neurons using ReLU as activation function as configured in section 4.2.

Inputs	MAP (kPa) Torque reference (%) Exhaust gas Mass Flow (kg/h)
Hidden Layers	1
Trainable Parameters	5761 (4672-1056-33)
Optimizer	ADAM (0.001)
Sample Time Step	3

Table 5.4: MAP Model Characteristics synopsis.

During the training procedure, 30 epochs were used with ADAM optimizer of learning rate $\eta = 0.001$. In order to visualize the training performance the MSE was saved during both training and testing process in each epoch. Figure 5.14 presents the model's loss function (Mean Squared Error).

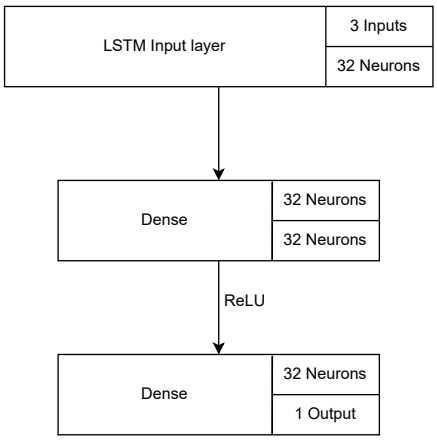
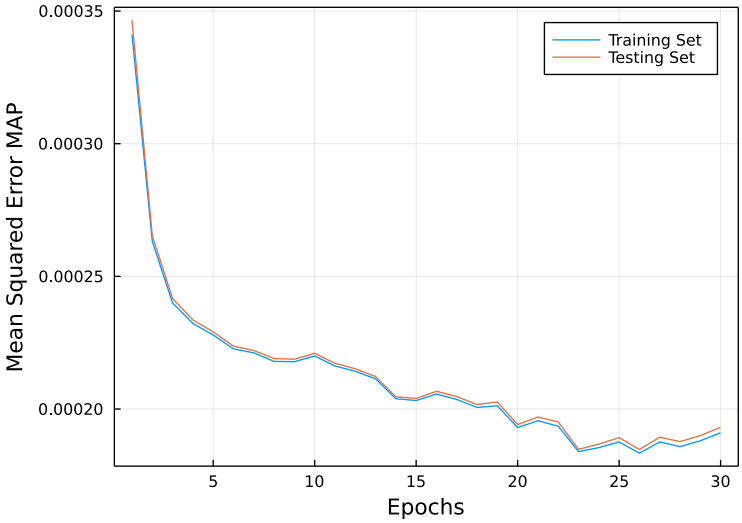
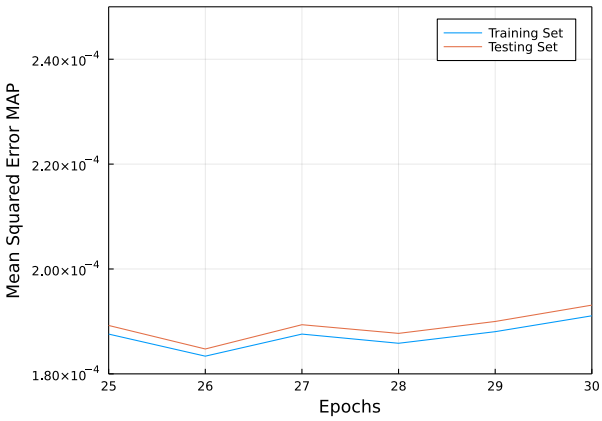


Figure 5.13: Model Chain.



(a) MSE during all epochs.



(b) Zoomed region.

Figure 5.14: Mean Squared Error during Training.

The MSE of the training process presents a good fitting result avoiding over fitting since Training and Testing set error are almost identical until 25 epoch and in the rest 5 epochs the gap between them is not exceeding the value of 0.00002 kPa^2. The R^2 score, calculated for all given data (training and testing), is also concluded in the figure while the Table 5.5 presents the detailed results of training process for MAP Model:

MAP Model Results						
Neurons	Input/Output Variables	Trainable Parameters	Model Accuracy (R^2) (Training Comp /Test Comp)	Mean Absolute Error (Train data /Test data)	Allocations (Number/Memory)	Time Cost (sec)
32	3/1	5761	99.69% / 99.69%	1.00% / 1.00%	83.87M /31.152 GiB	47.51

Table 5.5: Summarize Table of Model Results.

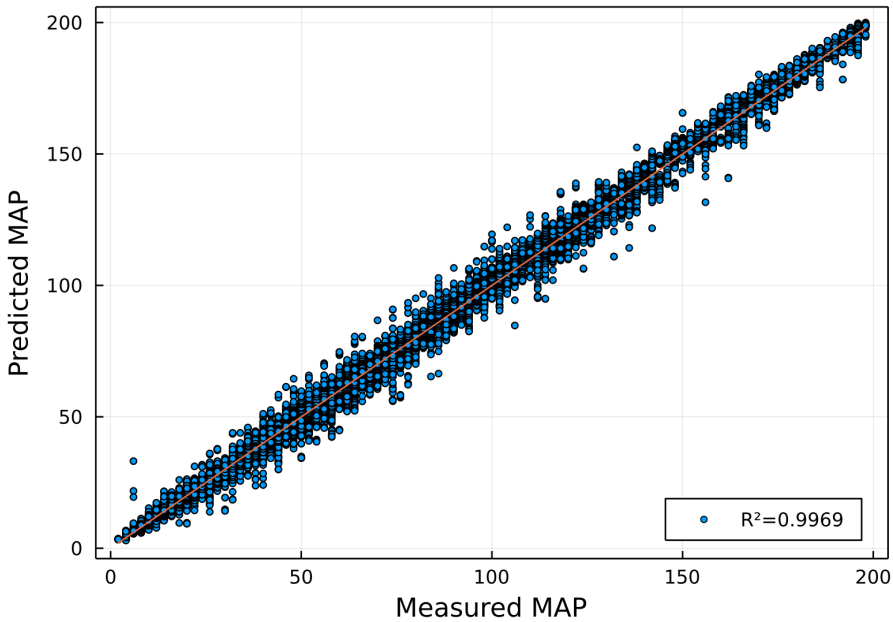
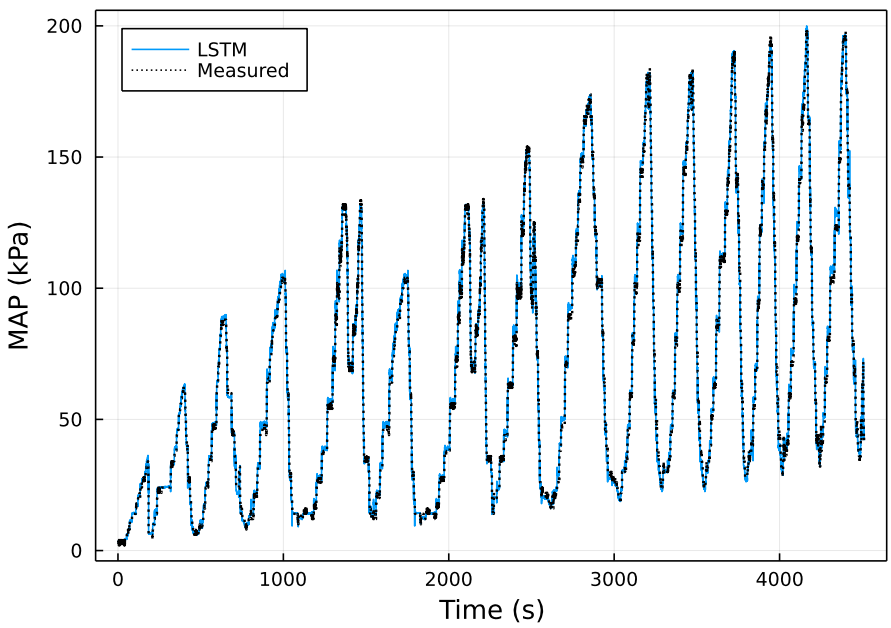
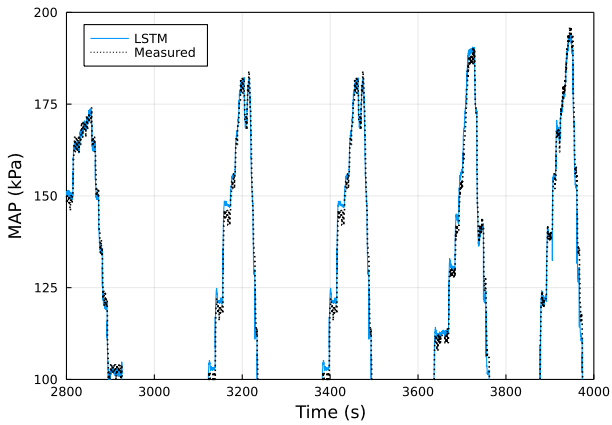


Figure 5.15: Scatter plot predicted-measured values.

In figure 5.15 the high accuracy of the model is presented while these two variables are linear distributed as the red line $y = x$ indicates. Both predicted and measured variables have been reconstructed and plotted in real scale. Some points could be distinguished from the linear distribution without exceeding it overly but still the error remains extremely low.



(a)



(b)

Figure 5.16: Results of MAP Prediction in comparison with time.

The fitting result is presented in figure 5.16 above, Model is fitting perfectly, without any delay in x-axis while the error, as it could be observed in zoomed area, isn't exceeding 5 kPa. Model follows the trend of the MAP over time.

5.3.2 Validation Results

Validation Data were used to confirm model's accuracy in total unknown area, in figure 5.17 model's input validation data are presented in comparison with time.

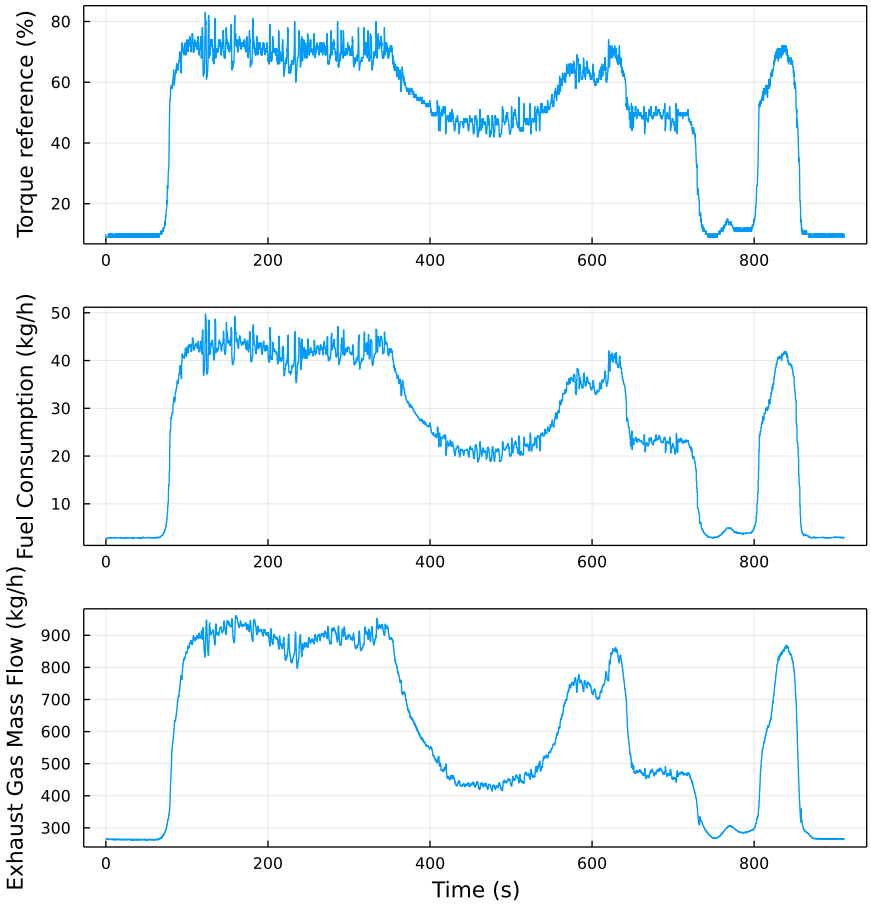


Figure 5.17: Validation data in comparison with time.

Model's performance in validation data could be considered as satisfactory enough taking into consideration the step behavior of MAP and the level of difficulty for a neural network model to adopt this kind of trend.

- **MAE:**3.57 kPa
- **MSE:**18.85 kPa^2

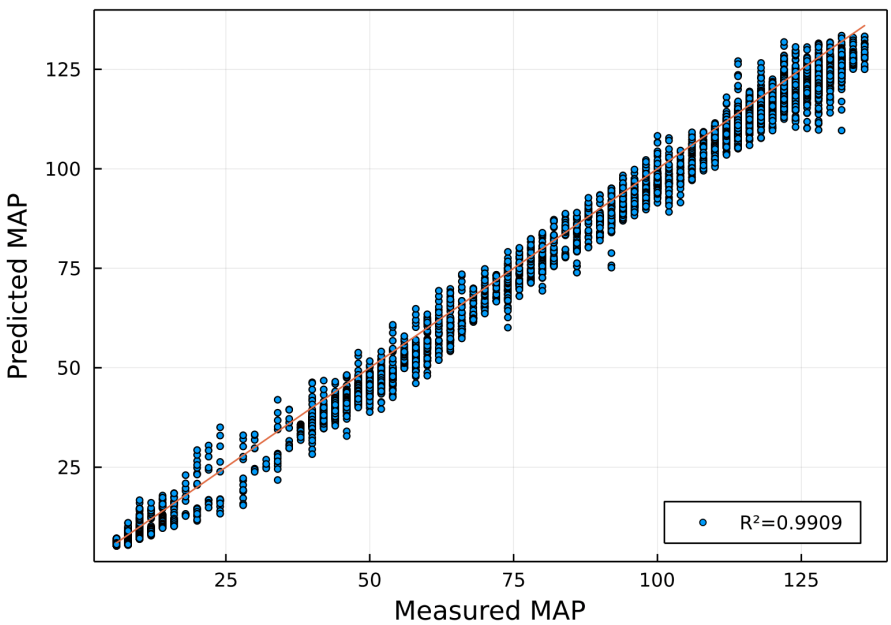
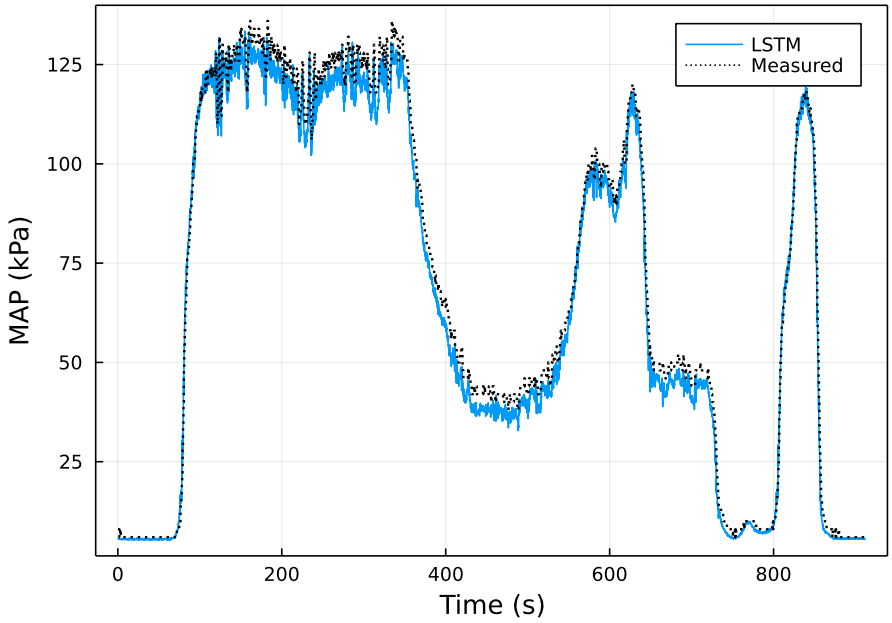
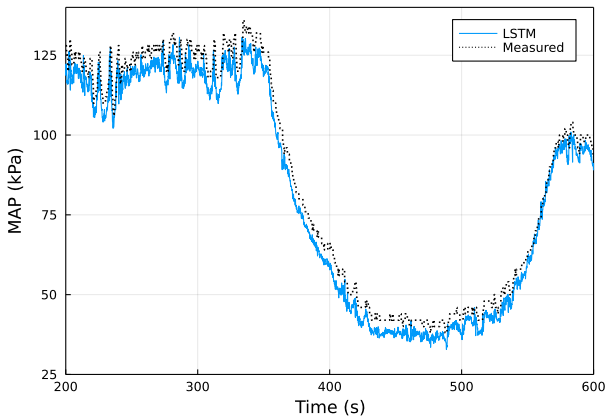


Figure 5.18: Validation Data scatter plot Predicted vs Measured Data.

As it could be noticed from the above scatter plot, the y=x line in flanked by the predicted values of MAP model, indicating model's significant accuracy but no perfection. Even though the MAP model isn't that accurate as Fuel Consumption Model, LSTM network is catching the validation data trend even in noisy regions as it could be seem in figures 5.19.



(a) Validation Data, Predicted and Measured,in comparison with time.



(b) Zoomed region.

Figure 5.19: Validation Data figures.

Once again it could easily observed that in the above figures, the difference between the Predicted and the Measured values doesn't exceeds the value of 4 kPa while in the x-axis there is not any delay.

5.4 Lambda Model

5.4.1 Training results

Lambda model configured to accept as input variables the **Exhaust Gas Temperature**, **Fuel Consumption**,**Torque reference** and **EGR Command** according to section 4.3.2 and SelectBest function results. The Seaborn function PairGrid was used to visualize the relation between model's inputs.

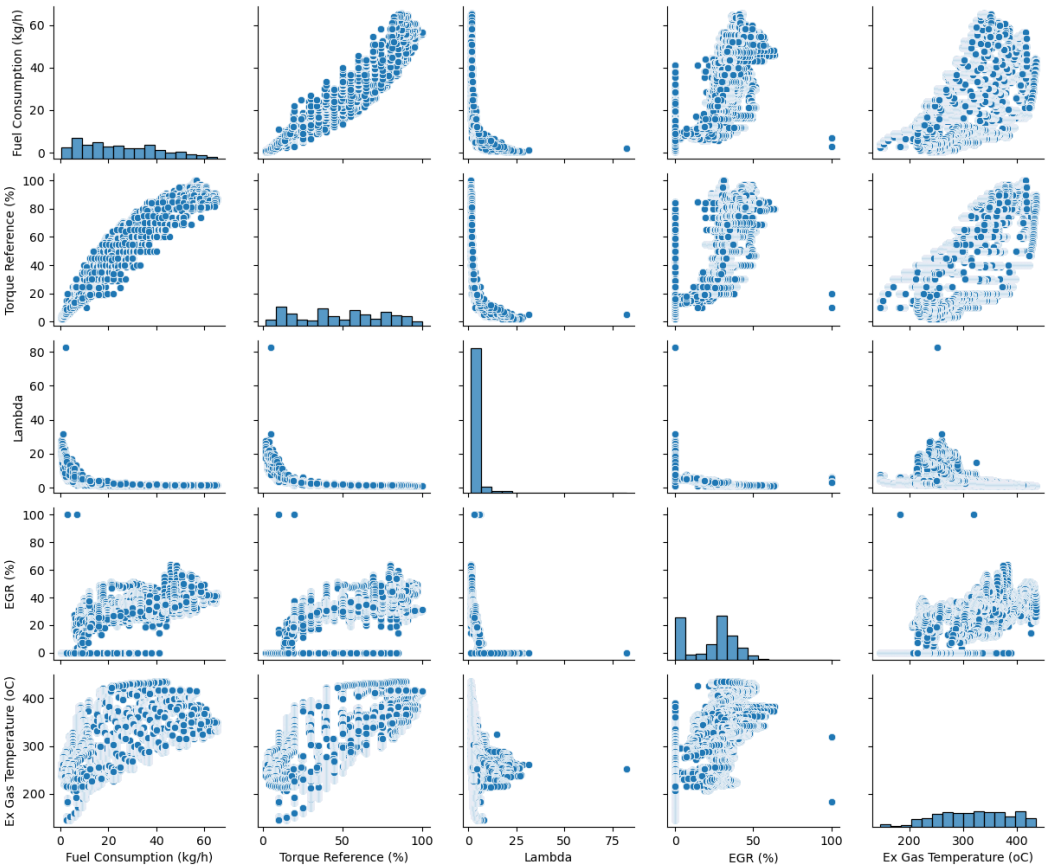


Figure 5.20: Pairwise Distributions.

The above figure demonstrate the non-linearity between the inputs variables, that doesn't necessary means that model will not fit properly. In particularly the relation between Lambda value and Fuel Consumption, EGR or Torque reference accordingly, seems to follows hyperbolic function distribution. As for the relation between Lambda and Exhaust Gas Temperature, scatter plots reveals a almost shapeless and complex figure.

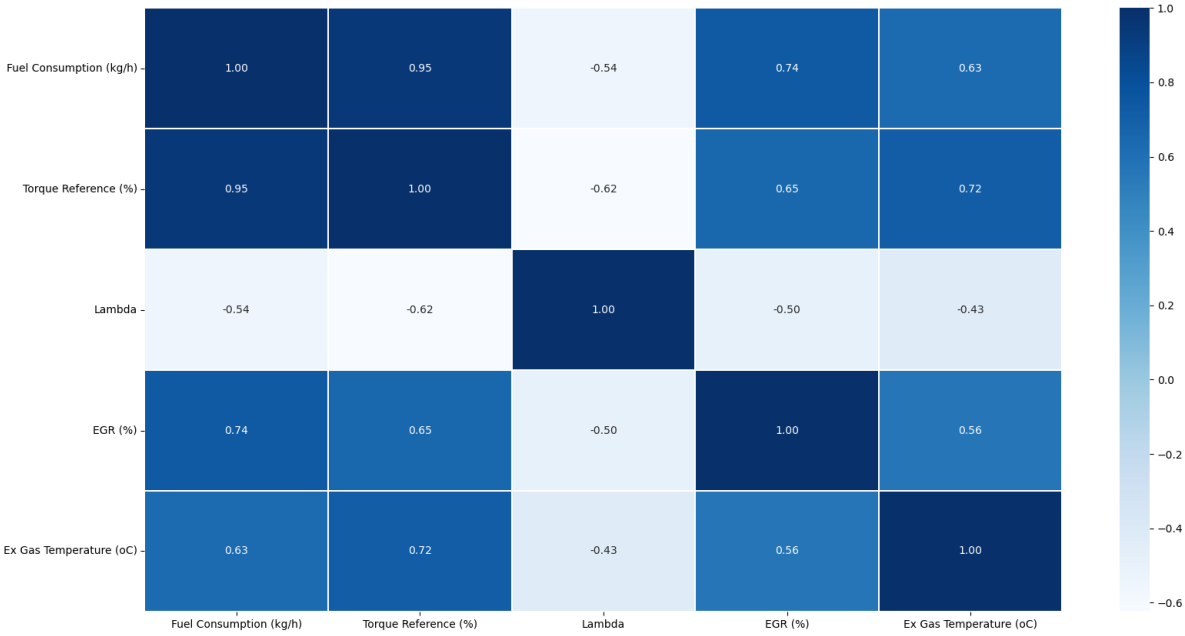


Figure 5.21: Heatmap of Pearson Correlation Coefficient.

Heatmap figure shows the existing correlation between inputs and Lambda but also shows that input variables are not correlated enough as the previous models were. After trial and error process concluded that for Lambda Model sigmoid activation function giving better results that ReLU. Finally Lambda model configured to have one LSTM input layer using 3 time steps in each sample, one hidden, and one output layer, connected to each other with 32 neurons using Sigmoid activation function.

Inputs	Exhaust Gas Temperature ($°C$) Fuel Consumption (kg/h) Torque Reference (%) EGR (%)
Hidden Layers	1
Trainable Parameters	5889 (4800-1056-33)
Optimizer	ADAM (0.001)
Sample Time Step	3

Table 5.6: Lambda Model Characteristics synopsis.

During the training procedure, 20 epochs were used with ADAM optimizer of learning rate $\eta = 0.001$. In order to visualize the training performance the MSE was saved during both training and testing process in each epoch. Figure 5.23 presents the model's loss function (Mean Squared Error).

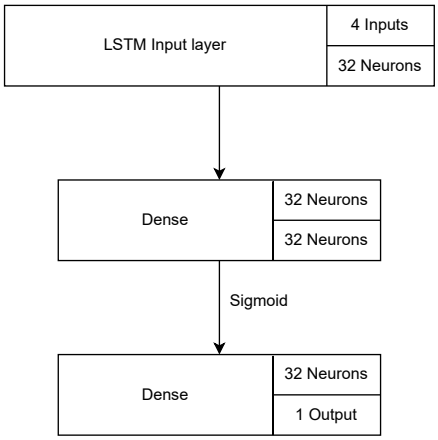
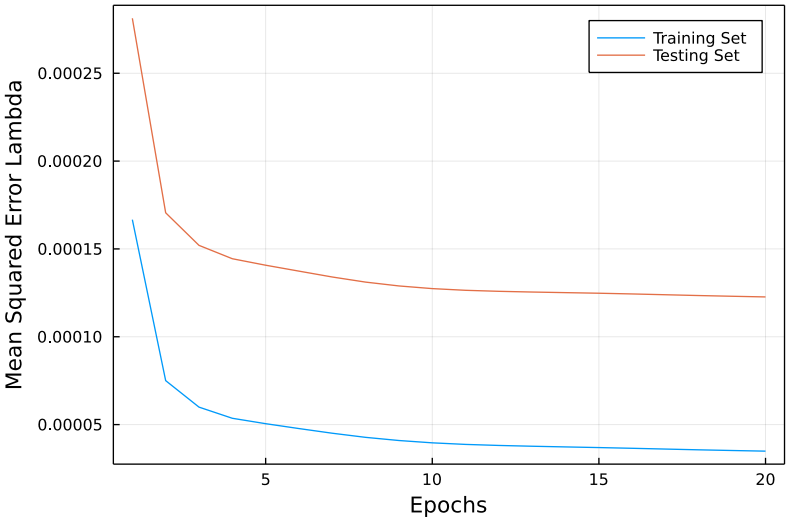
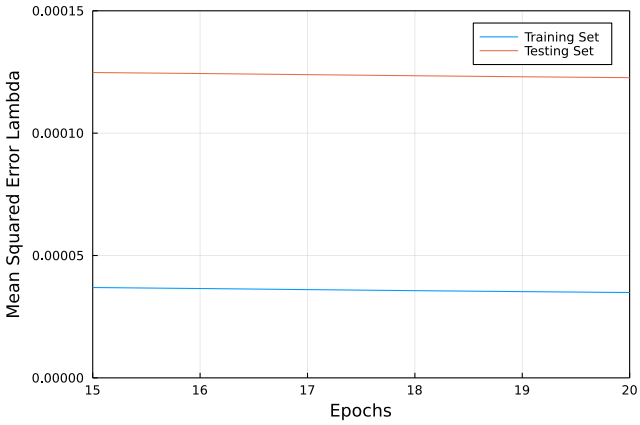


Figure 5.22: Model Chain.



(a) MSE during all epochs.



(b) Zoomed region.

Figure 5.23: Mean Squared Error during Training.

The MSE of the training process presents a good fitting result, over fitting ranges in very low values since Training and Testing set error difference is not exceeding the value of 0.001, after the 10 epoch, curves are getting smoother and almost constant. The R^2 score, calculated for all given data (training and testing), is also concluded in the figure while the Table 5.7 presents the detailed results of training process for Lambda Model:

Lambda Model Results						
Neurons	Input/Output Variables	Trainable Parameters	Model Accuracy (R^2) (Training Comp /Test Comp)	Mean Absolute Error (Train data /Test data)	Allocations (Number/Memory)	Time Cost (sec)
32	4/1	5889	97.20% / 91.55%	0.276% / 0.288%	55.84M /20.865 GiB	31.83

Table 5.7: Summarize Table of Model Results.

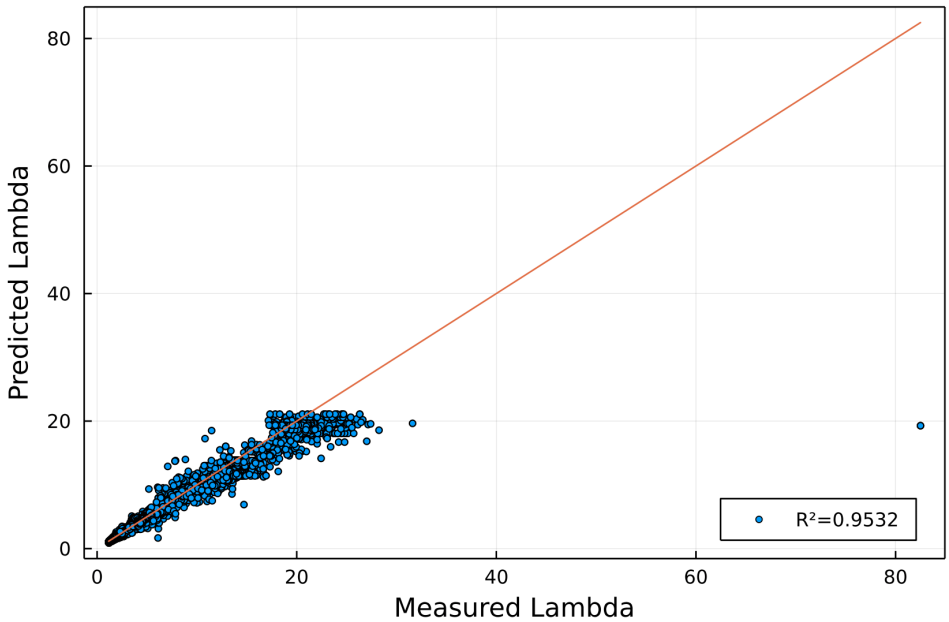
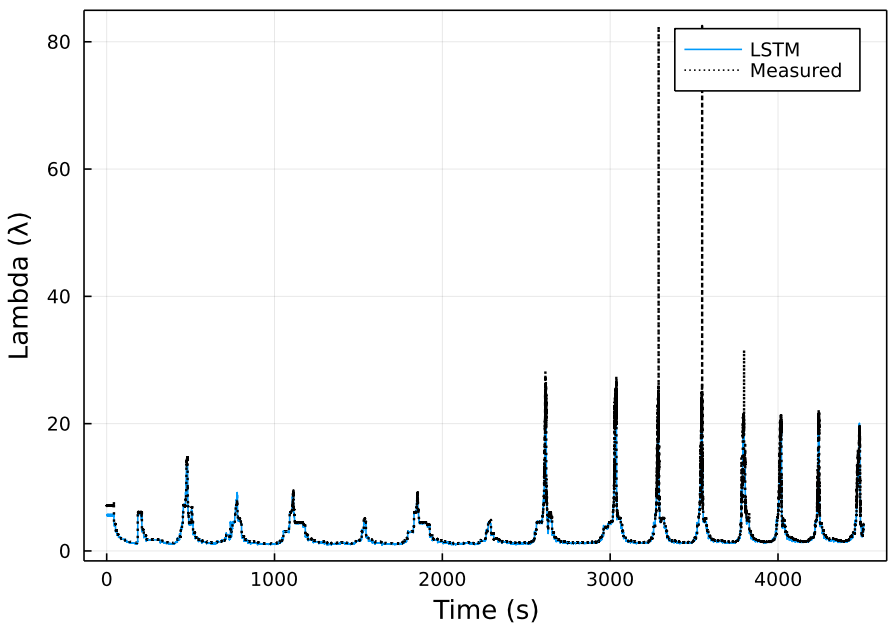
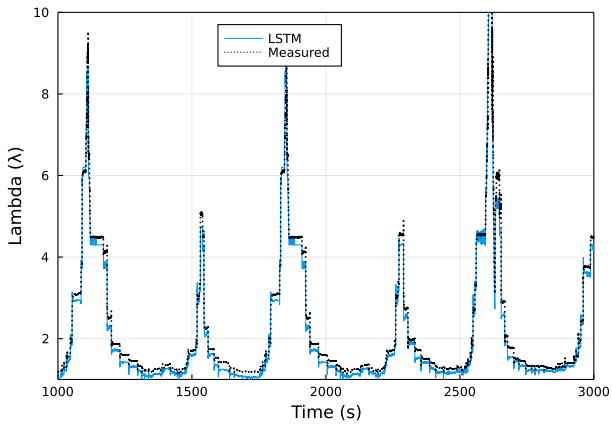


Figure 5.24: Scatter plot predicted-measured values.

It should be noticed that in training set there are some Lambda values that are not corresponding to reality and to real-time values for a diesel engine(as it seems in figure 3.3e), and occurred as mistake of physical sensors. Some of these points can be seen in the scatter figure 5.24. These values hasn't been excluded from data set hence model could easily handle them. Beyond that model is fitting perfectly, above mentioned figure shows that points are fitted on the $y = x$ line while accuracy reaches the value of $R^2 = 95.32\%$.



(a)



(b)

Figure 5.25: Results of Lambda Prediction in comparison with time.

The fitting result is presented in figure 5.25 above, Model is fitting perfectly, without any delay in x-axis. Also model isn't catching the false values of training set while the error, as it could be observed in zoomed area, isn't exceeding the value of 0.5. Model follows the trend of the Lambda over time.

5.4.2 Validations Results

Validation Data were used to confirm model's accuracy in total unknown area,in figure 5.26 models input validation data are presented in comparison with time.

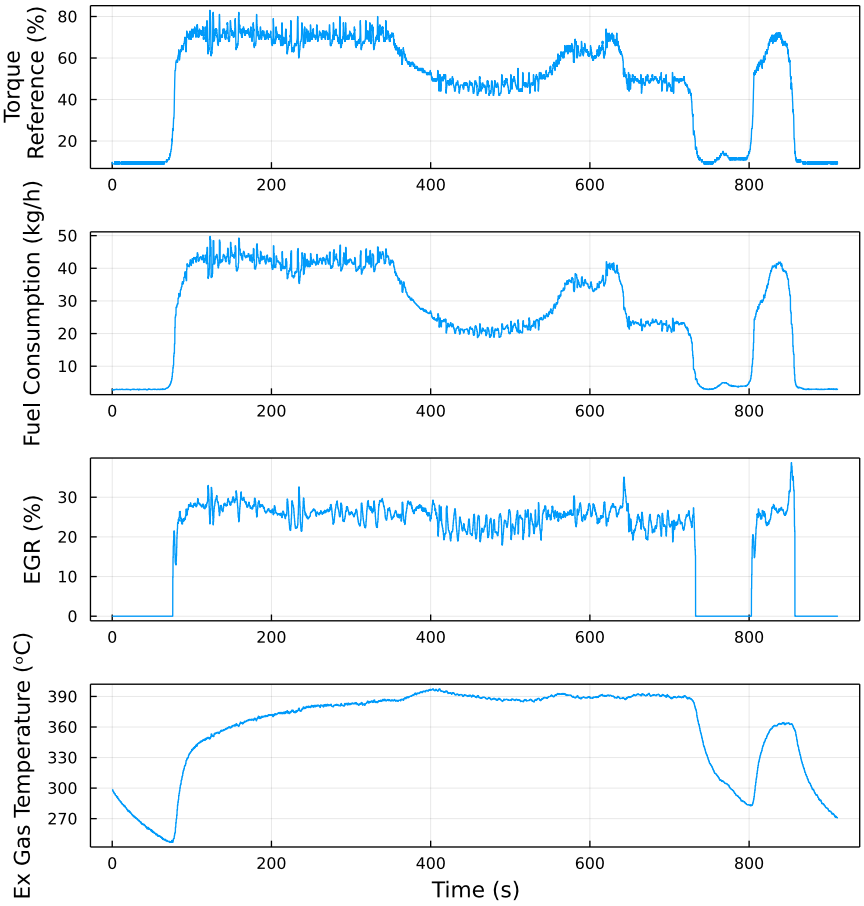


Figure 5.26: Validation data in comparison with time.

Models performance in validation data could be considered as satisfactory enough.

- **MAE:**0.092
- **MSE:**0.016

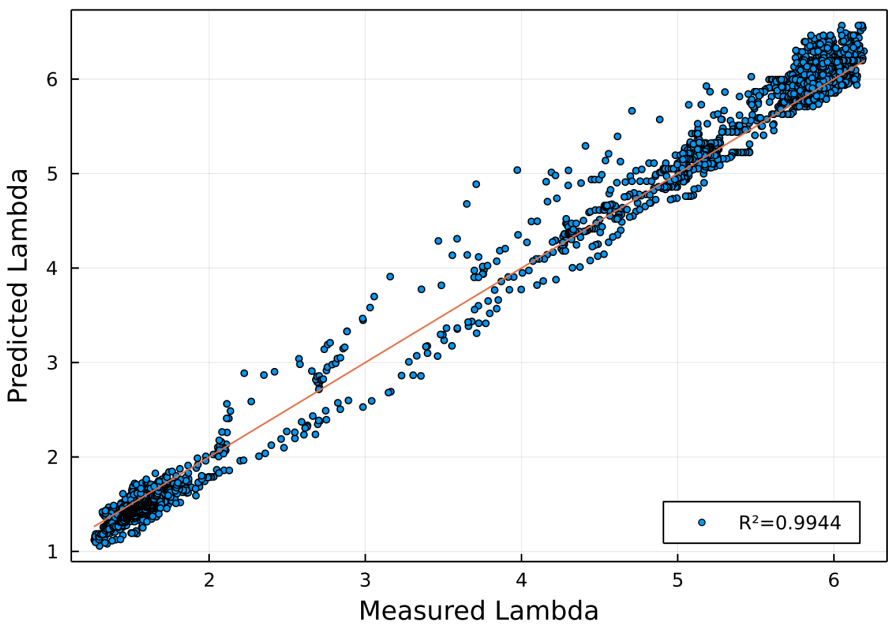


Figure 5.27: Validation scatter plot Predicted vs Measured Data.

The above scatter plot is confirming model's high precision since validation data is all lying on the red line and the R^2 score is equal to 99.44%. It is easy to observe that the vast majority of points are concentrated at the start and the end of validations data range, between the values of 1-2 and 5-6 respectively, where model is performing accurately. However in the intermediate values model performing less accurate but still satisfactory enough.

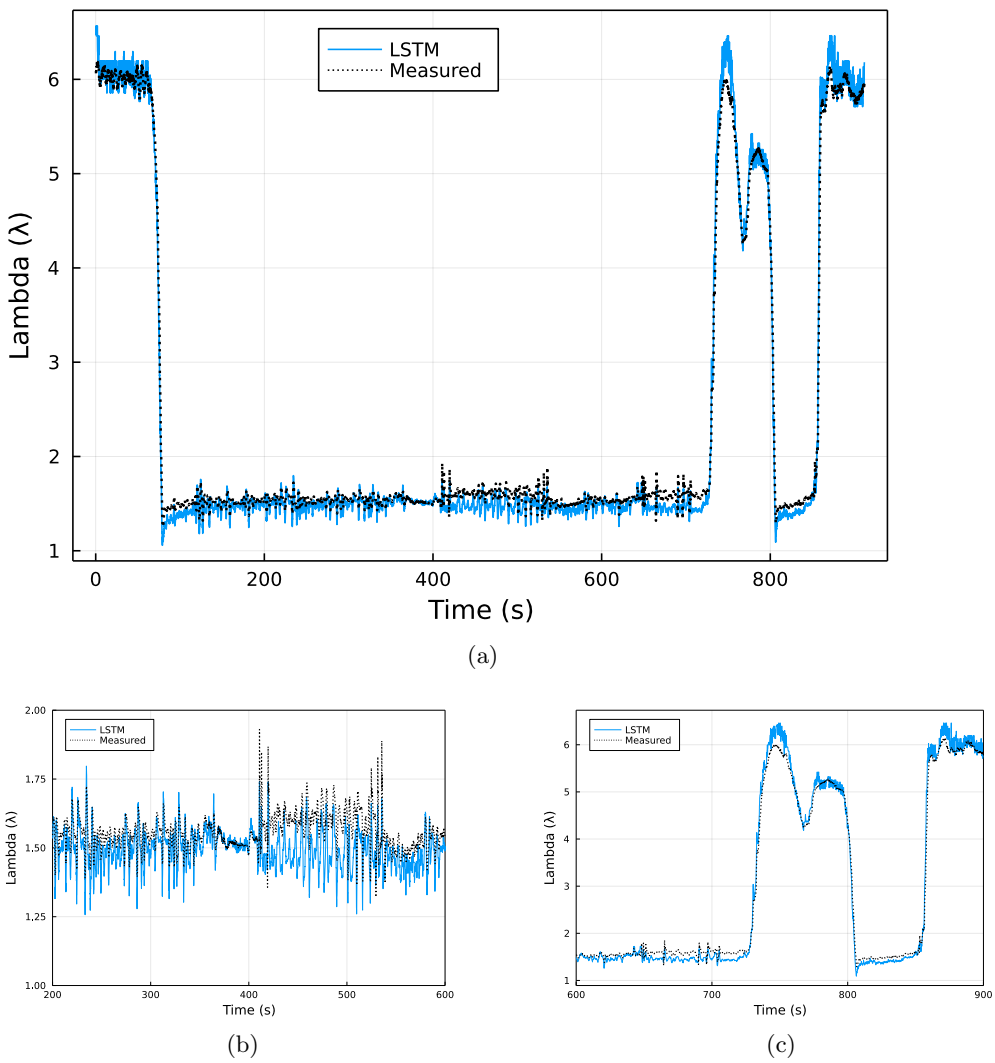


Figure 5.28: Results of MAP Prediction in comparison with time.

The above figures indicates that Lambda Model performing almost perfectly in the linear part while in the noisy areas which λ peaks the value o f 6, model follows the trend, without any delay. Error isn't exceeding the value of 0.5.

5.5 Engine Torque Model

5.5.1 Training Results

Engine Torque model configured to accept as input variables the **Torque reference**, **Fuel Consumption** and **MAP** according to section 4.3.2 and SelectBest function results. The Seaborn function PairGrid was used to visualize the relation between model's inputs.

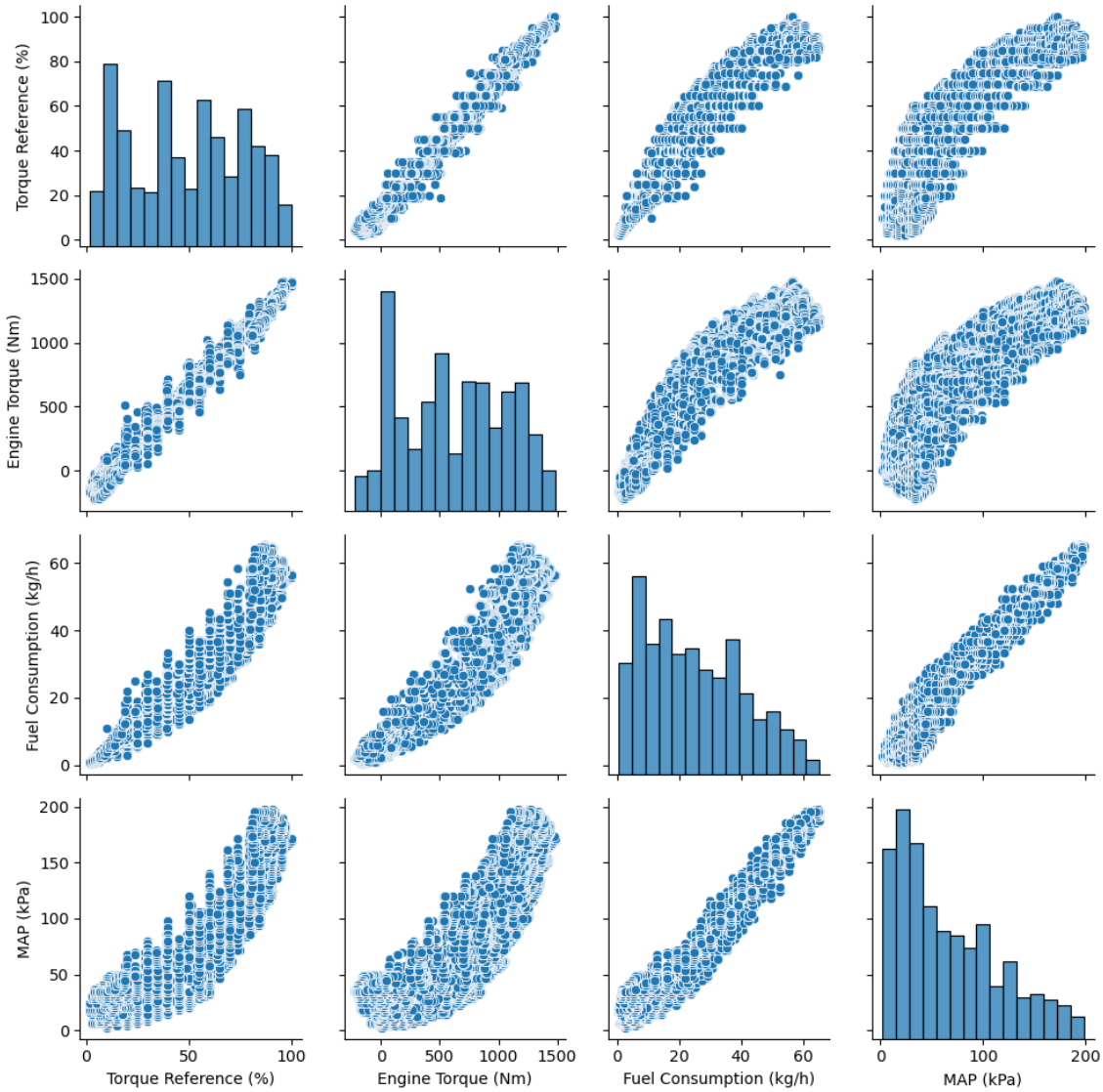


Figure 5.29: Pairwise Distributions.

As expected afromentioned Pairgrid shows the linear relationship between Engine Torque reference and Engine Torque and Fuel Consumption accordingly. On the other hand MAP presents a almost parabolic correlation with Engine Torque.

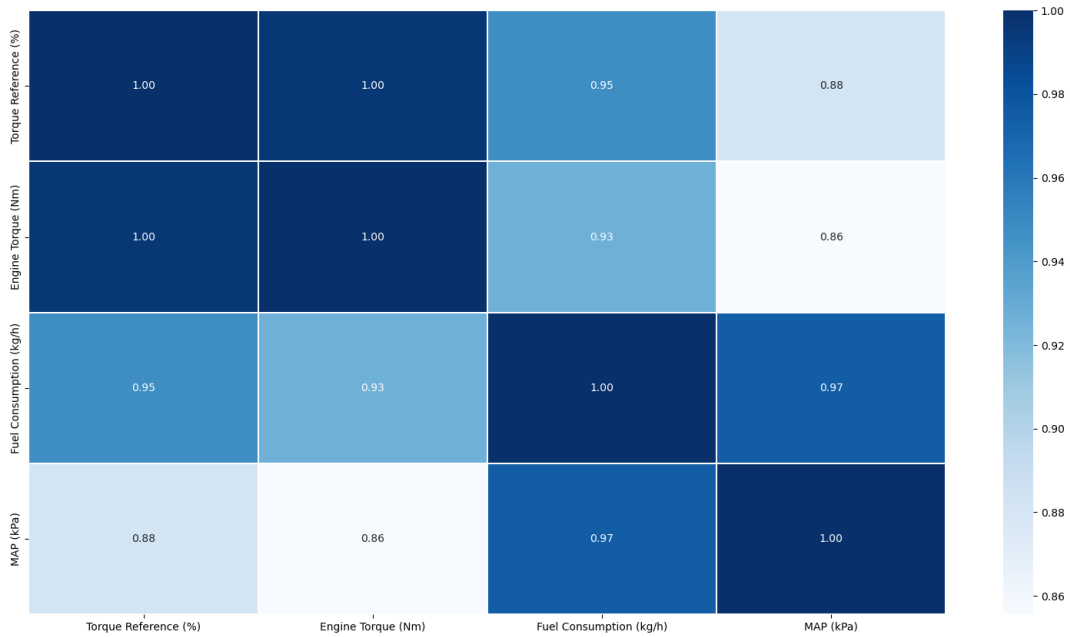


Figure 5.30: Heatmap of Pearson Correlation Coefficient.

Heatmap confirms the logical relationship between Engine Torque and Engine Torue reference, showing that these two variables are almost identical and strongly correlated.

Engine Torque model consist of one LSTM input layer using 3 time steps in each sample, one hidden, and one output layer, connected to each other with 32 neurons using ReLU as activation function as configured in section 4.2.

Inputs	Fuel Consumption (kg/h) Torque reference (%) MAP (kPa)
Hidden Layers	1
Trainable Parameters	5761 (4672-1056-33)
Optimizer	ADAM (0.001)
Sample Time Step	3

Table 5.8: Engine Torque Model Characteristics synopsis.

During the training procedure, 30 epochs were used with ADAM optimizer of learning rate $\eta = 0.001$. In order to visualize the training performance the MSE was saved during both training and testing process in each epoch. Figure 5.32 presents the model's loss function (Mean Squared Error).

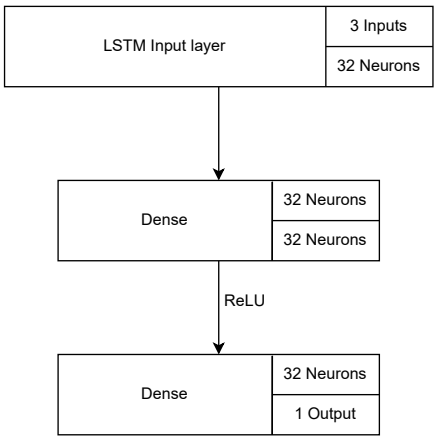
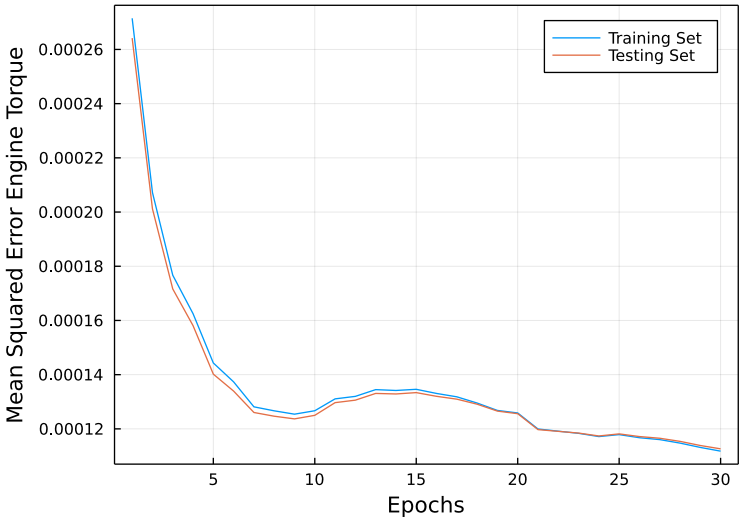
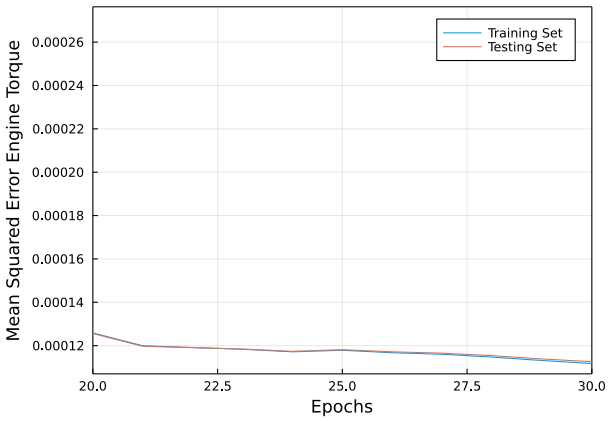


Figure 5.31: Model Chain.



(a) MSE during all epochs.



(b) Zoomed region.

Figure 5.32: Mean Squared Error during Training.

MSE in training process could be characterized as significant low while Training and Testing Error are almost identical avoiding overfitting phenomenons. After 20 epoch error is getting smoother and reduces slowly. The R^2 score, calculated for all given data (training and testing), is also concluded in the figure while the Table 5.9 presents the detailed results of training process for Lambda Model:

Engine Torque Model Results						
Neurons	Input/Output Variables	Trainable Parameters	Model Accuracy (R^2) (Training Comp /Test Comp)	Mean Absolute Error (Train data /Test data)	Allocations (Number/Memory)	Time Cost (sec)
32	3/1	5761	99.83% / 99.83%	0.792% / 0.792%	83.87M /31.153 GiB	55.97

Table 5.9: Summarize Table of Model Results.

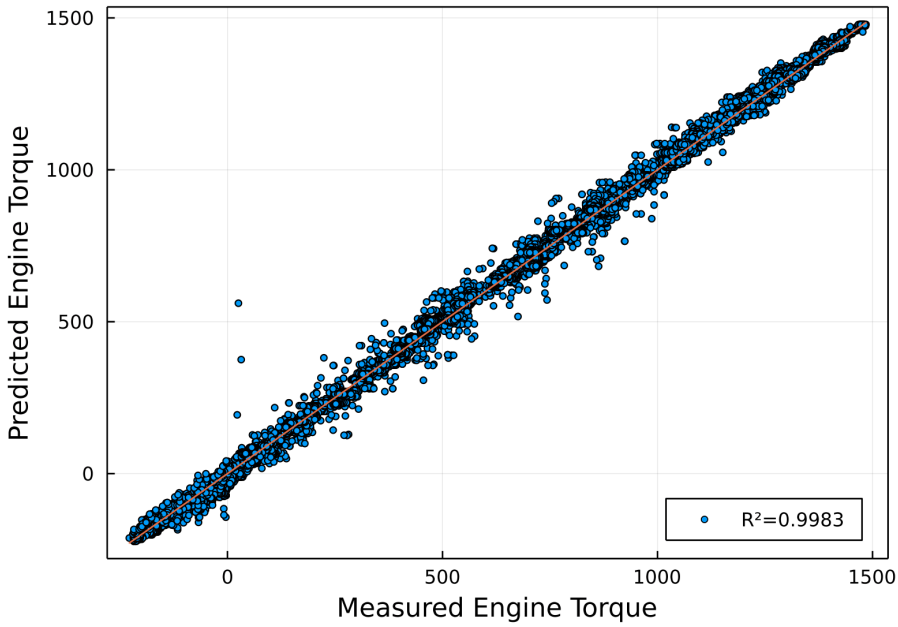
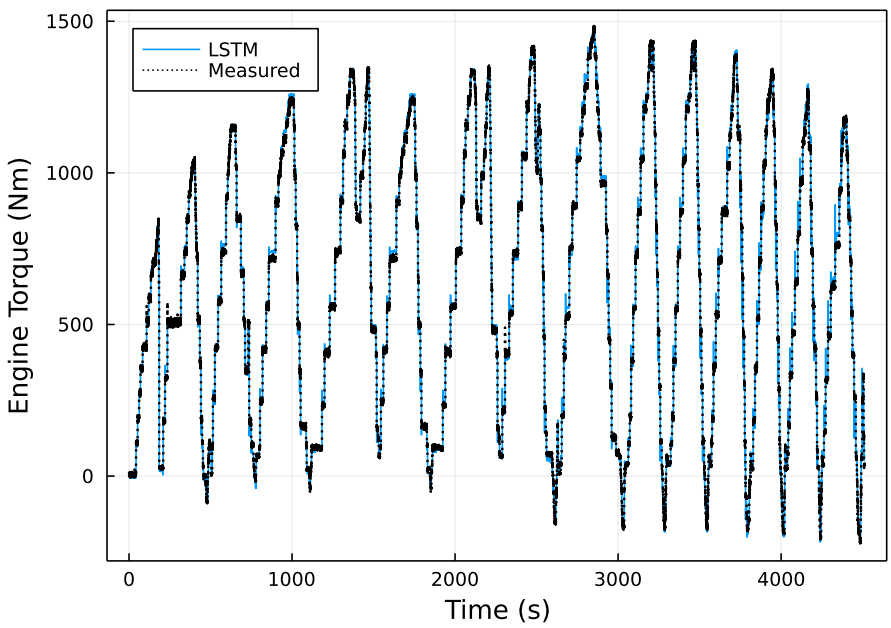
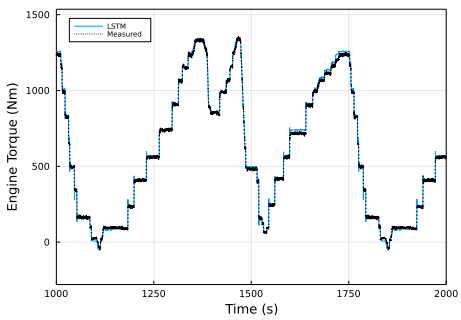


Figure 5.33: Scatter plot predicted-measured values.

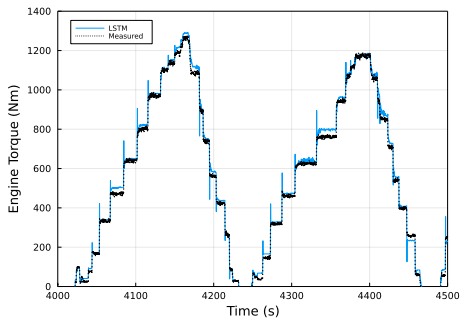
The above scatter plot confirms the overall good fitting of Engine Torque Model during training process, as the red line indicates the Predicted and Measured values are linear distributed. The accuracy reaching the significant high value of $R^2 = 99.83\%$.



(a)



(b)



(c)

Figure 5.34: Results of Engine Torque Prediction in comparison with time.

Fitting results is presented in figure 5.34 above, Model is fitting perfectly, without any delay in x-axis while the error, as it could be observed in zoomed areas, isn't exceeding 150 Nm. Model follows the trend of the Engine Torque over time.

5.5.2 Validation Results

Validation Data were used to confirm model's accuracy in total unknown area, in figure 5.35 model's input validation data are presented in comparison with time.

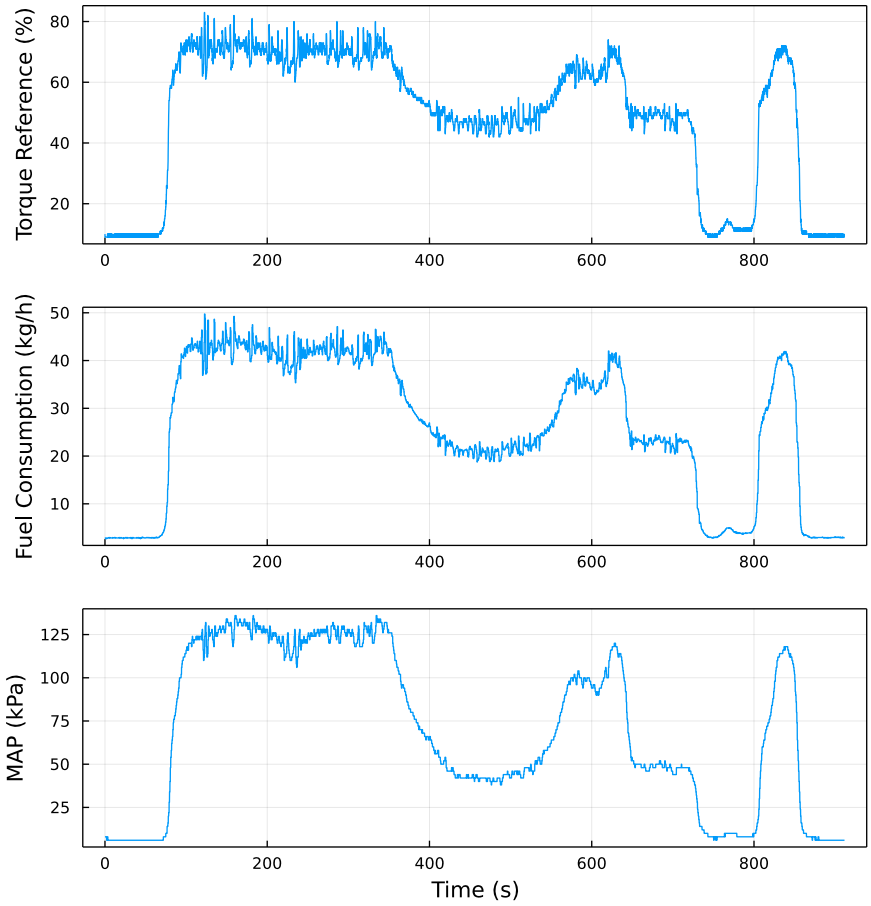


Figure 5.35: Validation data in comparison with time.

Model's performance in validation data, considering torque range from -228 up to 1500Nm, could be characterized as highly efficient.

- **MAE:**23.51 Nm
- **MSE:**896.22 $(Nm)^2$

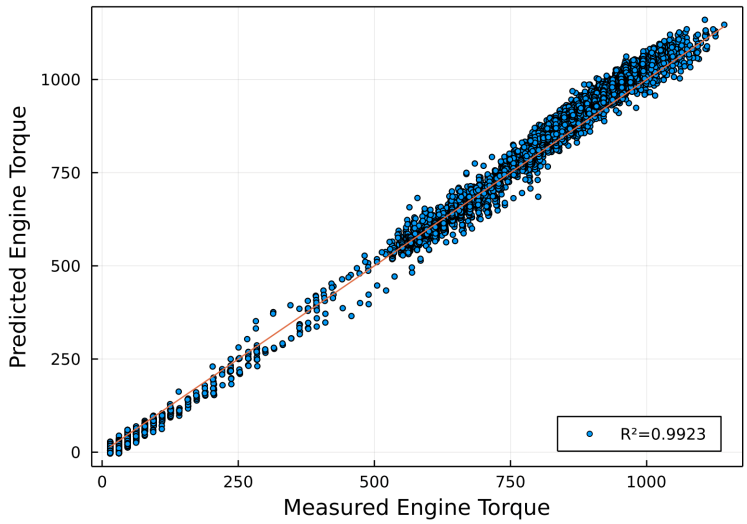
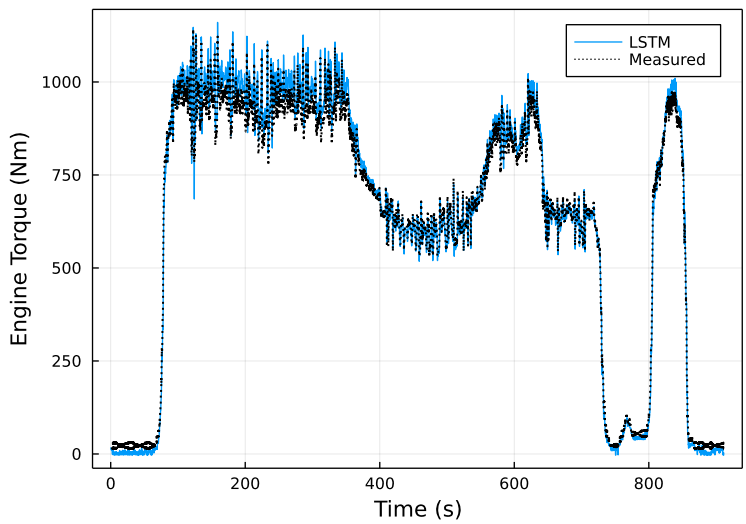
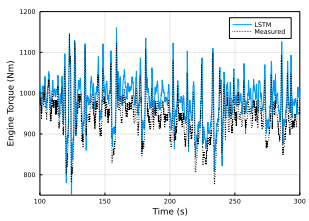


Figure 5.36: Validation scatter plot Predicted vs Measured Data.

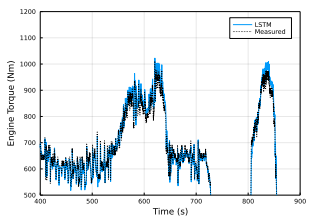
All points are placed too close to red line, that means that once again model accurate enough. In particular figure shows that in the most concentrated areas, points are linear distributed while the accuracy of overall data is $R^2 = 99.23\%$.



(a)



(b)



(c)

Figure 5.37: Results of Engine Torque Prediction in comparison with time.

The model is fitting perfectly on validation data, as could be seen in the above figures, even in noisy areas. Error remains very low.

5.6 NOx Model

5.6.1 Training Results

NOx model configured to accept as input variables the **Fuel Consumption**, **Torque Reference**, **RPM**, **Lambda** and **EGR** according to section 4.3.2 and SelectBest function results. The Seaborn function PairGrid was used to visualize the relation between model's inputs.

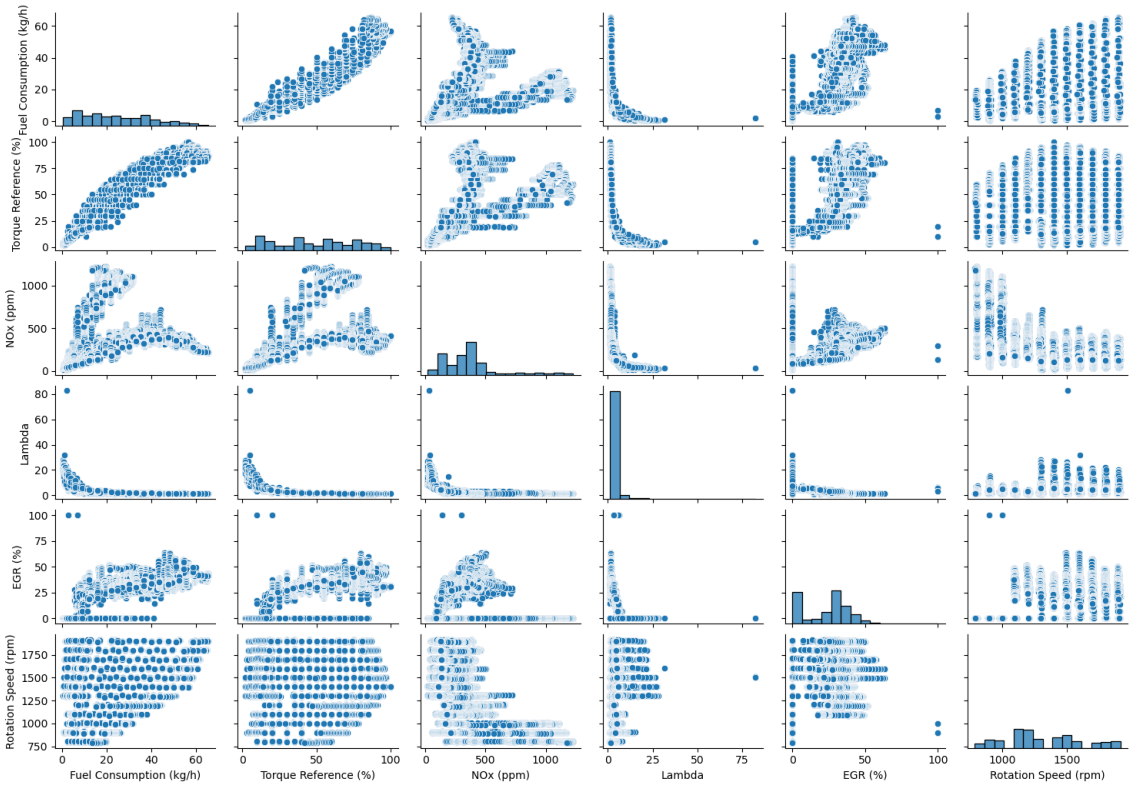


Figure 5.38: Pairwise Distributions.

The relation between NOx and the inputs features is more complicated than previous models due NOx formation natural procedure.

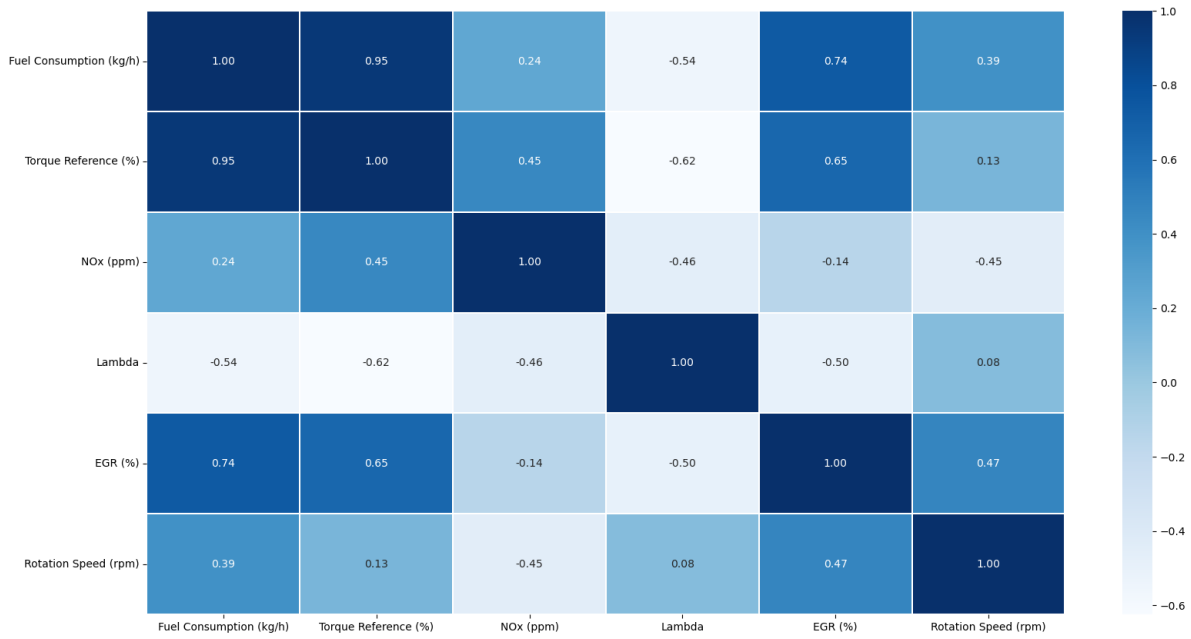


Figure 5.39: Heatmap of Pearson Correlation Coefficient.

As explained before Lambda value and EGR are strongly bonded with NOx value. In fact when EGR valve is open Lambda is increasing causing NOx reduction while when Lambda is small or the EGR valve is closed, NOx emissions reaching high concentration values.

NOx model consist of one LSTM input layer using 3 time steps in each sample, one hidden, and one output layer, connected to each other with 32 neurons using ReLU as activation function as configured in section 4.2.

Inputs	Fuel Consumption (kg/h) Torque Reference (%) Rotation Speed (rpm) Lambda (λ) EGR (%)
Hidden Layers	1
Trainable Parameters	6017 (4928-1056-33)
Optimizer	ADAM (0.001)
Sample Time Step	3

Table 5.10

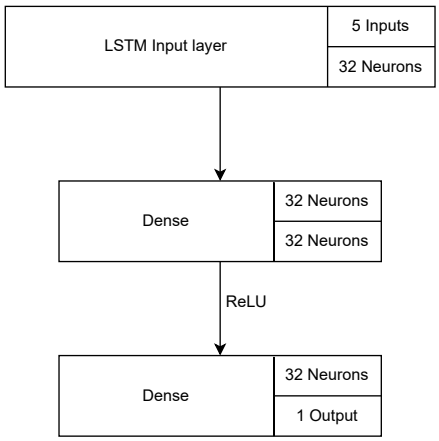
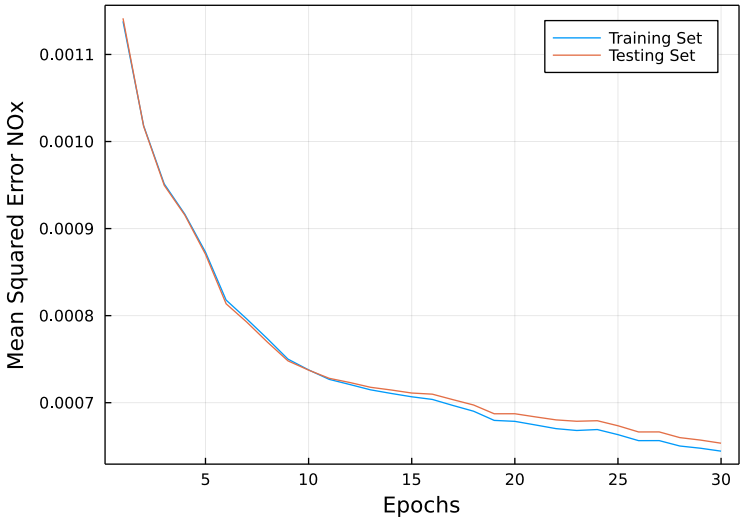
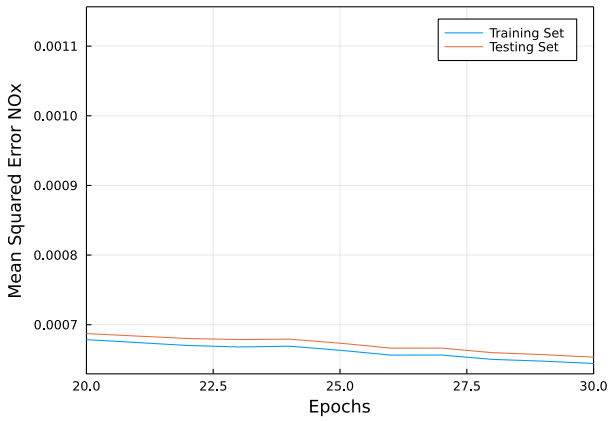


Figure 5.40: Model Chain.



(a) MSE during all epochs.



(b) Zoomed region.

Figure 5.41: Mean Squared Error during Training.

MSE in training process could be characterized as significant low while Training and Testing Error are almost identical avoiding overfitting phenomenons. Until epoch 14 training and testing error are almost identical and after that model slightly overfits. The R^2 score, calculated for all given data (training and testing), is also concluded in the figure while the Table 5.11 presents the detailed results of training process for Lambda Model:

NOx Model Results						
Neurons	Input/Output Variables	Trainable Parameters	Model Accuracy (R^2) (Training Comp /Test Comp)	Mean Absolute Error (Train data /Test data)	Allocations (Number/Memory)	Time Cost (sec)
32	3/1	6017	98.26% / 98.21%	1.707% / 1.715%	83.87M /31.452 GiB	53.46

Table 5.11: Summarize Table of Model Results.

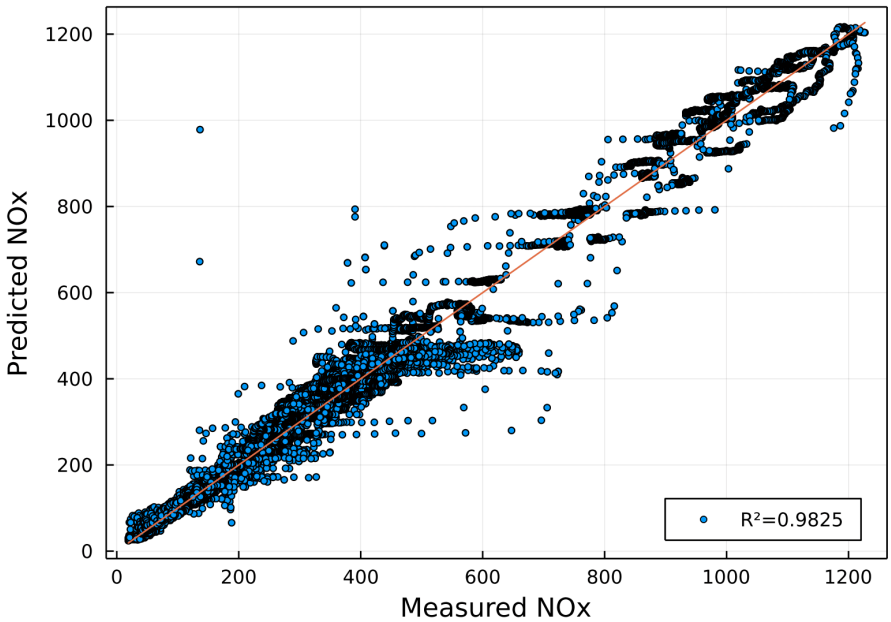
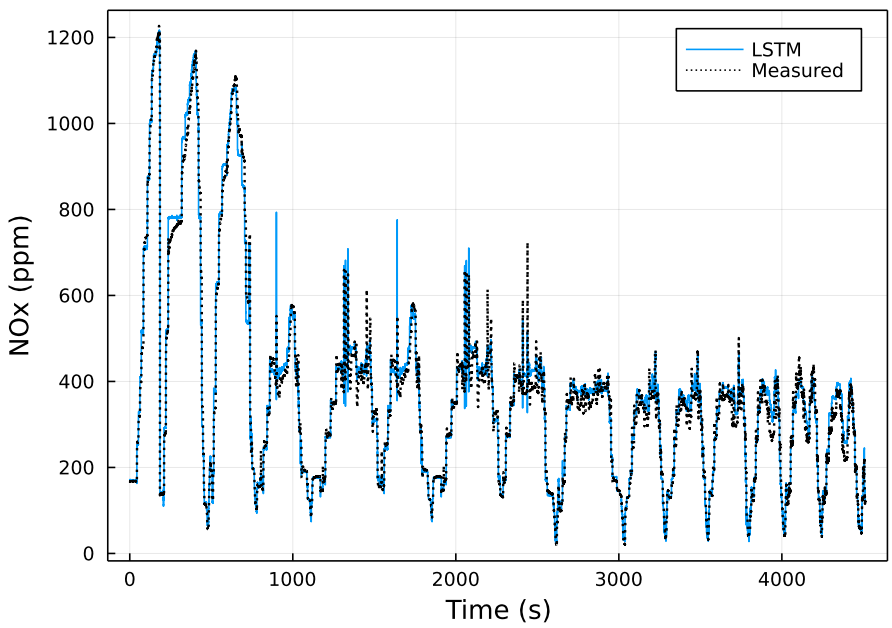
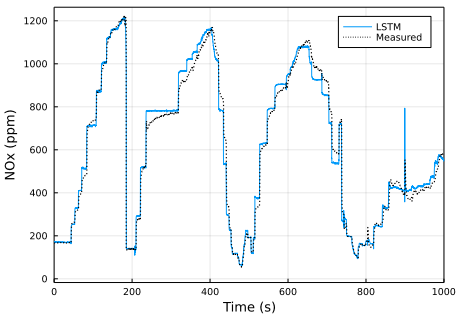


Figure 5.42: Scatter plot predicted-measured values.

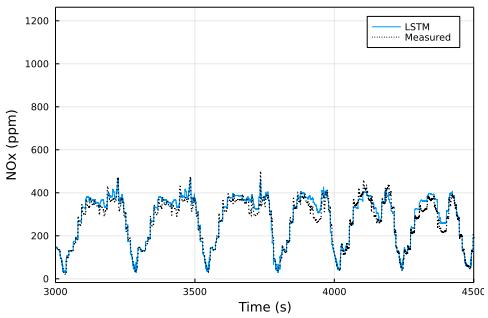
NOx model is significant accurate in the vast majority of NOx data set while the divergences from the $y = x$ line could be considered limited enough. However after the intermediate region until the end of NOx range there is a gap around the y=x line. This "failure" is reasonable and expected, since there are approximately 9 rapid up-and-downs of more than 200 ppm in a range of 2000 seconds, in the areas of the 800 sec-3000 sec. Consequently, the model instead of predicting accurately all these peaks and valleys, it reaches a mean between their minimum and maximum values. R^2 score is reaching the 98.25%, this result is satisfactory taking into consideration the complexity of NOx feature.



(a)



(b)



(c)

Figure 5.43: Results of NOx Prediction in comparison with time.

As expected, due to model's accuracy, NOx model is fitting quite well, following exactly the trend and direction of the curve even in the noisy areas.

5.6.2 Validation Results

Validation Data set for NOx emissions wasn't available as in the previous models. We consider here sufficient accuracy based on the training data set results.

Chapter 6

Conclusions

The main purpose of this thesis was to introduce and implement an LSTM neural network model for virtual sensors for a marine diesel engine. As presented in this thesis LSTM neural networks, due to their structure, have a significant high performance when it comes to engine parameters' predictions.
In particular, five models were tested in two different data sets, a relevant to training set (testing set or 30% of initial input data set) and a completely unknown one (validation set).
Testing set accuracy score was significantly high and almost identical to training one while training time cost was considered low. On the other hand 4 of 5 models were tested in validation set, which consists of experimental data under realistic operation conditions and they adopted perfectly to the data pattern, achieving results of accuracy of more than 99%.
In conclusion, LSTM-based models predict with significant accuracy the dynamics of engine parameters, both in the testing and validation set, following the pattern of the measured data.

Bibliography

[1] D. Martin, N. Kühl, and G. Satzger, "Virtual sensors," *Business & Information Systems Engineering*, vol. 63, no. 3, pp. 315–323, 2021.

[2] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, vol. 4. Springer, 2006.

[3] "Flux: The julia machine learning library." `https://fluxml.ai/Flux.jl/stable/`.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] I. Arsie, A. Cricchio, M. De 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.

[6] N. Planakis, G. Papalambrou, N. Kyrtatos, and P. Dimitrakopoulos, "Recurrent and time-delay neural networks as virtual sensors for nox emissions in marine diesel powertrains," tech. rep., SAE Technical Paper, 2021.

[7] V. Tzoumezi, "Parameters estimation in marine powertrain using neural networks," 2020.

[8] P. Dimitrakopoulos, "Real-time virtual sensor for nox emisions and stoichiometric air-fuel ratio λ of a marine diesel engine using neural networks," 2020.

[9] S. Shin, Y. Lee, J. Park, M. Kim, S. Lee, and K. Min, "Predicting transient diesel engine nox emissions using time-series data preprocessing with deep-learning models," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 235, no. 12, pp. 3170–3184, 2021.

[10] "3.1 linear regression." `https://d2l.ai/chapter_linear-networks/linear-regression.html`.

[11] C. Olah, "Understanding lstm networks." `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[12] "Nitrogen oxides (nox) – regulation 13." `https://www.imo.org/en/OurWork/Environment/Pages/Nitrogen-oxides-(NOx)-%E2%80%93-Regulation-13.aspx`.

[13] D. Crommelin and W. Edeling, "Resampling with neural networks for stochastic parameterization in multiscale systems," *Physica D: Nonlinear Phenomena*, vol. 422, 2021.

[14] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python.* Machine Learning Mastery, 2018.

[15] 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," tech. rep., SAE Technical Paper, 2017.

[16] "Sklearn.feature_selection.f_regression." https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression.

[17] S. SHARMA, "Activation functions in neural networks." https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.

[18] R. Gandhi, "A look at gradient descent and rmsprop optimizers." https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b, Jun 2018.

[19] A. L. Chandra, "Learning parameters part 5: Adagrad, rmsprop, and adam." https://towardsdatascience.com/learning-parameters-part-5-65a2f3583f7d, Jul 2021.

[20] M. L. Waskom, "seaborn: statistical data visualization." https://seaborn.pydata.org/index.html, 2021.