NATIONAL TECHNICAL UNIVERSITY OF ATHENS

School of Naval Architecture and Marine Engineering
Laboratory of Marine Engineering

Diploma Thesis

Prediction of Peak Cylinder Pressure of a Four-Stroke Marine Diesel Engine using Neural Networks

Galliakis Iakovos

Athens, February 2022

Acknowledgments

This diploma thesis 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 Associate Professor George Papalambrou.

I would first like to thank Associate Professor George Papalambrou for accepting me to conduct this thesis at the LME and by doing so, allowing me to work on the ever growing and exceptionally interesting field of machine learning. I would also like to sincerely thank him for his patience, support and his knowledge and experience that shared with me.

I would like to thank Doctoral Student Vasileios Karystinos for his invaluable assistance, as well as his patience and understanding. His immediate responses and supportive attitude helped me greatly through the development of this diploma thesis.

I would also like to express my gratitude to Professor Lambros Kaiktsis and Professor Gregory Grigoropoulos for evaluating my work and being members of my supervisors committee.

Finally, I would like to thank all my friends and family for being there for me, for their unending motivation and support throughout all these years of studies. I wouldn't be here without them, and for that I am deeply grateful.

Abstract

In recent years, the demand for more efficient operation of engines has lead to an increase in the need for inexpensive and reliable monitoring tools. One parameter that is of great importance to the work producing process of an internal combustion engine is the in-cylinder pressure. The most common method for measuring such a parameter is through a piezoelectric pressure sensor; this solution however is quite expensive and the installation impractical and time-consuming. Others, more complex indirect methods include prediction of the pressure waveforms via utilisation of the acoustic emissions of the engine, or through the momentary crankshaft speed. A different approach to this task is explored through this thesis; the utilization of artificial neural networks, a machine learning model, that by processing easy to acquire data, namely the engine Speed, Torque, Lambda and Specific Fuel Consumption (BSFC), aims to make accurate predictions of the peak cylinder pressure. By using datasets from two different engines, both however being of the four-stroke, diesel type, two model groups were created; each grouped housed a large amount of different neural network architectures, in order to deduce the best hyperparameters for this task. After training and testing, it was concluded that the models were successful in predicting the peak pressure, as accuracy of 99.32% and 97.04% was reached by Model Set A and Set B respectively; it was also discovered that using the BSFC parameter as input worsened the performance of the models, leaving the engine Speed-Torque-Lambda as the optimal input vector. All the calculations and model building utilized the Julia programming language, and specifically the Flux machine learning package.

List of Figures

2.1 A Machine Learning model during the training stage 15
2.2 A Machine Learning model after the training stage, during the validation stage 15
2.3 The Perceptron, as designed by Rosenblatt. 16
2.4 The NETtalk system schematic. 17
2.5 Venn diagram of the machine learning related fields of science. 19
2.6 A neural network diagram. 24
2.7 Analysis of a singular neuron. 24
2.8 Example of a support vector machine creating the optimal hyperplane between two distinct groups of data. 25
2.9 Example of a decision tree. 26
2.10 Example of a Bayesian network. 27
2.11 Outline of a typical neural network with a singular hidden layer. 32
2.12 Depiction of a two-layered neural network prediction, in four different target functions:(a) $f(x) = x^2$, (b) $f(x) = sin(x)$, (c) $f(x) = |x|$ and (d) $f(x) = H(x)$, where $H(x)$ is the Heaviside step function. 33
2.13 Geometrical depiction of the error function above the weight space. 35
2.14 Depiction of the input and error information flow in a network. 40
2.15 Depiction of the four-stroke operating cycle. 42

3.1 The schematic of the engine of Dataset A. 47
3.2 Torque-Speed diagram of the 9 operating states of Dataset A. 48
3.3 Engine Power-Speed diagram of the 9 operating states of Dataset A. 48
3.4 Lambda-Speed diagram of the 9 operating states of Dataset A. 49
3.5 BSFC-Speed diagram of the 9 operating states of Dataset A. . 49
3.6 Peak Pressure measurements per engine cycles, for Dataset A. 50
3.7 Peak Pressure-Speed diagram of Dataset A. 50
3.8 Peak Pressure-Engine Power diagram of Dataset A. 51

3.9 A picture of the engine of Dataset B. 52
3.10 Torque-Speed diagram of the operating states of Dataset B. . 53
3.11 Engine Power-Speed diagram of the operating states of Dataset B. 54
3.12 Lambda-Speed diagram of the operating states of Dataset B. . 54
3.13 Peak Pressure measurements per engine cycles, for Dataset B. 55
3.14 Peak Pressure-Speed diagram of Dataset B. 55
3.15 Peak Pressure-Engine Power diagram of Dataset B. 56
3.16 Peak Pressure-Engine Power diagram of both Dataset A and Dataset B. 56
3.17 Peak Pressure-Lambda diagram of both Dataset A and Dataset B. 57
3.18 The training and testing groups. 59
3.19 Depiction of the sigmoid activation function. 63
3.20 Depiction of the hyperbolic tangent function. 64
3.21 Depiction of the Rectifier Linear Unit function. 65
3.22 Depiction of the Leaky Rectifier Linear Unit function. 66
3.23 Depiction of the effect of the learning rate parameter on the model training, in three different scenarios. 68
3.24 Example of a trained model name. 70
3.25 Analysis of the above model name. 70

4.1 The zero-error-line diagram of the best performing model of Set A, with Speed-Torque as input. 73
4.2 The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda as input. 74
4.3 The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-BSFC as input. 75
4.4 The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda-BSFC as input. 77
4.5 The MAPE of the best model of the Set A with the Descent optimizer, for various epochs and batchsizes. 79
4.6 The MAPE of the best model of Set A with the ADAM optimizer, for various epochs and batchsizes. 79
4.7 Comparison of the Descent and ADAM optimizer of the best Set A model, for various batchsizes. 80
4.8 Comparison of variations to the number of hidden layers and neurons for the best model of Set A. 81
4.9 Comparison of different learning rates for the best model of Set A. 82

4.10 Evaluation of different activation functions for the best model of Set A. 82
4.11 Comparison of different training epochs for the best model of Set A 83
4.12 The zero-error-line diagram of the best performing model of Set B, with Speed-Torque as input. 85
4.13 The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda as input. 86
4.14 The MAPE of the best model of Set B with the ADAM optimizer, for various batchsizes. 88
4.15 Comparison of variations to the number of hidden layers and neurons for the best model of Set B. 89
4.16 Comparison of different training epochs for the best model of Set B 89

A.1 The MAPE of the best ST model of the Set A with the Descent optimizer, for various epochs and batchsizes. 102
A.2 The MAPE of the best ST model of Set A with the ADAM optimizer, for various epochs and batchsizes. 103
A.3 Comparison of the Descent and ADAM optimizer of the best ST model, for various batchsizes. 103
A.4 Comparison of variations to the number of hidden layers and neurons for the best ST model of Set A. 104
A.5 Comparison of different learning rates for the best ST model of Set A. 104
A.6 Evaluation of different activation functions for the best ST model of Set A. 105
A.7 Comparison of different training epochs for the best ST model of Set A 105
A.8 The MAPE of the best STB model of the Set A with the Descent optimizer, for various epochs and batchsizes. 106
A.9 The MAPE of the best STB model of Set A with the ADAM optimizer, for various epochs and batchsizes. 106
A.10 Comparison of the Descent and ADAM optimizer of the best STB model, for various batchsizes. 107
A.11 Comparison of variations to the number of hidden layers and neurons for the best STB model of Set A. 107
A.12 Comparison of different learning rates for the best STB model of Set A. 108
A.13 Evaluation of different activation functions for the best STB model of Set A. 108

A.14 Comparison of different training epochs for the best STB model of Set A 109
A.15 The MAPE of the best STLB model of the Set A with the Descent optimizer, for various epochs and batchsizes. 109
A.16 The MAPE of the best STLB model of Set A with the ADAM optimizer, for various epochs and batchsizes. 110
A.17 Comparison of the Descent and ADAM optimizer of the best STLB model, for various batchsizes. 110
A.18 Comparison of variations to the number of hidden layers and neurons for the best STLB model of Set A. 111
A.19 Comparison of different learning rates for the best STLB model of Set A. 111
A.20 Evaluation of different activation functions for the best STLB model of Set A. 112
A.21 Comparison of different training epochs for the best STLB model of Set A 112

B.1 The MAPE of the best ST model of Set B with the ADAM optimizer, for various batchsizes. 113
B.2 Comparison of variations to the number of hidden layers and neurons for the best ST model of Set B. 114
B.3 Comparison of different training epochs for the best ST model of Set B 114

List of Tables

3.1 Parameters of the 9 Steady States 47
3.2 Engine Specifications 53

4.1 Comparison of the best models for each combination of input variables - Set A 72
4.2 Hyperparameters of the best model, for ST input - Set A . . . 73
4.3 Error metrics of the best model, for ST input - Set A 73
4.4 Hyperparameters of the best model, for STL input - Set A . . 74
4.5 Error metrics of the best model, for STL input - Set A 75
4.6 Hyperparameters of the best model, for STB input - Set A . . 76
4.7 Error metrics of the best model, for STB input - Set A 76
4.8 Hyperparameters of the best model, for STLB input - Set A . 77
4.9 Error metrics of the best model, for STLB input - Set A . . . 77
4.10 Comparison of the best models for each combination of input variables - Set B 84
4.11 Hyperparameters of the best model, for ST input - Set B . . . 85
4.12 Error metrics of the best model, for ST input - Set B 85
4.13 Hyperparameters of the best model, for STL input - Set B . . 86
4.14 Error metrics of the best model, for STL input - Set B 87

Contents

1 Introduction **10**
- 1.1 Thesis Objective . . . 10
- 1.2 Literature Review . . . 11
- 1.3 Thesis Structure . . . 13

2 Theoretical Background **14**
- 2.1 Machine Learning . . . 14
  - 2.1.1 History of Machine Learning . . . 15
  - 2.1.2 Related Fields . . . 19
  - 2.1.3 Categories of Machine Learning . . . 21
  - 2.1.4 Models . . . 23
- 2.2 Neural Networks Overview . . . 28
  - 2.2.1 Linear Basis Function Models . . . 28
  - 2.2.2 Feed-Forward Neural Networks . . . 30
  - 2.2.3 Neural Network Training . . . 33
- 2.3 Four-Stroke Diesel Engine Operating Parameters . . . 41
  - 2.3.1 Brake-specific Fuel Consumption - BSFC . . . 42
  - 2.3.2 Air-fuel Equivalence Ratio - Lambda (λ) . . . 43
- 2.4 Programming Tools . . . 44

3 Model Design **45**
- 3.1 Data Overview . . . 46
  - 3.1.1 First Set of Data - Dataset A . . . 46
  - 3.1.2 Second Set of Data - Dataset B . . . 51
  - 3.1.3 Datasets Comparison . . . 56
- 3.2 Input Parameters . . . 58
- 3.3 Data Preparation . . . 59
- 3.4 Model Hyperparameters . . . 61
  - 3.4.1 Number of Hidden Layers . . . 61
  - 3.4.2 Number of Neurons per Layer . . . 61
  - 3.4.3 Activation Functions . . . 62

3.4.4 Optimization Algorithms 66
3.4.5 Error Function 67
3.4.6 Learning Rate 67
3.4.7 Batch size and Epochs 68
3.4.8 Model Accuracy Metrics 69
3.4.9 Model Nomenclature 70

4 Results and Discussion 71
4.1 Model Set A 72
4.1.1 Best performing networks, per Set of Inputs 72
4.1.2 Model Set A Summary 78
4.2 Model Set B 84
4.2.1 Best performing networks, per Set of Inputs 84
4.2.2 Model Set B Summary 87

5 Conclusions and Future Work 91
5.1 Conclusions 91
5.2 Future Work 93

A Model Set A Diagrams 102
A.1 Speed-Torque (ST) Models 102
A.2 Speed-Torque-BSFC (STB) Models 106
A.3 Speed-Torque-Lambda-BSFC (STLB) Models 109

B Model Set B Diagrams 113
B.1 Speed-Torque (ST) Models 113

Chapter 1

Introduction

1.1 Thesis Objective

In recent years, the great strides in the fields of engineering and computer science has allowed for more complex, sophisticated tools for controlling the various parameters during the power producing cycles of an internal combustion engine. The need for more environmentally conscious and productive engines has lead to the search for ways of measuring the in-cylinder conditions, for better control over the combustion process and for general evaluation of the engine operation. However, most of the methods that deal with hard to measure parameters, like the in-cylinder pressure or temperature, end up being not practical or economically feasible.

This thesis aims to tackle the issue of accurate in-cylinder pressure prediction, and more specifically of the peak cylinder pressure, of a four-stroke marine diesel engine. Pressure in general is an important factor in the operation of an internal combustion engine, as the fundamental concept of such engines is the production of work via the burning of air and fuel mixture and the subsequent high-pressure released gases that act upon the piston. Peak pressure, additionally, is a significant factor that contributes to the stress of several mechanical components, including the cylinder head, the piston crown and the piston rings.

Measurement of pressure by conventional means is usually carried out by piezoelectric pressure sensors, strong enough to withstand the extreme temperatures and pressures, which are installed inside the cylinder, on its head. Its installation, thus, entails the drilling of the cylinder; a quite expensive and often impractical process, as it requires the engine to be taken out of commission until the sensor is placed.

Other, indirect, measurement methods have also been developed to assess

the in-cylinder pressure. One method uses the acoustic emissions produced during the combustion phase to construct the engine cylinder pressure waveform [1], while a different one recreates the pressure waveform by processing the engine speed signals, taking advantage of the experimentally proven linear correlation between these two signals [2].

Having taken into consideration the advances in the area of artificial intelligence and machine learning over the last years, this thesis proposes an alternative solution; artificial neural networks could be utilized in order to make accurate predictions of the requested pressure. These models would make their predictions by taking as input other, easy and inexpensive to measure parameters, like the engine working speed or lambda coefficient.

So, in the premises of this thesis, the feasibility of such models is explored. Multiple types of neural networks with varying hyperparameters are constructed and consequently trained on experimental data from two different engines, both being of four-stroke, diesel type, and then their efficiency of prediction is tested against new sets of input data.

1.2 Literature Review

Before the development and training of the neural networks could be set into motion, a solid understanding of the fundamentals was required. Firstly, the basics of internal combustion engines and the various parameters that affect the power production cycle were revisited, as presented by [3]. Then, a thorough study of the intricacies of machine learning was carried out, with emphasis on artificial neural networks and deep learning models, both in theoretical level and in practical applications. The bulk of the information concerning the theoretical approach and mathematics groundwork of the networks stemmed from [4], as well as [5] and [6]. Regarding the practical applications of neural networks in the context of engine parameters prediction, a number of scientific papers were examined. One paper [7] dealt with the prediction of the maximum piston temperature in a dual-fuel engine using Support Vector Regression, a machine learning technique. For input, the engine speed, the NOx emissions, and the excess air coefficient were used in various combinations; after testing, it was determined that the models that utilized all of the above parameters had the superior accuracy. The outcome was an algorithm that predicted the requested peak temperature with an average absolute error of 1.98^oC and a maximum error of 6.63^oC. Another study [8] aimed to predict the brake specific fuel consumption, effective power and exhaust temperature of a gasoline engine through three different artificial neural network models. For all three models, the engine

speed, engine torque, fuel flow rate, intake manifold mean temperature and cooling water inlet temperature were used as the input parameters. The networks had a singular hidden layer; for the deduction of the best number of neurons, values of 3 to 15 were tested, while two different learning algorithms were employed: the scaled conjugate gradient (SCG) algorithm and the Levenberg–Marquardt (LM) algorithm. The results indicated that the better of the two algorithms was the SCG one. In regards to the number of neurons, the ideal number for the BSFC and the exhaust temperature models was 7, with the one for the effective power model being 5. In any case, all models when evaluated against the test data, presented an average error of less than 2.7% and a coefficient of determination greater than 0.99, proving that the training was indeed successful. A similar paper [9] used yet again a single hidden layered artificial neural network, to produce the break specific fuel consumption (BSFC), carbon monoxide (CO) and unburned hydrocarbon (HC) exhaust emissions, and AFR air-fuel ratio (AFR) of a four stroke spark ignition engine that operated using two different fuels: methanol and gasoline. As inputs, the fuel type, engine speed, engine torque, and fuel flow was used; regarding the fuel type, for methanol the arbitrary value of 1 was given and for gasoline the value of 2. The logistic sigmoid function was chosen as the activation function, while for the number of neurons of the hidden layer yet again multiple values were tested, from 5 to 15. Similarly, four different learning algorithms were tested: Quasi-Newton back propagation (BFGS), Levenberge-Marquardt (LM) learning algorithm, the resilient back propagation (RP) and scaled conjugate gradient learning algorithm (SCG). For each of the four distinct models, the outcomes showed that different parameters were optimal: for the CO model the LM algorithm and 4-7-1 neurons, for the HC model the RP algorithm and 4-14-1 neurons, for the BSFC model the SCG algorithm and 4-7-1 neurons, and finally for the AFR model the BFGS algorithm and 4-11-1 neurons. The networks that predicted the BSFC, HC and AFR parameters gave a coefficient of determination greater than 0.99; the model that predicted the carbon monoxide emissions on the other hand gave a smaller coefficient of 0.978.

Regarding studies that focused on pressure prediction, one of them [10] utilized Extreme Learning Machine (ELM) models for predicting two types of pressure: the pressure inside the combustion chamber and the pressure at the exit of the intake chamber of a spark ignition engine. An ELM model is a type of feed forward artificial neural network with a single hidden layer, the input weights and hidden biases of which are assigned arbitrarily and subsequently never updated, unlike in most other standard machine learning algorithms, having as a result significantly faster training times. Inputs for the models consisted of the crankshaft angle and engine speed; they included

five different engine speeds and crankshaft angles from -360° to 360°. The optimal model hyperparameters were determined through a Biogeography-based optimization (BBO) algorithm, a metaheuristic optimizer of functions that is based on the spread of a population of living creatures in an environment. The results for the in-cylinder pressure predictions produced a R^2 value of greater than 0.993, indicating that the model had adequate accuracy and good consistency with the experimental values. Finally, a different, and more closely related to the methods and objective of this thesis, paper [11], was one that aimed to predict the in-cylinder pressure of a Homogeneous Charge Compression Ignition (HCCI) engine using deep neural networks. The architecture of the network consisted of 4 hidden layers with the number of neurons being 100, 150, 120 and 120 respectively and having the rectified linear unit (ReLU) as activation function, while for input the crank angle and air excess coefficient values were used. The resulted model achieved prediction of 99.84% accuracy, which faired better in comparison to "shallow" neural networks. The methodology, through which the type and hyperparameters of the networks of this thesis were chosen, was influenced by the conclusions of the aforementioned paper; of course many other designs and options were explored during the process of model building and training.

1.3 Thesis Structure

In Chapter 2 the theoretical groundwork of this thesis is laid; that includes a synopsis of the machine learning as a whole, the mathematical theory of the neural networks and a brief description of a four stroke diesel engine. In Chapter 3 the analysis of the model design process takes place, including an overview of the acquired training datasets. In Chapter 4 the results are presented and the trained networks are evaluated, while finally in Chapter 5 the conclusions of this thesis are presented and some ideas for future studies are propositioned.

Chapter 2

Theoretical Background

2.1 Machine Learning

In the context of machines, learning is considered when a machine changes its structure, program, or data, based on external information, in a way that enhances its performance[12]. More specifically, the term *Machine Learning* describes a series of computational methods that, through experience, improve themselves, providing better and more accurate results[13]. The aforementioned experience is being provided through a set of data, typically called the *training data*, which they use to train on and build a prediction model. After sufficient training, such algorithms are tested against new datasets, called *test data*, and the results are evaluated; should they have learned from the given example dataset, they will perform with high accuracy. This ability is known as *generalization* and is broadly regarded as the fundamental objective of any prediction algorithm[13] [4].

An outline of the learning process is shown in the following diagrams 2.1 and 2.2 [14]. Note that the model in not known before the training stage, but instead is developed through the introduction of the training data.

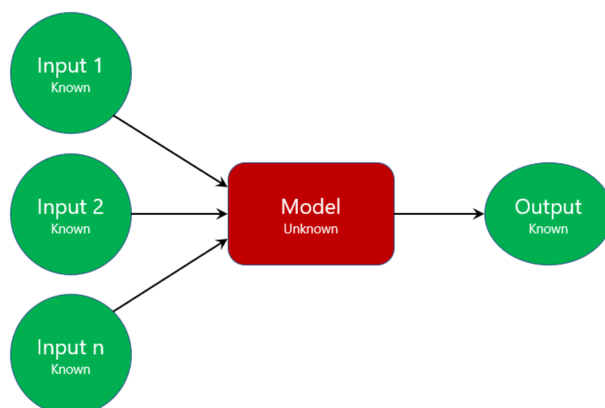

Figure 2.1: A Machine Learning model during the training stage

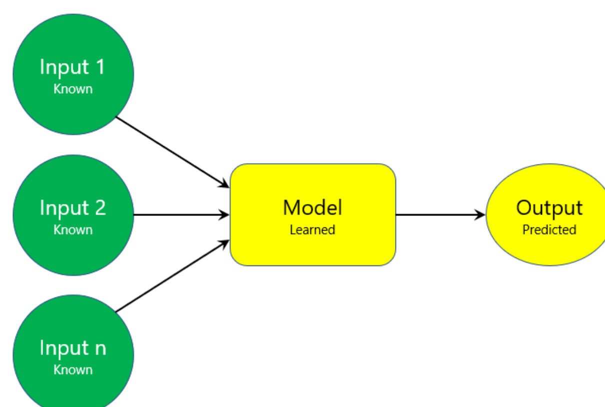

Figure 2.2: A Machine Learning model after the training stage, during the validation stage

2.1.1 History of Machine Learning

In 1950, Alan Mathison Turing, a British mathematician and logician[15], in his paper *"Computing Machinery and Intelligence"*[16] introduced the *Imitation Game*, nowadays referred to as the *Turing test*, a test which aimed to challenge a machine's ability to think and judge whether it can become indistinguishable from a human person[17]. In the same paper, he also presented the concept of a *learning machine*. He proposed that instead of trying

to create a machine that imitates an adult human mind, a long and time-consuming process due to the complexity of the matter, it would be more fruitful to produce a program that simulates a child mind, a far simpler and easier task. Then, the *child-programme* would have to be properly educated and allowed to gain experience in order to grow into the the final machine, with intellectual power that can compete with humans[16].

Later, in 1952, the term *Machine Learning* formally appeared; it was first introduced by Arthur Samuel, an American pioneer in the field of artificial intelligence[18][19]. He developed a computer algorithm for playing checkers that, in a relatively short period of time and with minimal input and directions, not only learned how to play the game, but also outperformed any average person[19]. In the same paper, Samuel recognized and pointed out that the basic principles and approach behind the checkers algorithm could be utilized to build more complex programs for solving real life issues.

Samuel's program used what later evolved into the *minimax algorithm*, alongside an *alpha-beta pruning* optimization[20]. The minimax algorithm ensured that the action taken by the computer would be the optimal one, by minimizing the potential loss of the worst case scenario, while the alpha-beta pruning sped up the process by eliminating any move that had been found to be worse than a previously examined one[21].

In 1957 Frank Rosenblatt, an American psychologist with contributions to the field of artificial intelligence, created the first artificial neural network; an algorithm that himself called the *Perceptron*[22]. Similar to its biological counterpart, it received through a series of input neurons a set of data, and via weighted connections it activated an output neuron[23]. It started as a computer program, but later a hardware prototype named *Mark I Perceptron* was built and tested[4]. In 1961 Rosenblatt published a paper titled *"Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms"*, where he presented multilayer and cross-coupled perceptrons[24]. One such design is depicted in Figure 2.3[22].

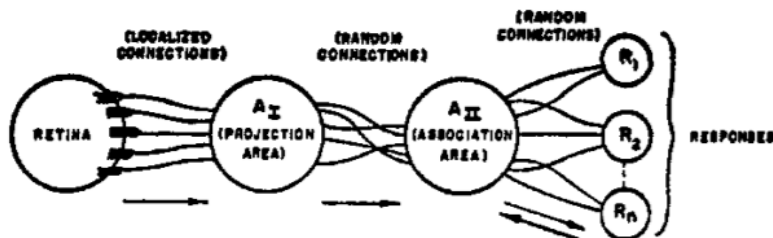

Figure 2.3: The Perceptron, as designed by Rosenblatt.

In 1967 two Ukrainian soviet scientists Alexey Ivakhnenko and Valentin Lapa developed what is considered the first feedforward multilayer perceptron, by training neural networks with the Group Method of Data Handling[25], an inductive learning algorithm suited for complex models[26]. The creation of the 8-layered neural network was marked as the first *Deep Learning* network[25] and as such, Ivakhnenko was widely regarded as the father of Deep Learning.

In spite of the progress of the previous years, the following decades saw heavy fluctuations in the interest of the scientific community in machine learning, as the focus shifted more towards artificial intelligence, which caused a schism between the two fields[27]. Machine learning returned rejuvenated, as a separate branch in 1990[27].

During the years of recession, however, several achievements in the field were made, leading to a significant milestone in 1997 with the *Deep Blue*, a chess playing supercomputer.

In 1979 at the Stanford University, an autonomously moving vehicle, the *Stanford Cart*, successfully navigated an obstacle-filled room on its own. It had been in development since 1960 and initially started as a remote operated vehicle[28].

In 1987 the NETtalk, a parallel neural network system that could pronounce English text, was developed by Terrence J. Sejnowski and Charles R. Rosenberg[29]. The neural network utilized 3 layers of processing nodes; the input information was fed to the first layer, then moved through weighted connections to the second layer and finally reached the third and final output layer. A depiction of the NETtalk system is presented in Figure 2.4 [29].

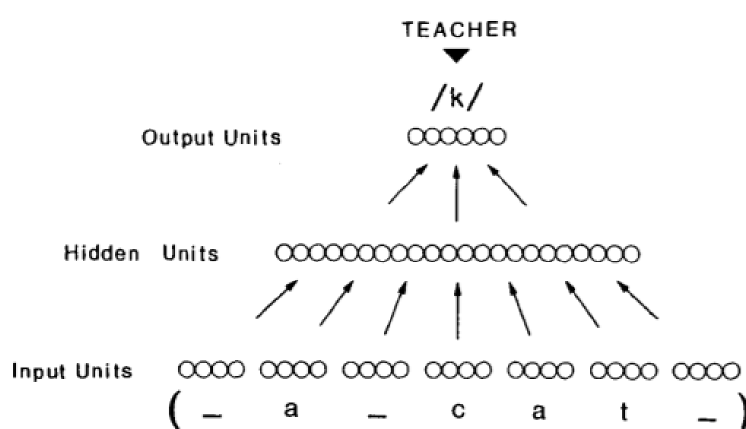

Figure 2.4: The NETtalk system schematic.

Finally in 1997, the *Deep Blue* computer, developed by IBM, managed to win a game of chess against the then reigning world champion Garry Kasparov[30]. Kasparov had previously beaten *Deep Thought*, a predecessor to the winning super-computer in 1994, as well as *Deep Blue* itself two years later in 1996[31]. This victory was of great importance, not only for the computer science, but for several other fields as well, because it helped push the processing abilities of computers and contributed to the better understanding of the complexity of massively parallel computations[32].

In the 21st century, due to the advance of computer hardware technology, more complex neural networks were feasible, leading to a boom in artificial intelligence and machine learning.

In 2006 Fei-Fei Li, an American computer scientist, started working on *ImageNet*, a large scale visual database used for training complex object recognition models[33]. The dataset was presented publicly in 2009 at the *Conference on Computer Vision and Pattern Recognition* in Florida[34]. The motivation behind ImageNet was twofold; it aimed to offer both a large amount of image data for training sets and a high quality evaluation benchmark for object recognition and categorization.[33].

In 2010 Microsoft released *Kinect* for the Xbox 360 gaming console, a sensor that allowed players to operate the machine without a controller, but instead with hand gestures and body movement. It was equipped with RGB cameras and infrared sensors that allowed it to construct a depth image of a person via triangulation and subsequently detect and categorize the person's limbs using a decision tree[35].

One year later, in 2011, Google founded *Google Brain*, a research team that focuses on projects in the fields of artificial intelligence and machine learning. Within a year, the team had developed a software for visual recognition, that could identify images of cats[36]

In 2014 Facebook made a breakthrough in the area of facial recognition with its deep learning system called *Deep Face*. The software could verify whether a picture showed the same person, regardless of differentiations in position or lighting, with 97.25% accuracy, performance comparable to a human's[37]. The process of identification included a forward-facing reconstruction of the input image using a generic 3D face model and the subsequent numerical analysis of the correctly oriented image using deep neural networks[37].

In 2016, DeepMind Technologies, a British artificial intelligence company-subsidiary of Google and later of Alphabet Inc., using its computer software *AlphaGo* beat Lee Sedol, a 9-dan professional player, in a game of Go[38], a feat viewed as a decade ahead of its time[39]. Go is a board game that was considered as the biggest challenge for board gaming artificial intelligence,

due to its massive pool of legal board positions and difficulty in evaluating moves, despite its relatively simple set of rules[40].

The software contained deep neural networks that were trained by a combination of supervised learning from human expert games and reinforcement learning from thousands of self-play games, which the program simulated via a Monte-Carlo tree search algorithm[40].

2.1.2 Related Fields

Machine learning is often associated with the fields of artificial intelligence, deep learning and statistical science. While they all share common areas,with some being subfields of others, they are still considered as distinct entities with different goals and tools. It is therefore important that clear definitions of these terms are to be presented and their differences highlighted, so as to avoid any confusion.

Figure 2.5 [41] presents the relations between these four fields.

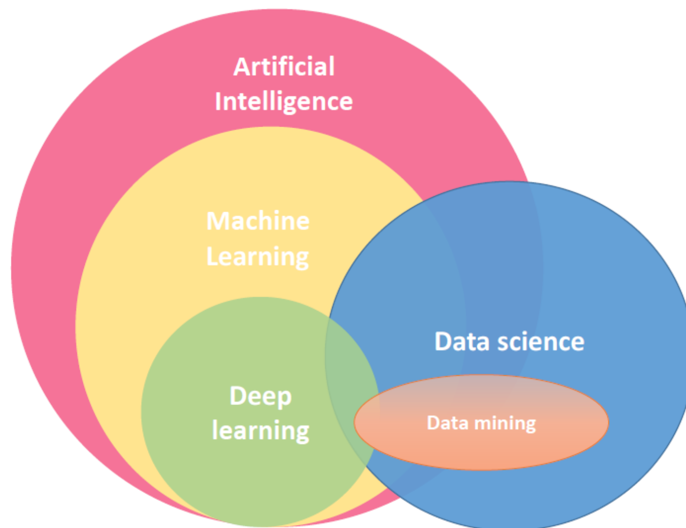

Figure 2.5: Venn diagram of the machine learning related fields of science.

Artificial Intelligence

Regarding artificial intelligence, machine learning in its conception started as branch of that field[27]. However, during the decades of 1970-80, new trends in the scientific community caused a rift between the two fields[6]. People started focusing more on intelligent systems, developing programs aimed at

tackling more complex problems related with human cognitive abilities like multi-step reasoning, heuristic problem solving and language understanding while at that point, the machine learning community emphasized their work on classification and regression[27]. The field later re-emerged as a separate area of study, with its objective shifting towards the development of probabilistic models from data[6].

The key difference between the two concepts lies in their definition. Artificial intelligence is described as the ability of a system to accurately interpret external stimuli, learn from them and then use this information to adapt appropriately, in order to achieve its goals[42]. Therefore, it is inferred that the system interacts with its environment while trying to follow its objective. In contrast, machine learning, as previously defined, is generally restricted to passive analysis of information and construction of a model that fits the input data, with the purpose of utilizing the model to accurately predict future datasets.

It is generally regarded that machine learning is a subset of artificial intelligence[43][5][44].

Deep Learning

Deep Learning is a sub field of machine learning that passes information through multiple intermediate processing layers. The defining characteristic that distinguishes deep from "shallow" learning is the number of hidden layers, however the exact amount differs from definition to definition; some argue that at least two hidden layers are required[43], while others three or more[45][46].

These extra levels enable the model to better handle complex data structures with multiple levels of abstraction[46], making deep learning an advantageous option for handling image, speech and text recognition.

Data Science - Data Mining

The term data science describes the entire process of data handling, from acquisition and analysis, to interpretation and utilization[41]. Since machine learning algorithms depend strongly upon the quality of the input dataset, this field of study is tied to the fields of data science and computational statistics[13], at the same time borrowing from and lending to them tools and techniques. Of course, each of these fields share a different objective; statistics, for instance, focus on inference, by building a project-specific probability model, while machine learning concentrates on prediction, by applying and training general-purpose learning algorithms on datasets of considerable size

and complexity, in order to find generalizable patterns[47].

Data mining constitutes a part of the broader family of data science. This procedure specifically involves combing sets of data for useful patterns, often unearthing hidden pieces of information[41]. As a result, data mining relies heavily on machine learning algorithms[41].

2.1.3 Categories of Machine Learning

Machine learning can generally be divided in three types; *supervised learning*, *unsupervised learning* and *reinforcement learning*. In practise, different techniques that don't fall under the aforementioned categories are sometimes implemented as well; these will be briefly touched upon towards the end of this section.

Supervised learning

In supervised training, the training dataset consists of variables which are labelled and with corresponding target values; in essence they are examples of the desired correlation that the model is tasked to learn[6]. The algorithm then analyses the input-output pairs, finds the relation between the two and formulates an appropriate function that fits the data.

Subtypes of supervised learning include *classification* and *regression*; the former is used for problems where the objective is to categorize the input data into a finite amount of different classes and the latter for ones that predict continuous variables[4].

Examples of this type of machine learning include recognition of numerical characters from hand-drawn digits[4], estimation of the price of a house using as input data its area, number of rooms and floors, whether it is equipped with furnitures or appliances, etc.[43], or weather prediction by examining data like the temperature, humidity, wind and so forth.

Unsupervised learning

While in supervised learning the train data included the corresponding output values, in unsupervised learning there are no such values; only the input variables are given[4]. In these types of learning process, the objective is for the algorithm to find the regularities in the data and uncover hidden patterns[5].

Generally, two subtypes of unsupervised learning can be discerned, *density estimation* and *data clustering*. Data estimation is used to discover the

distribution of the input data, while clustering to detect useful groups of variables, that share similar characteristics[4].

Typical uses of unsupervised learning consist of finding communities of people with similar interests in a social media algorithm example, or for creating statistics of past clients for a company[5].

This category is also used for projecting a high-dimension data set to one with fewer dimensions, while retaining some of its characteristics, a process called *dimensionality reduction*[13]. It is usually applied in image compression problems, were the goal is to reduce the amount of pixels in order to save storage space, while keeping the image as close to the original as possible[5].

Reinforcement learning

Lastly, by the method of reinforcement learning, the algorithm learns to decide which action is the most profitable one, in order to receive the best outcome[4]. Like unsupervised learning, the training dataset does not include the desired output, instead the algorithm is tasked with figuring out the optimal course of action through a process of reward and punishment[6]. This technique is unique in that, in some of it stages, the algorithm interacts with its environment[4].

Reinforcement learning is commonly associated with the *Markov Decision Processes*, a method of environment simulation. Based on this system, the environment is being represented by a number of states that can be interacted with and altered; changes to these states affect the simulated space[48]. The algorithm can utilize this simulation to hone its decision-making, by running trials and self-test.

One of the more well-known application of this particular subtype is in game playing[4]. Backgammon, chess and checkers are examples of boardgames that lend themselves to this kind of machine learning, due to their varying degrees of strategical complexity, in spite of having simple rules. Another area that takes advantage of reinforcement learning is automated navigation[5].

Other types

Bellow, a few more different types of machine learning are listed. These ones do not quite fit in the three previous general categories, thus the need to be presented separately. The list should not be considered all-inclusive.

- *Semi-supervised learning.* This type falls in between supervised and unsupervised learning. The training dataset consists of mostly unlabelled data, with a few labelled examples, which are used to categorize the unlabelled ones, before fitting a model[43].

- *Transductive inference.* Similarly to the semi-supervised learning, the algorithm receives a mix of labelled and unlabelled data, with the specific objective of only predicting the missing labels[13].
- *Representation learning.* It is mostly used for intermediate process of data. Its purpose is to extract useful information from a series of raw data, later used for building other prediction algorithms[49].
- *On-line learning.* In this type of learning, the input data is being provided sequentially, while the algorithm passes through a number of train and test loops. The general aim of such process is to either minimize the total loss across all rounds of evaluation, or minimize the difference between the cumulated past loss and best theoretical result[13].
- *Active learning.* Here, the algorithm begins the learning process with a small amount of labelled data, and later requests for more labelled points by submitting a query to an *oracle*, like a human user for example. This unique algorithm is utilized for instances were unlabelled data is abundant, but labelling it is expensive or time-consuming[50].

2.1.4 Models

For the implementation of a machine learning system, a number of distinct models are at one's disposal. No specific method provides the best all around results; each algorithm has its strength and weaknesses, and different tasks require different approaches. Below, an enumeration of the most notable machine learning models is presented; once again, this list should not be considered comprehensive.

Artificial Neural Networks

Artificial neural networks are mathematical models that, by mimicking the behaviour of a real, biological neural system (i.e. a brain), uncover patterns between a set of input and output data[51]. They consist of at least 2 layers of processing units, the input and output ones, and a number of intermediate, hidden layers. These processing units are called *neurons*.

Each neuron receives input data from a singular or multiple previous neurons; each information stream is first multiplied with a weight, afterwards all of them are aggregated together and the end result is passed through an *activation* function. This function allows the network to work non-linearities that may exist in the task into the model[52] and has a significant impact

on the accuracy and effectiveness of its prediction. The processed data then exits the neuron as its output and proceeds to the next layers.

A depiction of a typical neural network is shown in Figure 2.6[53], while the data flow through a processing unit of a network is depicted in Figure 2.7[52].

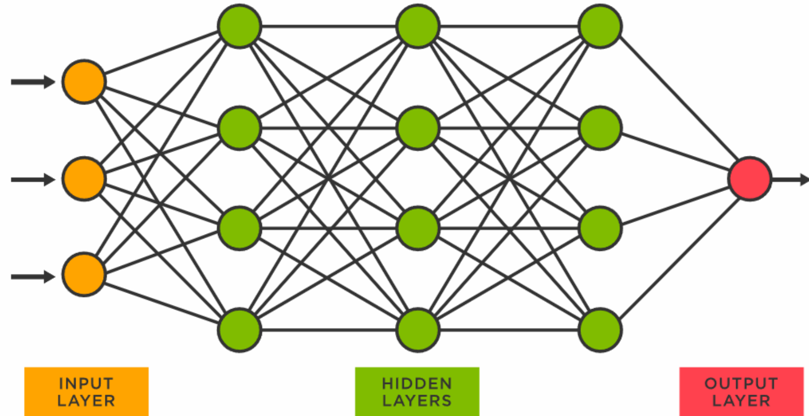

Figure 2.6: A neural network diagram.

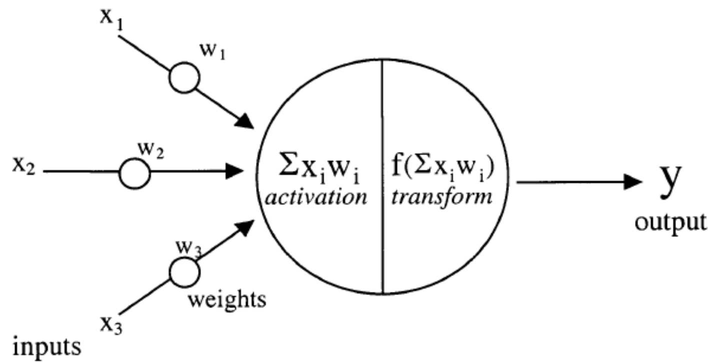

Figure 2.7: Analysis of a singular neuron.

During the training phase of the network, the weights are being constantly updated in order to reduce the prediction error of the model to a desired level. The type of optimiser that is being used for such purpose, as well as the *learning rate*, meaning the speed with which the optimisation takes place, are important model-dependant and task-specific parameters that greatly affect the success of the trained network.

Since neural networks were the chosen machine learning model for the purpose of engine parameters prediction of this diploma thesis. To this end, the underlying mathematics of the networks, the specific activation functions and optimizers used, as well as any other important aspect of the algorithm are discussed in greater detail in the following Section 2.2.

Support Vector Machines

Support vector machines (SVMs) are a type of non-probabilistic linear algorithm [54], used mainly for problems of classification and linear regression[55].

Given two sets of data, the SVM is tasked to produce the optimal hyperplane that successfully separates them. Since there is an infinite number of planes that achieve the same result, the extra notation "optimal" implies the existence of a best one. Indeed, the algorithm predicts the hyperplane that bisects with the biggest margin the space between two parallel hyperplanes that are marginally close to the two sets of points, without splitting any number of objects from its corresponding group[54]. The model then uses that plane to correctly categorize any new points to one of the two groups.

This procedure is presented in Figure 2.8[56].

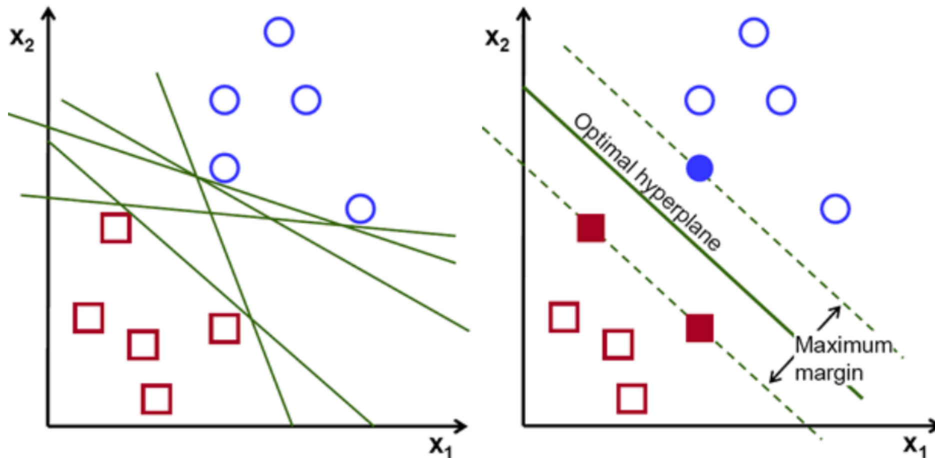

Figure 2.8: Example of a support vector machine creating the optimal hyperplane between two distinct groups of data.

The above process works for linearly separable sets of data; if the objects are non-linearly separable, then they have to first be transformed to a higher dimension space using the kernel trick[54].

Decision Trees

Decision trees are classifier-type models[57] that receive an input data, process it through a series of internal assessments and finally make an appropriate decision, in essence producing a singular result[6].

They consist of the first node, from which the algorithm begins, called the *root*, a number of intermediate nodes that evaluate the information while it passes through them and split the out-coming paths, called *decision nodes* and the end-nodes that release the output, called *leaves*. The structures that connect the nodes are called *branches*.

An example of a decision tree structure is provided in Figure 2.9 [58].

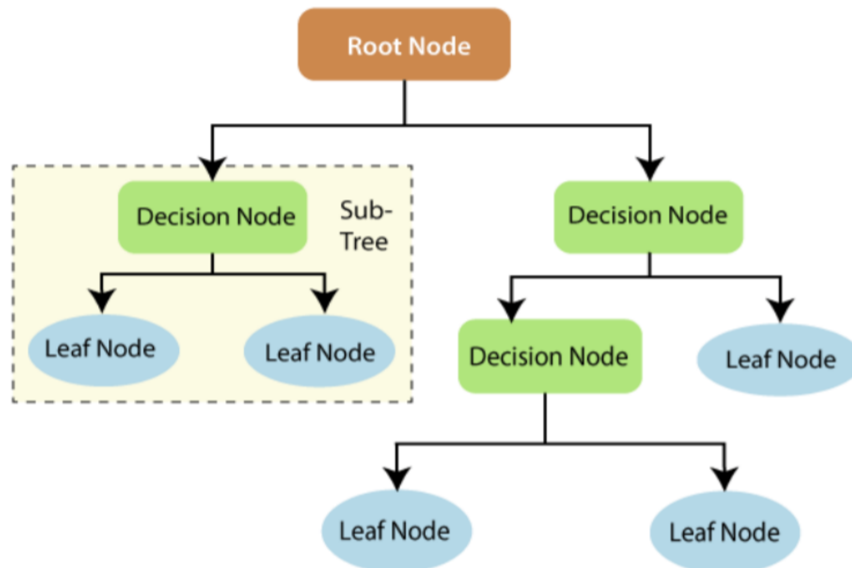

Figure 2.9: Example of a decision tree.

Bayesian Networks

A Bayesian network as a structure describes and presents graphically the probabilistic relationship of a group of dependent or independent variables[59], in essence being a combination of graph theory and the Bayesian inference Theorem[60].

The variables are depicted through a directed acyclic graph in the form of nodes; should a pair of variables be dependant of one another, then the graph includes a connecting path with an appropriate direction that links them, otherwise the nodes are separated. Each node is also equipped with a corresponding conditional probability distribution, that covers every possible scenario regarding the outcome of the earlier nodes.

In the following Figure 2.10 [60] the ”wet grass problem” is being analysed via a Bayesian network.

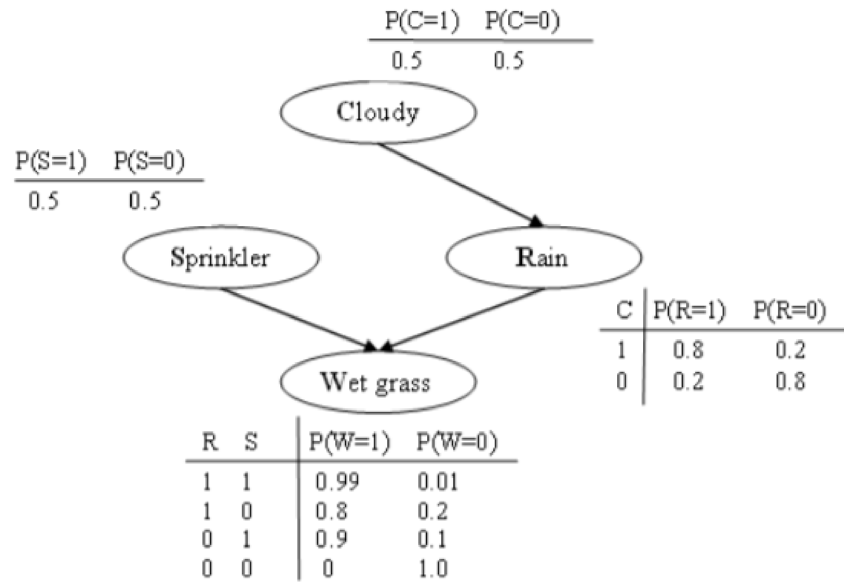

Figure 2.10: Example of a Bayesian network.

2.2 Neural Networks Overview

In this section, a brief synopsis of the mathematics behind neural network models is given. A linear model in a regression example is firstly presented, so as to introduce some basic definitions in a simple environment. Afterwards, the concepts are extended into a neural network of feed forward type, which, being the core of this thesis, is further analysed. This section is based on the relevant chapter of the book *Pattern recognition and Machine Learning* of C. Bishop [4].

2.2.1 Linear Basis Function Models

For describing a simple model that has linear basis functions, terms that will be explained further in the section, the basic problem of regression is introduced. In this type of task, the objective is to predict the target value t given a D-sized input vector x. The model receives a data set that comprises of N observations $\{x_n\}$, where $x_n = x_1, ..., x_N$ and their corresponding target values $\{t_n\}$, again with $t_n = t_1, ..., t_N$. It then uses this dataset to train, meaning that it constructs a mathematical formula that fits the input data, in order to then predict the t value of a new x input.

The most basic linear regression model can be described as

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

where $x = (x_1, ..., x_D)^T$ are the input variables and $w = (w_1, ..., w_D)$ the model parameters. The important aspect of this model is that, not only is it a linear function of the parameters w_i, but also of the input variables x_i. This linear relation with the input is however problematic, as it limits its capabilities. This is overcome by utilizing a linear combination of non-linear functions of the input variables, as in

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

where the $\phi_j(x)$ are called the *basis functions*. With the maximum value of the j being $M-1$, the total number of parameters are M. The w_0 parameter is called a *bias* parameter and helps introduce to the model any fixed offset in the data. By defining a dummy basis function $\phi_0(x) = 1$, the above equation 2.2 can be neatly written as

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

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

Thus, the input data given via the vector x is beforehand transformed through the basis functions into terms of $\phi_j(x)$, consequently allowing the $y(x, w)$ function to develop a non-linear relation to x. However, such models are still regarded as linear, due to them being a linear function of the w parameters.

Regarding the basis functions, a number of different types are at ones disposal, each with varying degrees of competency and limitations.

One example is the *polynomial* basis function, defined as

$$\phi_j(x) = x^j \tag{2.4}$$

where the function consists of powers of the input variable x. Here, the main issue that arises is that they are global functions of the x and as a result changing one area of the input has an effect on all the others. This can be overcome by splitting the input space into different regions, each with their own separate polynomial function, thus producing basis functions called *spline* functions.

A different example is the *Gaussian* function

$$\phi_j(x) = exp\left\{-\frac{(x - \mu_j)^2}{2s^2}\right\} \tag{2.5}$$

where the locations and spatial scales of the basis functions are dependant on the μ_j and s parameters respectively. Despite their naming, these functions do not have to be necessarily connected to any form of probability.

Finally, another common choice for basis functions are functions that are related to the logistic sigmoid one, given by the equation

$$\sigma(a) = \frac{1}{1 + exp(-a)}. \tag{2.6}$$

This group includes the *sigmoidal* basis function, as defined

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right) \tag{2.7}$$

as well as the *tanh* function, which is related to the logistic sigmoidal one through the equation

$$tanh(a) = 2\sigma(a) - 1. \tag{2.8}$$

While the parameter linearity of this type of models helps in making their analysis considerably simpler, it also greatly limits their capabilities. More specifically, the basis functions $\phi_j(x)$ are selected before the observation of the

input data, thus they do not adapt to it appropriately. Moreover, when the dimension D of the input data increases, the number of basis function needed also increases but with a significantly steeper rate, often exponentially. This issue is known as the *curse of dimensionality*. As a result, different, non-linear models are needed for practical problem-solving tasks; one of these complex models are the neural networks.

2.2.2 Feed-Forward Neural Networks

The above discussed linear models, as given by the equation 2.3, is now written as

$$y(x, w) = f\left(\sum_{j=1}^{M} w_j \phi_j(x)\right) \tag{2.9}$$

where $f(\cdot)$ is a non-linear activation function. In order to circumvent the short-comings of the linearity of this model, the basis functions $\phi_j(x)$ are to be made dependant on adjustable coefficients that, alongside the w_j parameters, are adapted during the training phase of the model.

This process is used for the basis functions of a typical neural networks; their functions are a non-linear function of a linear combination of the inputs, with the linear combination having adjustable parameters.

Having a D-sized input vector $x_1, ..., X_D$, we produce M linear combinations of them, as in

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2.10}$$

where $j = 1, ..., M$ and the (1) superscript denotes that the above calculations belong to the first *(*layer) of the neural network. The $w_j i$ parameters are from here on referred to as *weights* and the $w_j 0$ as *biases*.

The a_j quantities, known as *activations*, are then passed through a non-linear, differentiable function $h(\cdot)$, called *activation* function, producing the output of the basis functions of 2.9

$$z_j = h(a_j). \tag{2.11}$$

The z_j are called *hidden units*. As for the activation function, sigmoidal functions like the logistic sigmoid or the tanh function are common candidates.

The hidden units are then, in turn, linearly combined to produce the K *output unit activations*

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} x_j + w_{k0}^{(2)} \tag{2.12}$$

where $k = 1, ..., K$, with K being the total number of outputs. The (2) superscript denotes that the above process takes place after the one in 2.10, in the second layer of the neural network.

Lastly, in order to get the output values y_k, the output unit activations are transformed via an appropriate activation function. As previously discussed, a number of different activation functions are available, with the most suitable one being task-dependant.

By combining the above processes, and assuming that the network utilizes a logistic sigmoid function for its output, as defined in 2.6, the model can be described as

$$y_k(x, w) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \tag{2.13}$$

where with the term w all the weight and bias parameters are included. As before, by defining a dummy input variable $x_0 = 1$ we can absorb the biases of the first layer to the sum, as

$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i \tag{2.14}$$

and then, subsequently, the second layer biases can also be combined, to produce a more compact writing of the model

$$y_k(x, w) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{2.15}$$

Figure 2.11[4] depicts the architecture of the example neural network, as described by the equations 2.13 or 2.15.

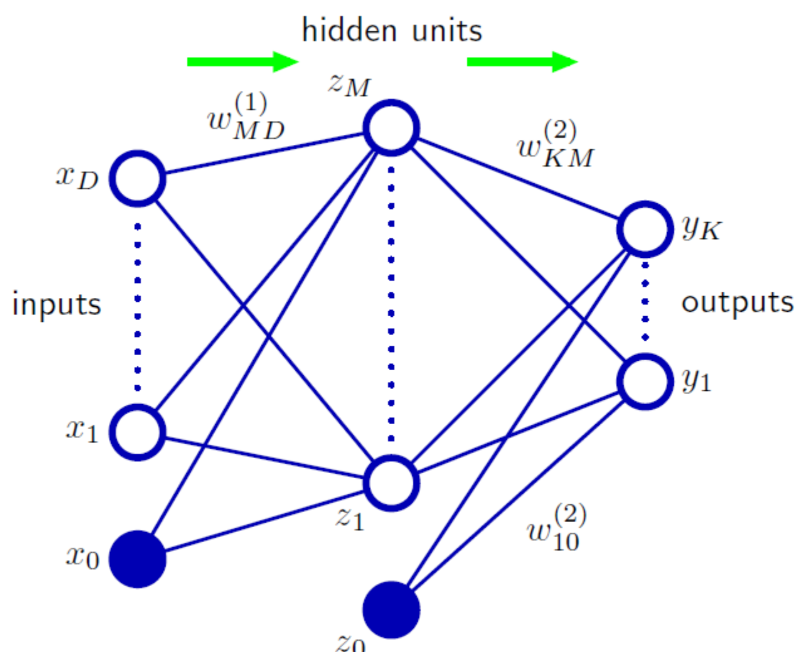

Figure 2.11: Outline of a typical neural network with a singular hidden layer.

One important characteristic of this type of neural network is the fact that the information flow is strictly directed, in that there are no closed loops between the processing units, as shown by the green arrows in Figure 2.11.This type of network is therefore called a *feed-forward neural network*.

At this point in the Chapter, a convention regarding the number of layers that a neural networks has, will be agreed upon. There is often confusion in expressing the amount of layers; the network depicted in Figure 2.11, for example, can be both described as having three layers, amounting to the three distinct rows of processing units, counting the input one as well, but also as having a singular one, counting only the hidden processing units.

From here on, the terminology that will be adhered to is the one that describes the neural network of Figure 2.11 as a two-layered model; number that occurred after counting the number of layers with adjustable weights.

In Figure 2.12 an example of the capabilities of a two-layered network that has three different hidden processing units with the tanh activation function, is presented. The model was given as a prediction target four different functions: (a) $f(x) = x^2$, (b) $f(x) = sin(x)$, (c) $f(x) = |x|$ and (d) $f(x) = H(x)$, where $H(x)$ is the Heaviside step function. The input vector consisted of 50 different data points in the (-1,1) interval, marked as blue dots on the diagram, while the model prediction is marked with a continuous

red line. The three dashed lines represent the three hidden units, exhibiting how the model output is produced by their sum.

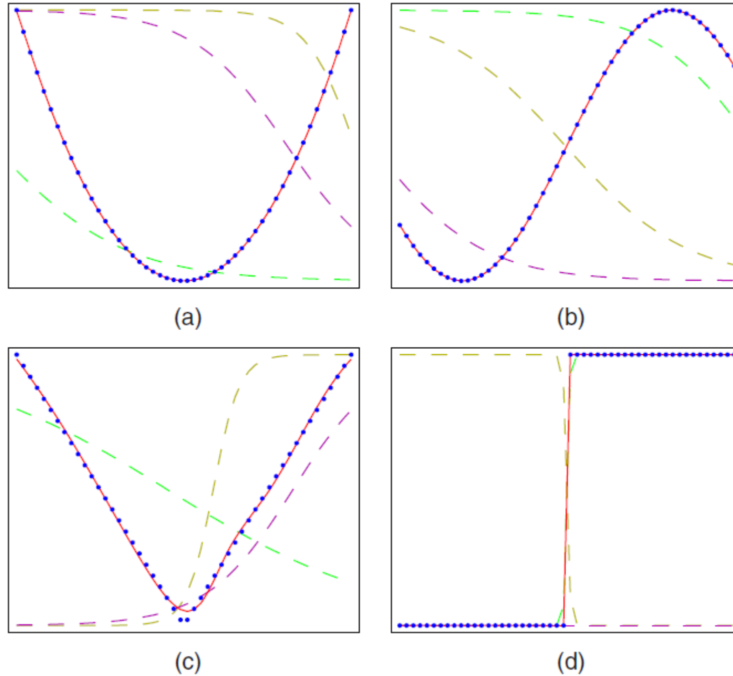

Figure 2.12: Depiction of a two-layered neural network prediction, in four different target functions:(a) $f(x) = x^2$, (b) $f(x) = sin(x)$, (c) $f(x) = |x|$ and (d) $f(x) = H(x)$, where $H(x)$ is the Heaviside step function.

2.2.3 Neural Network Training

The process of training for the neural network corresponds to the process of adapting the network parameters. In order to calculate these weights, a general error function is to be defined and subsequently minimized.

Given the input vector $x_n = \{x_1, ..., N\}$, complemented by their own target values t_n, we choose as the error function the sum of least squares one, which is defined as

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} ||y(x_n, w) - t_n||^2. \tag{2.16}$$

At this point, in order to provide a more general approach to the notion of network training, it is fruitful to give a probabilistic view of the network and its output.

Limiting the scope of this section to problems of regression, we consider a single target variable t that has a Gaussian distribution with an x-dependent mean, which is produced by the neural network, so that

$$p(t|x, w) = \mathcal{N}(t|y(x, w), \beta^{-1}) \tag{2.17}$$

where β is the precision (inverse variance) of the Gaussian noise. For the above conditional distribution, we can assume that the output unit activation function of the neural network is the identity, constructing in this way a network that can predict any continuous function.

Given that $X = \{x_1, ..., x_N\}$ is a set of independent, identically distributed observations, and that $t_N = \{t_1, ...t_N\}$ is their target values, the corresponding likelihood function can be calculated as

$$p(t|X, w, \beta) = \prod_{n=1}^{N} p(t_n|x_n, w, \beta). \tag{2.18}$$

Subsequently, the error function can be produced by taking the negative logarithm

$$\frac{\beta}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2 - \frac{N}{2}ln\beta + \frac{N}{2}ln(2\pi) \tag{2.19}$$

which can be utilized in finding the w and β parameters in the following manner:

First the error function $E(w)$ is being produced, by discarding all the additive and multiplicative constants

$$E(w) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, w) - t_n\}^2. \tag{2.20}$$

Then, the w parameters are being determined by minimizing the $E(w)$ function; since they correspond to the *maximum likelihood* solution they are denoted as $w_M L$.

Finally, with the $w_M L$ known, the β parameters are obtained through the following equation

$$\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\{y(x_n, w_{ML}) - t_n\}^2. \tag{2.21}$$

Regarding the output unit activation function, there is a natural pairing between it and the error function. Specifically for regression problems, where the identity function is taken as the output activation, the sum of squares error function has the following property

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{2.22}$$

which will be utilized later in the **Error Backpropagation** paragraph.

Parameters Optimization

In order to obtain the best possible model parameters, the weights w have to be selected so that the chosen error function $E(w)$ is minimized.

This task is made easier to understand by depicting the error function as a continuous surface above the 2D weight space, as shown in Figure 2.13.

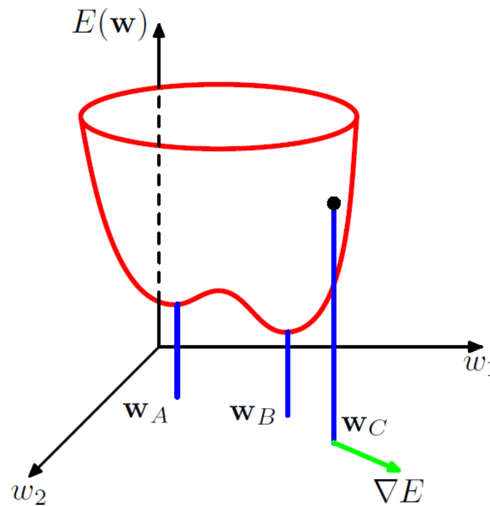

Figure 2.13: Geometrical depiction of the error function above the weight space.

By moving on the weight space by a value of δw, from w to $w + \delta w$, a point on the error function will move from the $E(w)$ to the $E(w) + \delta E$, where $\delta E \simeq \delta w^T \nabla E(w)$, with the $\nabla E(w)$ being directed towards the greatest rate of increase of the error function.

Since is the $E(w)$ is a smooth continuous function of w, the point in which it receives its minimum value is a point where the w eliminates the gradient of the error function, meaning

$$\nabla E(w) = 0 \tag{2.23}$$

as should its value be different than zero, then a point on the $E(w)$ could have been found where the error is smaller, by moving by $\nabla E(w)$. These

points where the above condition is satisfied are called *stationary points*, and can be categorized in *maxima*,*minima* and *saddle points*.

Note that while the objective is to find the w where the error function takes its smallest value, be it zero or otherwise a value adequately close to zero, it is not important to find the global minimum of the function. There are multiple points on the $E(w)$ which have a value that is the smallest compared to all the other points of the surrounding area, but do not have the absolute smallest value of the entire error function. These points are known as *local minima* and consist a potentially equally acceptable solution to the minimization problem.

Since the analytical solution of the 2.23 equation has been proven to be difficult and complex, iterative numerical processes are preferred. The most common one dictates that an initial value $w^{(0)}$ of the weight vector is to be arbitrarily chosen, and then the vector is to be updated in a number of iterations following the rule

$$w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)} \tag{2.24}$$

where τ are the iteration steps and $\Delta w^{(\tau)}$ the weight update, a term that, depending on the selected technique, takes different values.

A common choice is utilizing for the weight update the error gradient $\nabla E(w)$, evaluated at the new $\tau + 1$ step. In a technique known as the *gradient descent* optimization, the weight vector is updated each iteration by

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w^{(\tau)}) \tag{2.25}$$

where the $\eta > 0$ parameter is known as the *learning rate*. When the entire dataset is used to calculate the $\nabla E(w)$ term at each step, then this method is known as the *batch* method. Conversely, if the $\nabla E(w)$ term is calculated and the weights are updated by using smaller batches of the dataset, then the method is called *mini-batch*; should the batch size be one, meaning that the updates happen after every point of the set, then the technique is known as *stochastic gradient descent*. In this way, the weight vector is updated by the rule

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)}) \tag{2.26}$$

where the $E_n(w)$ error terms are segments of the error function calculated for each data point, as shown in equation 2.27

$$E(w) = \sum_{n=1}^{N} E_n(w). \tag{2.27}$$

A different, more complex approach comes in the form of the *Adam* optimization technique [61], whose name derives form the initials of *adaptive*

moment estimation. Following this method, the weights are adapted through the first and second moment of the gradient of the error function $E(w)$.

Given an initial value for the first and second moment of the gradient,$m_w^{(0)}$ and $v_w^{(0)}$ respectively, the two moments are calculated at each step with the formulae

$$m_w^{(\tau+1)} = \beta_1 m_w^{(\tau)} + (1-\beta_1)\nabla_w E^{(\tau)} \tag{2.28}$$

$$v_w^{(\tau+1)} = \beta_2 v_w^{(\tau)} + (1-\beta_2)(\nabla_w E^{(\tau)})^2 \tag{2.29}$$

where the β_1 and β_2 parameters are called the *exponential decay rates* of the first and second moment respectively. They receive values in the $[0,1)$ range, typically close to 1.

Afterwards, the two bias-corrected moments are calculated

$$\hat{m_w}^{(\tau+1)} = \frac{m_w^{(\tau+1)}}{1-\beta_1^\tau} \tag{2.30}$$

and

$$\hat{v_w}^{(\tau+1)} = \frac{v_w^{(\tau+1)}}{1-\beta_2^\tau}. \tag{2.31}$$

Finally, the weight vector is updated through the equation 2.32

$$w^{(\tau+1)} = w^{(\tau)} - \eta\frac{\hat{m_w}^{(\tau+1)}}{\sqrt{\hat{v_w}^{(\tau+1)}}+\epsilon}. \tag{2.32}$$

where η is once again the learning rate parameter and ϵ a really small scalar, typically having a magnitude of 10^{-8}, that helps avoid division by zero problems.

Error Backpropagation

As shown in the previous paragraph, while trying to optimize the weights of a network, the need to evaluate the gradient of the error function $E(w)$ arises. This is achieved through the utilization of a local message passing scheme, where information is alternately passed forwards and backwards through the network, called *error backpropagation*.

Specifically in the context of neural networks, the term error backpropagation is used to describe the evaluation of the error function derivatives, before they are used to optimize the weights. The reasoning behind this

naming convention is the fact that, at this specific point of the iterative process of network training, the derivatives are transmitted backwards in the network in order to compute the updated model parameters.

Equation 2.27 shows that the type of error function that is currently discussed in this section can be described as a sum of partial error terms, calculated for each data point. In the following paragraphs, the derivative of one of these terms, $\nabla E_n(w)$ will be analysed, first in a linear model and secondly in a feed forward network.

In an example of a simple linear model, whose output y_k is given by

$$y_k = \sum_i w_{ki} x_i \tag{2.33}$$

and whose error function, computed for an n point in the input dataset, is the following one

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2, \tag{2.34}$$

where $y_{nk} = y_k(x_n, w)$, the error function gradient with respect to the weight parameters is calculated as

$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni}. \tag{2.35}$$

Thus, each partial gradient can be described as a product of an input stimuli term, x_{ni}, and an error signal term, $y_{nj} - t_{nj}$, which is related to the output, w_{ji}.

Next, a neural network exampled will be discussed; the network will be considered as a feed forward type, with arbitrary differentiable non-linear activation functions and a generic error function.

The activation of a j unit is given by

$$z_j = h(a_j) \tag{2.36}$$

where $h(\cdot)$ is a non-linear activation function and the a_j term is the weighted sum of the previous i unit, as in

$$a_j = \sum_i w_{ji} z_i \tag{2.37}$$

Before the calculation of the error terms that will be propagated backwards in the network takes place, it is assumed that the input information has already passed forward through the entire network and the activations of all

the units, hidden and output ones, are calculated via the equations 2.36 and 2.37. This process is known as *forward propagation*.

Once again, the partial gradient of the error function, with respect to the weight parameters, is calculated, by applying the chain rule. Since the computation is regarding an n input point, every variable in the equation should include an appropriate subscript, but is omitted for reading clarity.

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}. \tag{2.38}$$

From 2.37 we get

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \tag{2.39}$$

and by introducing the δ's parameters, terms known simply as *errors*, which are defined by

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}, \tag{2.40}$$

the 2.38 equation can be compactly written as

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{2.41}$$

an equation that bears resemblance to the one of the linear model example, equation 2.35.

Thus, in order to obtain the error function derivatives, the δ parameters, for the hidden and output units, are to be calculated and then used in equation 2.41.

For the output units, as discussed in a previous paragraph, as per equation 2.22, the δ_k are given by

$$\delta_k \equiv \frac{\partial E}{\partial a_k} = y_k - t_k \tag{2.42}$$

while for the hidden units, the δ_j parameters are calculated using the chain rule

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}. \tag{2.43}$$

Finally, by applying the definition of δ from equation 2.40 into 2.43 we get a general backpropagation algorithm

$$\delta_j = h'(a_j)\sum w_{kj}\delta_k \tag{2.44}$$

The above equation shows that the δ values of a j hidden unit can be obtain by propagating backwards the δ values of k units further up the network.

A graphical depiction of this process is shown in Figure 2.14; with a blue arrow the forward propagation of information through the network is illustrated, while the red arrows represent the backpropagation of the error information.

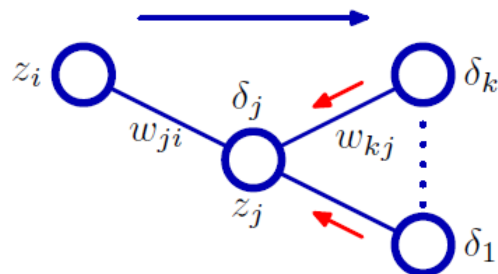

Figure 2.14: Depiction of the input and error information flow in a network.

2.3 Four-Stroke Diesel Engine Operating Parameters

The engine that the neural network models have been fitted on belongs to the category of *internal combustion engines*. The general purpose of such engines is the production of mechanical power through the chemical reaction of oxidization, or simply put, burning of fuel, a process that releases and utilizes its contained chemical energy. In internal combustion engines, this reaction takes place in specially built chambers inside the machine; this sets them apart from their *external combustion* counterpart, as in the case of the latter, the combustion happens from an outside factor.

Regarding the way the combustion process is initialized in the chamber, there are two categories of engines: *Spark Ignition* (SI) engines, also known as Otto engines, and *Compression Ignition* (CI) engines, also known as Diesel engines. In the first type, the fuel is mixed with the air before the insertion to the combustion chamber and the following burning of the mixture starts with the introduction of a spark. In the second type, the one that corresponds to the engine type of this thesis, air is inducted first in the combustion chamber and compressed, with the fuel being injected before the desired ignition point. Then, due to the significant increase in pressure and temperature in the chamber, the mixtures reaches the combustion point and auto-ignition occurs.

Finally, when referring to the operation cycles, two distinct groups exist: the *two-stroke* (2X) engines, which produce one power stroke for every revolution of the crankshaft, or two strokes of the piston, and the *four-stroke* (4X) engines, that produce one power stroke for every four piston strokes. The engine that is being modelled belongs to the latter subcategory of four-stroke engines; their four-part operating cycle is presented in Figure 2.15 [3].

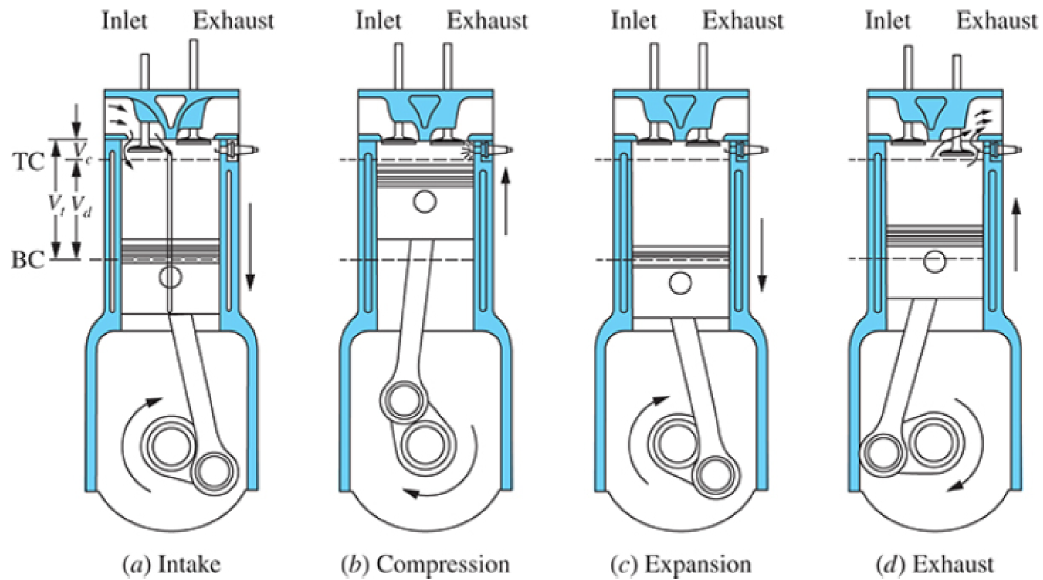

Figure 2.15: Depiction of the four-stroke operating cycle.

2.3.1 Brake-specific Fuel Consumption - BSFC

For an engine to produce work, a sufficient quantity of fuel needs to be provided and subsequently burned inside the cylinder. Since the generated power comes from the released chemical energy of the fuel matter, it can be generally implied that an increase to the fuel corresponds to an increase in the produced work.

A useful measurement of the fuel consumption in regards to the produced engine power is the *Brake-specific Fuel Consumption*, defined as

$$bsfc = \frac{\dot{m}_f}{P} \tag{2.45}$$

where $\dot{m}_f$ is the fuel mass flow rate per unit time, usually measured in gr/sec or gr/h, and P the engine power output.

Since this parameter is not dependant on the size of the engine, but rather takes into consideration only their power production, its regarded as a good assessment of the engine efficiency and can be used to compare different ones directly. Between two bsfc values, the smaller one is the optimal, as for the same amount of fuel consumed, higher levels of work are outputted by the machine.

2.3.2 Air-fuel Equivalence Ratio - Lambda (λ)

In order for an internal combustion engine to initiate the power production cycles, an appropriate mixture of air and fuel is required to be introduced in the combustion chamber. The ratio between the two is an important characteristic of the engine operation, as it controls the completeness of combustion process, and consequently the amount of energy produced, as well as the type and quantity of the created pollutants, like COx and NOx. This parameter is known as the *Air-fuel ratio*, or AFR, and is defined as

$$AFR = \frac{m_a}{m_f} \tag{2.46}$$

where m_a is the air mass and m_f the fuel mass. A different, less used approach to this parameter, is the *Fuel-air ratio*, or FAR, defined as the inverse of the AFR parameter

$$FAR = \frac{m_f}{m_a} = \frac{1}{AFR} \tag{2.47}$$

If the provided mass of air is exactly enough for a complete combustion of the entire mass of fuel, then this ratio is called *stoichiometric*. If the fuel mass is more than the required for a 1-1 combustion with the air mass, then the mixture is characterized as *rich*; on the other hand, if the mixture contains a greater amount of air that fuel, then it is known as *lean*.

In practise, most engines run on a different that the stoichiometric air-fuel analogy. The ratio between the actual mass of air and the ideal for a complete combustion mass of air is known as the *Air-fuel Equivalence Ratio*, or *Lambda* (λ). Using the above definition of the AFR, the λ is formally defined as

$$\lambda = \frac{AFR_{actual}}{AFR_{stoich}}. \tag{2.48}$$

Consequently, for $\lambda = 1$ the combustion is stoichiometric, while for $\lambda < 1$ and $\lambda > 1$ the mixture is considered rich and lean respectively.

Given their combustion method, all compression ignition engines operate on lean mixture; meaning that the λ parameter is always greater than 1.

2.4 Programming Tools

The programming language through which the construction, training and testing of the neural networks took place was *Julia*[62], a high-level, dynamic language. It combines several strengths of other existing programming tools, namely the speed of C/C++, the accessibility and readability of Python, the mathematical prowess of Matlab and the dymamism of Ruby, among others[63][64][65]. It is generally regarded as a powerful tool for numerical computing and analysis, thus it lends itself well for machine learning purposes. The development started in 2009, with the first public announcement taking place in 2012 through a blog post of the developers[63]; nevertheless the first official 1.0.0 build was released in 2018.

The main package used in the premises of this thesis was *Flux*[66], a machine learning library that had built-in tools and specialized functions, which streamlined significantly the process of neural network building and training. Other packages included the *CUDA* package[67], which allowed GPU utilization for quicker processing time, the *BSON* package[68], for saving and loading the trained models and the *CSV* package for working with delimited files.

Chapter 3

Model Design

In this chapter, firstly a description of the utilized data on which the models of this thesis were trained, is presented. In total two different sets of data were collected, corresponding to two different engines and operations parameters: *Dataset A* and *Dataset B*. Dataset A was a small in size set, consisting of 900 data points; in contrast, Dataset B was vastly bigger in comparison, comprising of roughly 25000 data points.

Afterwards, due to the fact that two different datasets were available, two sets of models were constructed and trained: *Model Set A* and *Model Set B*.

The first group, *Model Set A*, was trained on the Dataset A. The small quantity of data points that were used as train input meant that the training stage required less time, therefore a substantial number of model hyperparameters combination could be tested.

The second set of models, *Model Set B*, was to be trained with data from the vastly bigger Dataset B. Having taken into consideration the size of training set, a few, specific model hyperparameters were chosen to be utilized in the model building, reducing the training time considerably. The selection of the hyperparameters was based on the evaluation of the Model Set A; any combination of model parameters that produced poor results was discarded.

3.1 Data Overview

A total of two distinct datasets were compiled for the purpose of this thesis. The sets corresponded to two different engines operated at various states in order to ensure that a broad input vector was collected.

3.1.1 First Set of Data - Dataset A

The first training dataset, *Dataset A*, was collected from a 13 litres 6-cylinder heavy duty diesel engine[69][70], whose schematic is depicted in Figure 3.1[69].

The engine is equipped with an electrically actuated Exhaust Gas Recirculation (EGR) valve, that introduces through the intake manifold in the combustion chambers part of the exhaust gases. The purpose of such systems is to decrease the NOx emissions by lowering the in-cylinder temperature and by diluting the combustible mixture with inert gases. For this system to divert and recirculate the appropriate amount of exhaust gases, adequate back pressure is required.

The use of EGR systems, however, contrasts the requirements for engine operation without the production of visible smoke. The introduction of inert, burned gas inside the cylinder leads to a lowered AFR parameter, while the diversion of a fraction of the exhaust gas away from the turbine results in decreased turbine power, which in turn further restricts the intake of fresh air[71].

To amend this issue, an asymmetric twin-scroll architecture has been implemented to the turbine. The exhaust manifold is not uniform, but instead it separates the exhaust flow in two, diverting the gases from three of the cylinders in the small turbine scroll, while the rest of the flow is being lead to the large scroll turbine. The small scroll provides the high pressure needed for the EGR system to operate optimally and feeds through a cooler the gases into the intake manifold. The large scroll, on the other hand, has lower pressure and therefore tasked with helping the engine meet the AFR requirements. The end result of such a construction is lessened fuel consumption.

The engine also comes with a built-in pneumatically actuated wastegate valve that bypasses the large scroll, leading to over-boosting avoidance and pumping loss reduction.

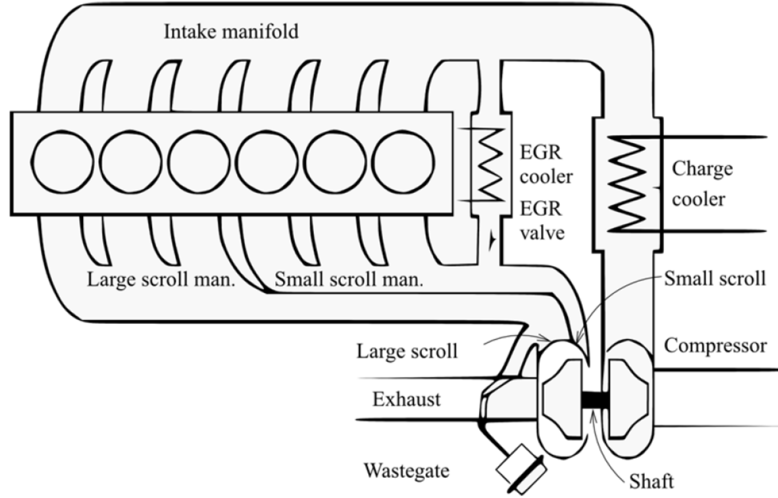

Figure 3.1: The schematic of the engine of Dataset A.

This engine was operated in 9 different steady states, covering 3 speed configurations, specifically 750, 1100 and 1400 RPM and a torque range of roughly 300 to 1600 Nm. The parameters of these states are presented in Table 3.1 bellow, as well in Figures 3.2 to 3.5. These parameters, namely Speed in RPM, Torque in Nm, Lambda and BSFC in gr/kWh were the input data for the training and testing phases of the neural networks.

Steady States Parameters				
Index	Speed [RPM]	Torque [Nm]	Lambda	BSFC [gr/kWh]
1	750	1289.6	1.37	192.81
2	750	815.4	1.76	191.75
3	750	343.6	3.12	207.18
4	1100	314.2	2.88	223.88
5	1100	1008.2	1.53	187.57
6	1100	1708	1.35	186.04
7	1400	1651.1	1.47	187.44
8	1400	1093.1	1.66	190.76
9	1400	341.9	2.45	232.93

Table 3.1: Parameters of the 9 Steady States

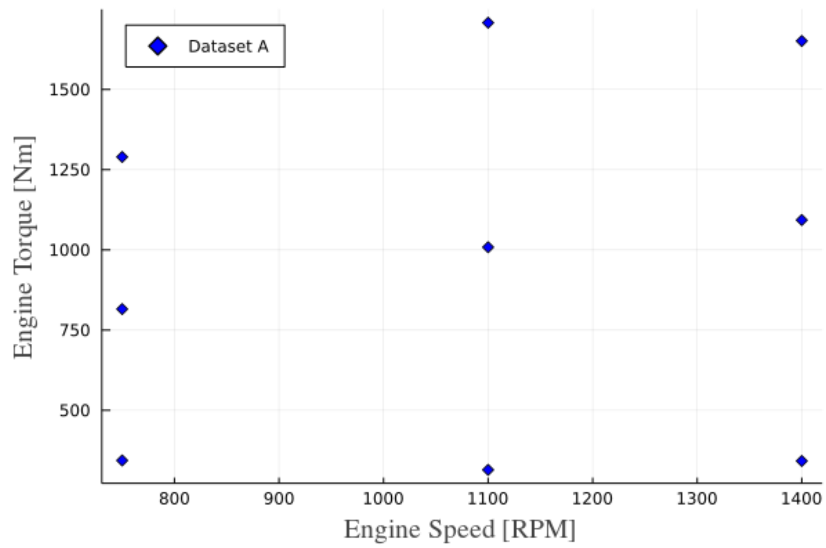

Figure 3.2: Torque-Speed diagram of the 9 operating states of Dataset A.

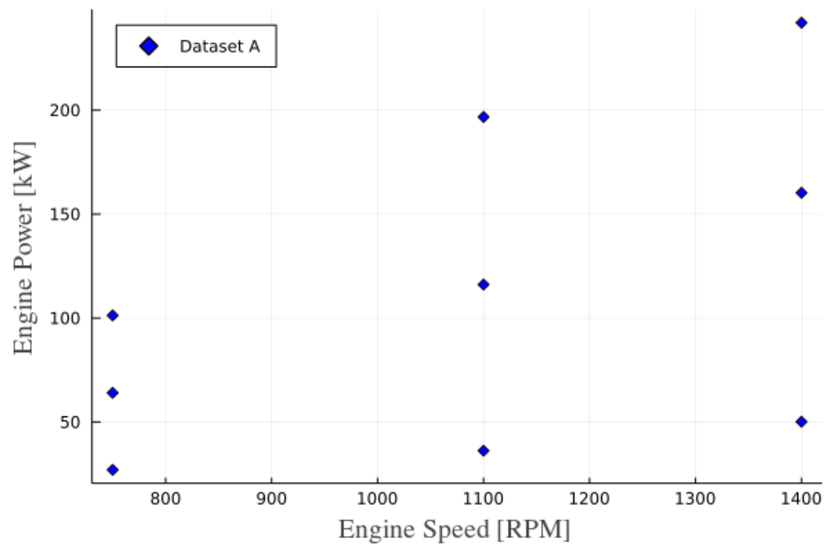

Figure 3.3: Engine Power-Speed diagram of the 9 operating states of Dataset A.

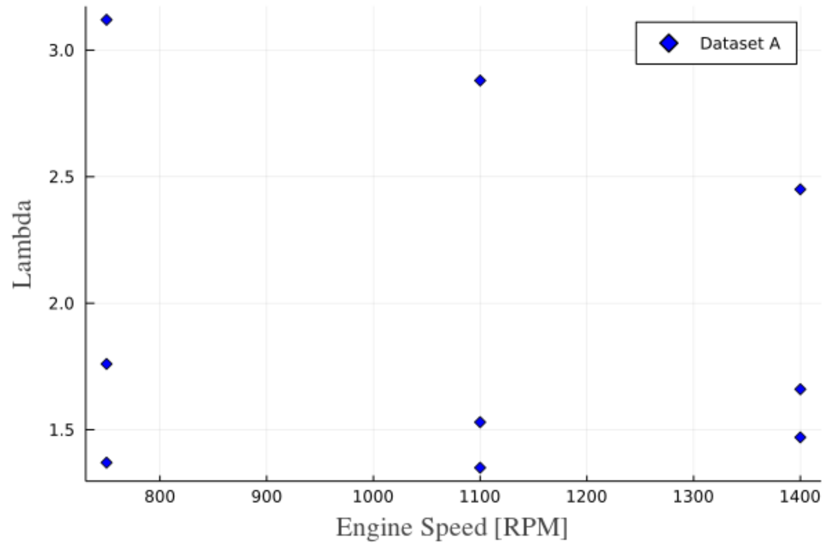

Figure 3.4: Lambda-Speed diagram of the 9 operating states of Dataset A.

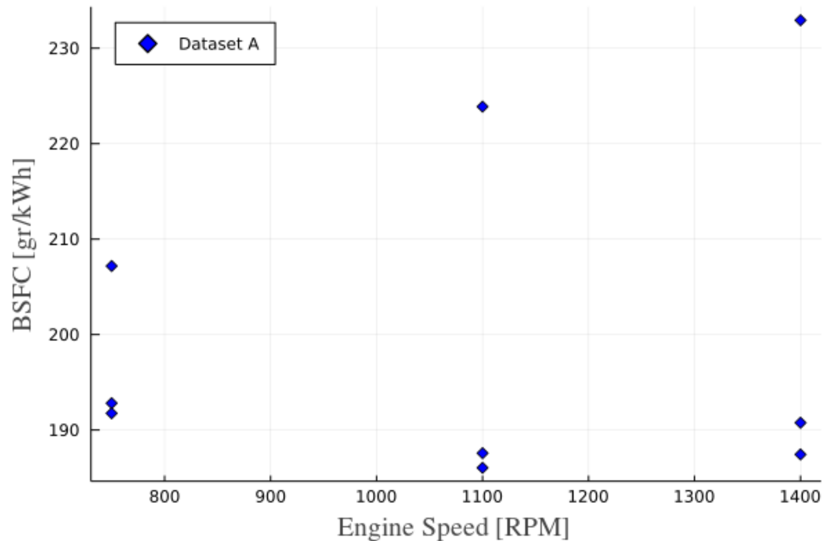

Figure 3.5: BSFC-Speed diagram of the 9 operating states of Dataset A.

For each state, in-cylinder pressure data was collected every 0.1 crankshaft *deg*, with the tests lasting two full rotations of the crankshaft, or one operating cycle, taking into consideration the type of the engine. The tests were repeated a total of 100 times for every operating state.

Out of all the available pressure measurements, the ones that corresponded to the peak cylinder pressure were singled-out and collated; thus

the final dataset comprised of the 100 maximum pressure values of each of the 9 states, adding up to a total of 900 points.

These data points are depicted in Figure 3.6 for each engine cycle, and in Figures 3.7 & 3.8 with regards to the engine speed and power.

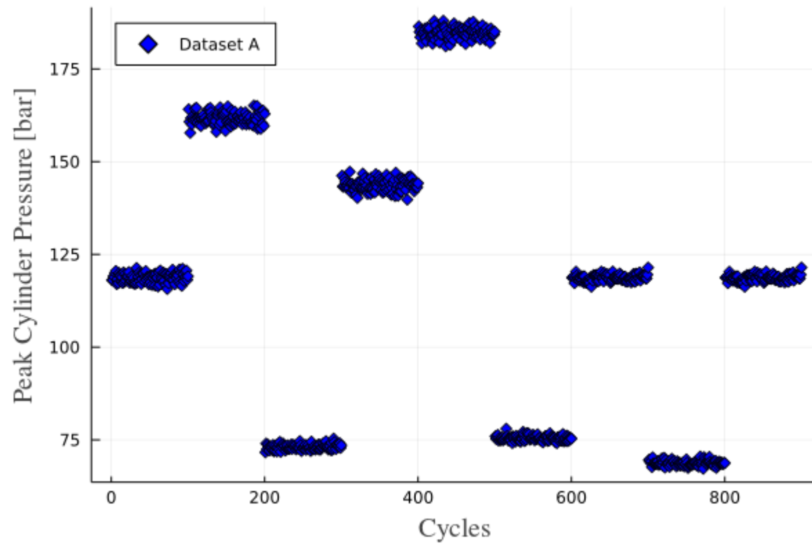

Figure 3.6: Peak Pressure measurements per engine cycles, for Dataset A.

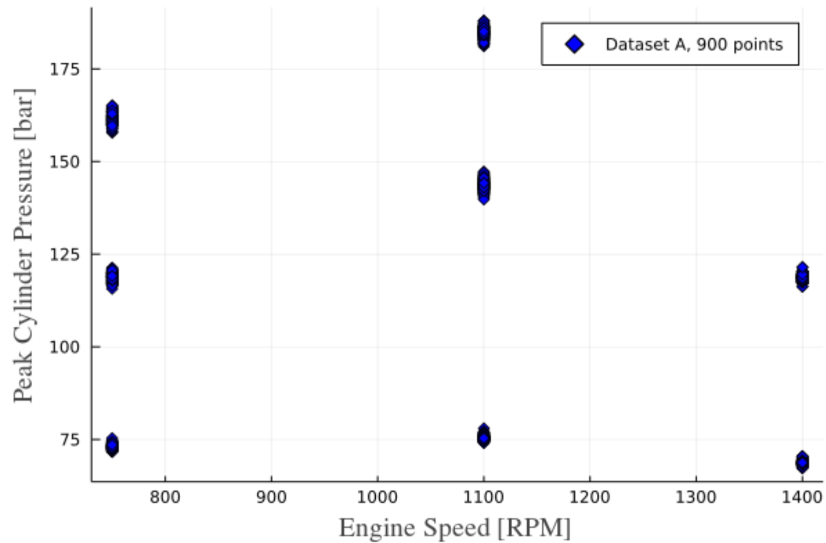

Figure 3.7: Peak Pressure-Speed diagram of Dataset A.

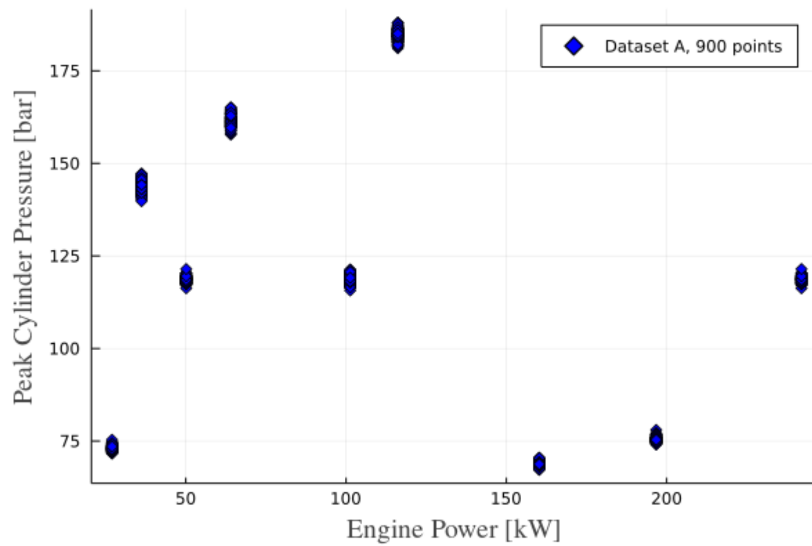

Figure 3.8: Peak Pressure-Engine Power diagram of Dataset A.

3.1.2 Second Set of Data - Dataset B

The second set of training data, *Dataset B*, consisted of measurements taken from the $MAN^{©}$ $5L16/24$ diesel generator [72] of the Laboratory of Marine Engineering, whose main characteristics are presented in Table 3.2.

The engine is coupled with an AEG electric dynamometer via a shaft and an elastic coupling, resembling as a result a ship's propulsion power train. It operates at a maximum of 500 kW and 1200 RPM. One important attribute of this engine is its ability to use as fuel both Heavy Fuel Oil (HFO) as well as Marine Diesel Oil (MDO)[73].

The engine is showcased in Figure 3.9[74].

Figure 3.9: A picture of the engine of Dataset B.

Engine Specifications	
Bore [mm]	160
Stroke [mm]	240
Cycle	Four-stroke
Cylinders	5
Speed [RPM]	1200
Power [kW]	500
Compr. Ratio	16.2:1
Dry Mass [t]	9.5
Configuration	In-line
Aspiration	Turbocharged

Table 3.2: Engine Specifications

The available data from the test runs of this engine were considerably more numerous comparatively to Dataset A. In total, 25320 operating cycles were carried out and data was recorded every 1 *deg* of crankshaft angle. The input parameters for the models that were extracted from these measurements were Speed in RPM, Torque in Nm and Lambda, and are presented in the following figures 3.10 to 3.12.

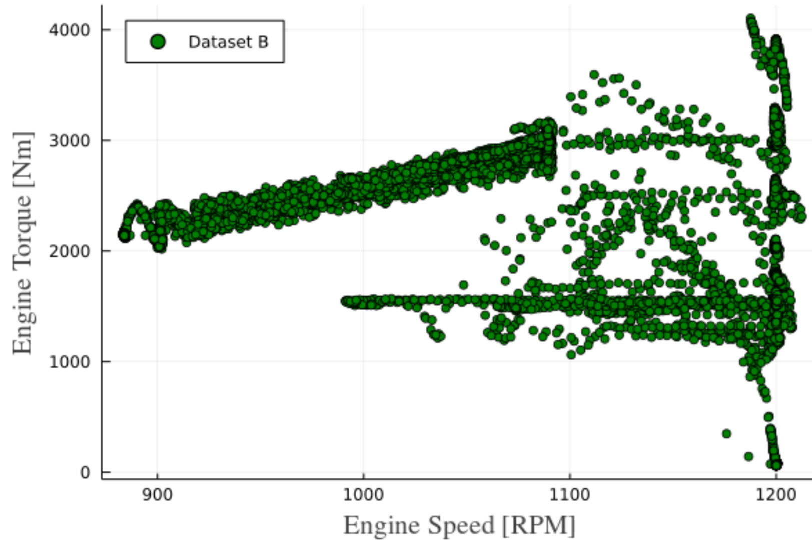

Figure 3.10: Torque-Speed diagram of the operating states of Dataset B.

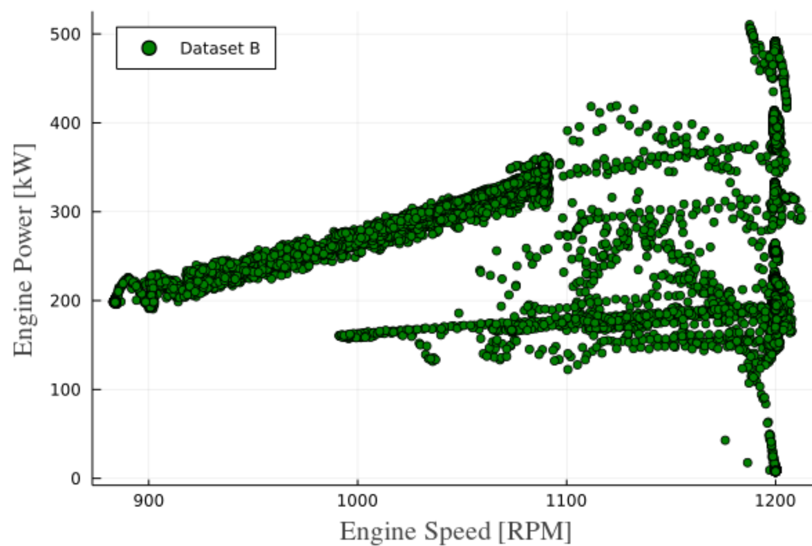

Figure 3.11: Engine Power-Speed diagram of the operating states of Dataset B.

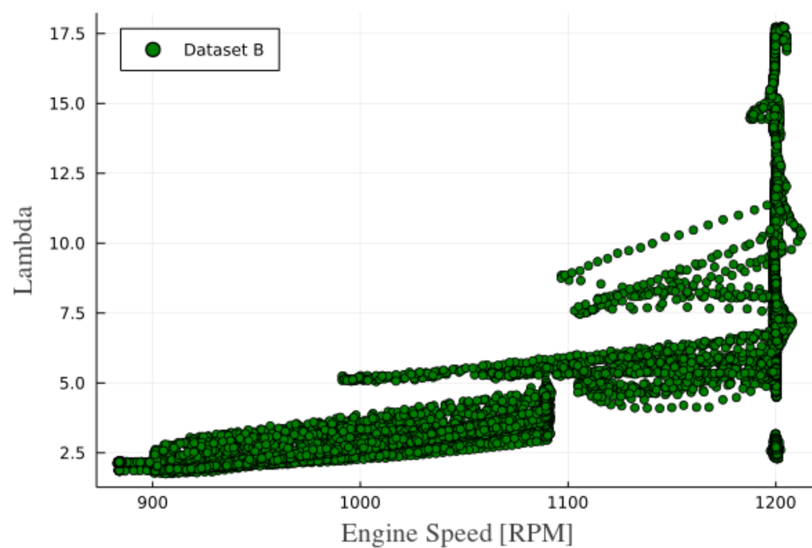

Figure 3.12: Lambda-Speed diagram of the operating states of Dataset B.

Once again, since the focus of the models were the prediction of the peak cylinder pressure, the maximum pressure values of each cycle were selected and concatenated into a singular dataset with 25320 data points. Figure 3.13 presents the peak pressure measurements for each engine cycle, while 3.14

and 3.15 depicts the pressure data in conjunction with the various engine speed configurations and power profiles.

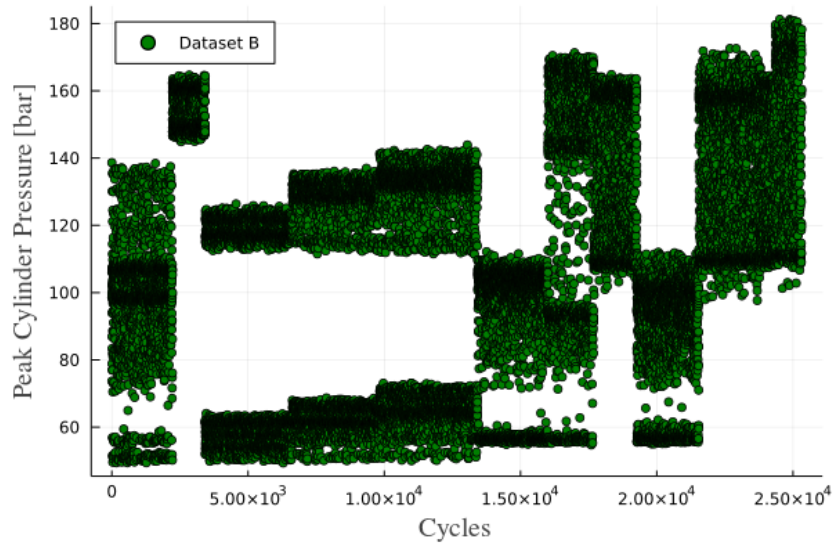

Figure 3.13: Peak Pressure measurements per engine cycles, for Dataset B.

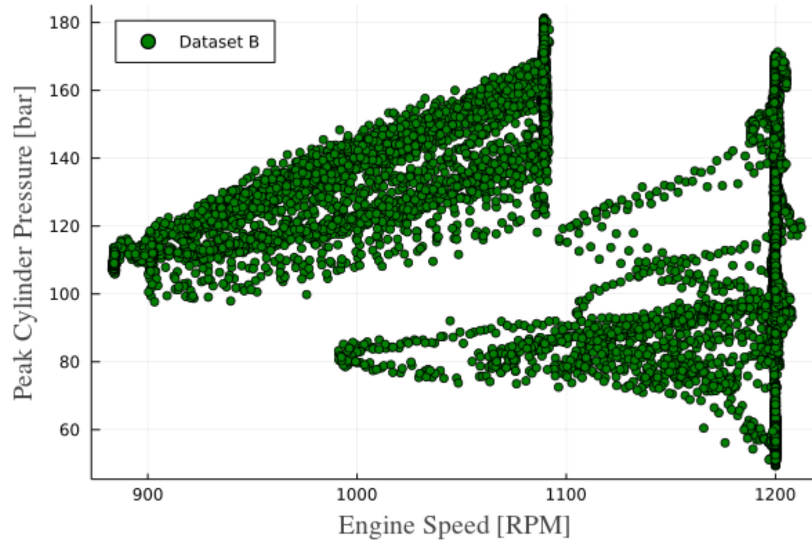

Figure 3.14: Peak Pressure-Speed diagram of Dataset B.

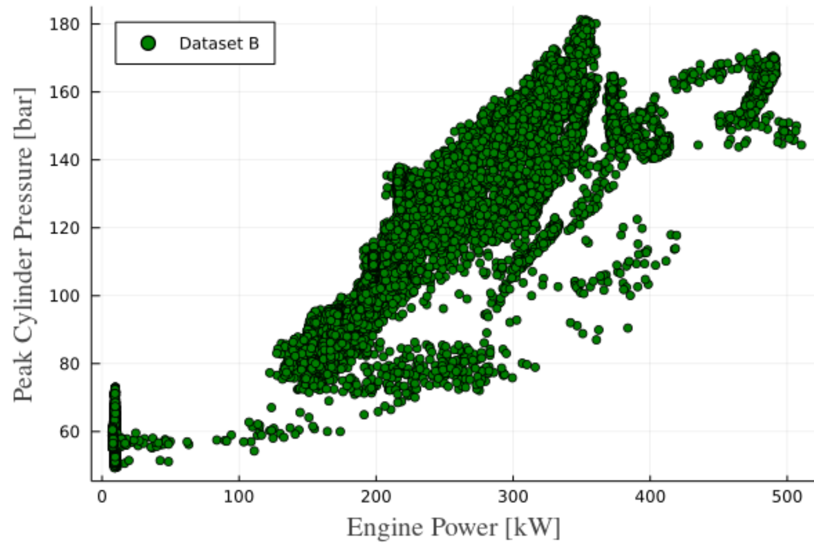

Figure 3.15: Peak Pressure-Engine Power diagram of Dataset B.

3.1.3 Datasets Comparison

The following figures 3.16 and 3.17 present both Dataset A and B in the same plot, for easier comparison and better understanding of their differences.

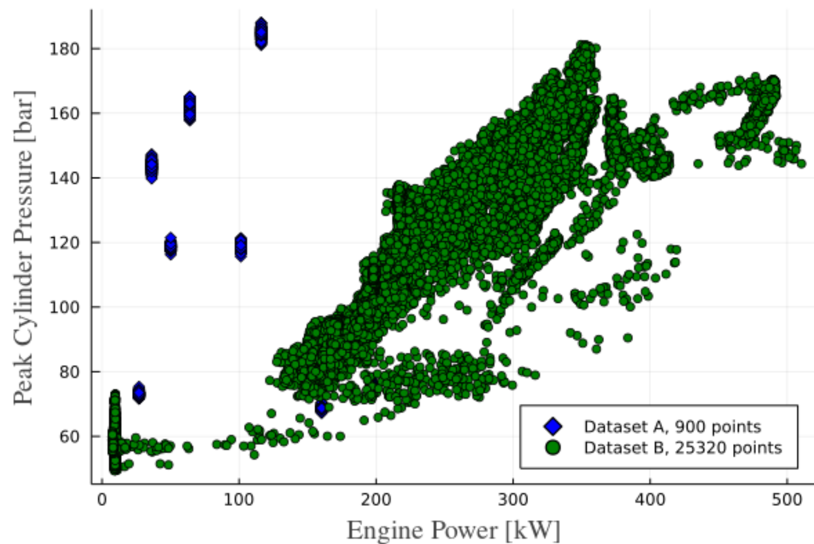

Figure 3.16: Peak Pressure-Engine Power diagram of both Dataset A and Dataset B.

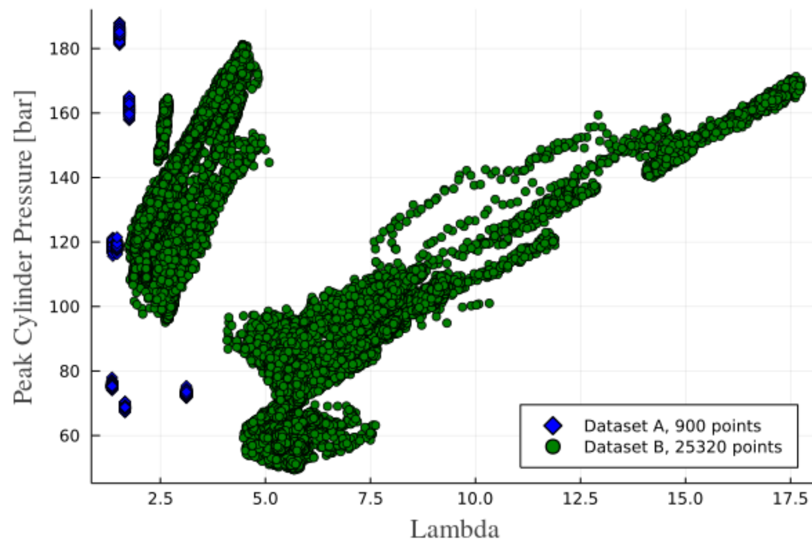

Figure 3.17: Peak Pressure-Lambda diagram of both Dataset A and Dataset B.

As is plainly depicted, the two sets of data contrast each other not only in their length, with Dataset B being roughly 30 times the size of A, but also in their parameters range. Set A lies in a much narrower area, for both Engine Power and Lambda, with little variation to its data points; on the other hand, Set B spans for the most part a different parameter range, while at the same time providing a much more expansive spectrum of data points.

3.2 Input Parameters

As mentioned in Section 3.1, Dataset A provided information regarding the Speed, Torque, Lambda and Specific Fuel Consumption of the various engine operating cycles, while Dataset B included data only for Speed, Torque and Lambda. All these parameters were used for the training of the respective model sets, however a few combinations were tried in order to investigate whether better results are produced with specific input pairs.

For Dataset A, four different input scenarios were tested: using only Speed and Torque as input, using Speed, Torque and Lambda, using Speed, Torque and BSFC and finally using all of them in conjunction, Speed, Torque, Lambda and BSFC.

For Dataset B, two combinations were tried: one with Speed and Torque and one with Speed, Torque and Lambda.

3.3 Data Preparation

Before the training sets could be utilized by the neural networks, they needed to undertake some initial pre-processing.

Firstly, the data points were split into two distinct groups: one used for the actual training purposes and the other one for testing the accuracy of the models afterwards. The train group consisted of 70% of the total points, while the rest 30% was used for testing. The train/test data points were randomly allocated to their respective sets, but the groups themselves were kept unchanged between the different models, in order to ensure that the exact same conditions were implemented during the training of the networks.

Data Set	
Training 70%	Testing 30%

Figure 3.18: The training and testing groups.

Afterwards, the train set was normalized via the formula

$$\hat{x}_i = \frac{x_i - max(x_i)}{max(x_i) - min(x_i)} \tag{3.1}$$

where x_i is an i point if the dataset, $\hat{x}_i$ is the normalized point and $max(x_i)$, $min(x_i)$ are the maximum and minimum values of the set respectively. By this type of normalization, the entire information range is converted into a $[0, 1]$ range.

The reasoning behind the normalization of the data stems from the fact that the process of network training has been proven to be enhanced when the data is confined in a relatively smaller range. Namely, the calculated model error is significantly reduced, while at the same time the duration of the training phase is also reduced by a non-neglectable amount[75].

This process took place for every column of the input set, including the target values used for the error estimation, with every type of input having, naturally, different maximum and minimum values. These normalization parameters were saved even after the conclusion of the network training, since they consist a defining characteristic of the models; if the models were to be utilized to acquire further results with an unknown input vector, or during the testing phase, any information fed to them would have to be normalized by these same parameters. If, instead, different maximum and minimum values were arbitrarily chosen, perhaps based on the range of data that was to be processed by the networks, then the risk of incorrect use of

the models would arise, as different input condition would be implemented. Note that a normalized "0" input, based on specific maximum and minimum parameters, would not be equal to a "0" input procured by using different normalization parameters.

3.4 Model Hyperparameters

As presented in the previous Chapter, a number of distinct neural network architectures have been designed, each bearing its own merits and used for tackling different tasks.

Both the number of hidden layers and the number of neurons per layer are variables that factor in the successful training of a model. Too few neurons or layers hinder the process of learning and disallow a network to correctly adapt to the training data; conversely too many neurons or layers may cause the known issue of overfitting[4], leading to bad generalization of the model to a new, unknown input vector. The general consensus is that complex problems with vast training sets tend to require a large amount of neurons with many layers to efficiently fit the data, and vice versa.

Moreover, several different other hyperparameters, example being the activation function, the optimization algorithm, the learning rate and so forth, influence the end result of a neural network. Once again, the optimal choice is not definite, rather it depends on the type of problem and the size and complexity of the input dataset.

For obtaining the best performing model, a number of varying neural networks with numerous combinations of model parameters were constructed, tested and evaluated. The exact types of hyperparameters that were tested are detailed in the following sections.

3.4.1 Number of Hidden Layers

For the problem that this thesis aims to tackle, it is generally expected that a small number of hidden layers should suffice, since it can be considered as being relatively simple. However, for completeness of the parameter investigation, models with higher-level structures were implemented.

Thus, three different sets of architectures were tested for both Model Set A and Set B: one with three, one with four and one with five hidden layers. As a result, according to the definition given in Chapter 2, the networks that were being trained can be considered as *Deep Neural Networks*.

As a note, the layers that were used in all models were of the *Dense* type, meaning that every neuron in a layer is linked through weighted connections to every neuron in the proceeding and preceding layer[76].

3.4.2 Number of Neurons per Layer

Similarly to the number of hidden layers, due to the nature of the prediction task, a small number of neurons were expected to be needed to accurately fit

the training subset of Dataset A. Nevertheless, in order to verify the degree of model adaptation and to confirm that overfitting has been avoided, varying numbers of neurons were tested.

Considering a model with three hidden layers, the chosen number of hidden neurons were 15-18-17. For every extra hidden layer, the chosen number of neurons were 17. Hence, for four hidden layers the neurons sequence was 15-18-17-17, and for five 15-18-17-17-17. Two more variations to the number of neurons were tested; in one version the number was tripled, while in the second one it was increased tenfold.

Summarizing, the three different architectures in regards to the amount of neurons that were created once again for both model Set A and B are the following:

- *Four Layered Network*

  Hidden neurons: 15-18-17, 45-54-51, 150-180-170

- *Five Layered Network*:

  Hidden neurons 15-18-17-17, 45-54-51-51, 150-180-170-170

- *Six Layered Network*:

  Hidden neurons 15-18-17-17-17, 45-54-51-51-51, 150-180-170-170-170

3.4.3 Activation Functions

The activation function, as discussed in greater detail in Chapter 2, is a method that is tasked with receiving the information as passed from a previous layer, transform it according to its rule, and then direct it into the next layer. A common choice of activation functions are non-linear ones; linear activations are sometimes used as well, but they greatly hinder the learning abilities of models, especially when faced with highly non-linear data, so their use is limited.

Three different, non-linear activation functions were tested for the neural networks of Set A: sigmoid, tanh and leakyReLU activation function. For Model Set B only leakyReLU was used.

- *Sigmoid activation function*, as given by the formula:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.2}$$

  The sigmoid function, also known as *logistic* function, takes any value from (-∞, ∞) and returns them in the $(0, 1)$ range, so as a result, it is

especially utilized in problems that have probabilities as an outcome. Large positive values are returned as 1, while large negative values are returned as 0. Its main advantage lies in that it is continuous and differentiable everywhere; however, its derivative produces very small gradients, as a result hindering significantly the training process. This issue is known as the *vanishing gradient* problem[77].

The sigmoid function is depicted in Figure 3.19[78].

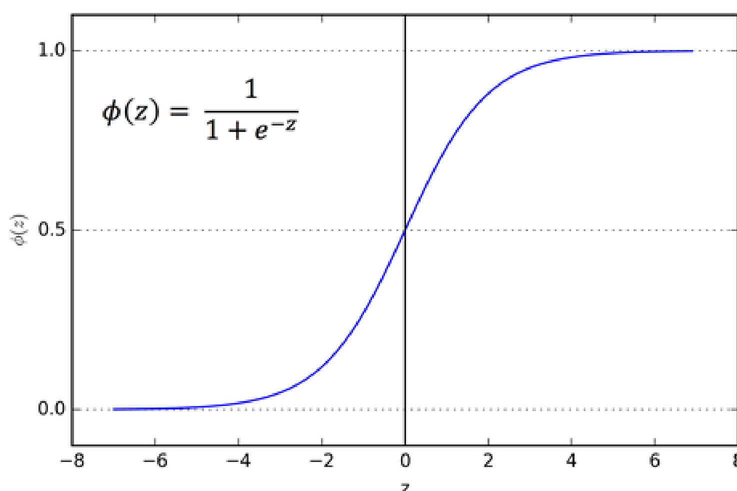

Figure 3.19: Depiction of the sigmoid activation function.

- *Tanh activation function*, as given by the formula:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \quad (3.3)$$

The hyperbolic tangent, or tanh, function can be considered as being very similar to the sigmoid one; they are both S-shaped, bounded, continuous and differentiable functions. Its chief difference, however, is its boundaries. In contrast to the sigmoid function, that exists in the $(0, 1)$ range, the tanh function is zero-centred, and spans the $(-1, 1)$ range. This symmetry is generally regarded to help the algorithm learn faster[79]. Of course, the problem of vanishing gradient is also present in this case, albeit to a slightly lesser degree.

The tanh function is depicted in Figure 3.20[80].

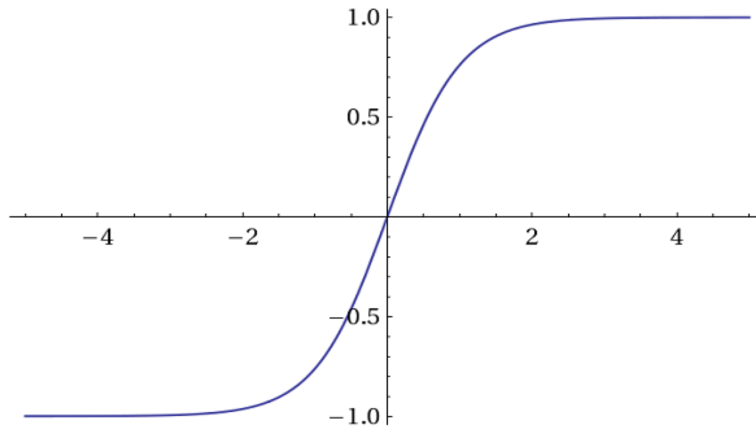

Figure 3.20: Depiction of the hyperbolic tangent function.

While both the hyperbolic tangent and the sigmoid function have been deemed as mediocre choices for activations, due to their tendency of highly positive or negative values saturation and their weak derivative, the former is generally preferred, if a choice were to be made between the two[81].

- *ReLU/LeakyReLU activation function.*

  The Rectifier Linear Unit, or ReLU, activation function is a two-branched function, given by the following equation:

$$relu(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}, \tag{3.4}$$

  which can also be compactly written as:

$$relu(x) = max(0, x). \tag{3.5}$$

  The above equation entails that negative values ar returned as 0, while positive ones are returned as-is. This attribute grants the ReLU function the property of being a non-linear function, while also retaining the "useful" qualities of a linear function. It is computationally easier to implement than the sigmoidal ones, at the same time succeeding in mitigating to a satisfactory degree the vanishing gradient and saturation issues.

  The ReLU function is depicted in Figure 3.21[82] bellow.

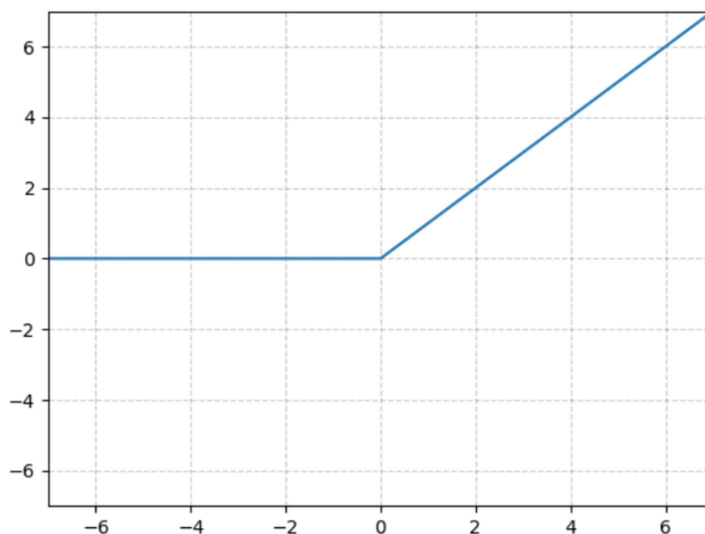

Figure 3.21: Depiction of the Rectifier Linear Unit function.

Naturally, issues that come with the use of this particular activation function do exist; namely, the fact that it is not differentiable at 0 and it not being zero centred. A notable one is the *Dying ReLU* problem: the optimization algorithm doesn't update the weights of those units that have a 0 gradient, thus any initially not activated neuron will often remain inactive for the entire course of training. Since there isn't any way for the unit to recover from the inactivity, this effectively leads to a "neuron necrosis".

One way to overcome the Dying ReLU issue is through variations to the original function algorithm. Such a variation takes the form of a two-branched function that doesn't return the negative values to zero, but instead returns them multiplied by a relatively small variable; this type of ReLU is called *LeakyReLU*, as it allows some negative information to "leak" through to the function output. Its formula is presented in Equation 3.6

$$leakyrelu(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \tag{3.6}$$

where the α variable is any small, positive number. A common choice is $\alpha = 0.01$.

The LeakyReLU function, with $\alpha = 0.02$, is presented in figure 3.22[83].

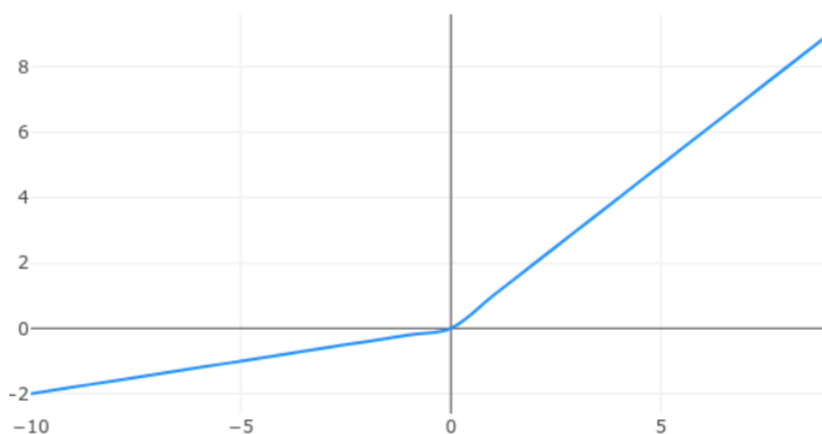

Figure 3.22: Depiction of the Leaky Rectifier Linear Unit function.

During the training stage, most of the designed models that used ReLU as their activation functions produced zero-value outputs, results that indicated the existence of the dying ReLU problem. Consequently, all the networks that were used and evaluated incorporated the LeakyReLU counterpart of the function, with the α variable being equal to 0.01.

3.4.4 Optimization Algorithms

The chosen optimization technique's task is to update the model's weight, according to the training data provided, in order to minimize the deviation between the model output and the desired target value. This is achieved by backpropagating the error information through the model, usually in the form of the gradient of the chosen error function, by following a specific rule. This update rule is dependant on the particular selected algorithm.

For the constructed models of Set A, two different algorithms were tested: Gradient Descent and ADAM. For Model Set B, however, only the ADAM optimizer was used.

- *Gradient Descent*, which is the simplest form of optimization. The weight update rule, as given by the Equation 2.26, takes the form of

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E_n(w^{(\tau)}), \tag{3.7}$$

where w are the weight coefficients, τ is the step of the iterative process, η is the learning rate parameter and $\nabla E_n(w)$ the gradient of the error function, calculated in an n-batch of the dataset.

- *ADAM Optimizer*, a more complex optimization method that utilizes the first and second moment of the gradient of the error. The weight update, given by the Equation 2.32, is the

$$w^{(\tau+1)} = w^{(\tau)} - \eta \frac{\hat{m_w}^{(\tau+1)}}{\sqrt{\hat{v_w}^{(\tau+1)}} + \epsilon}, \tag{3.8}$$

where η is once again the learning rate parameter and ϵ a small scalar. The $\hat{m_w}$ and $\hat{v_w}$ parameters are factors that introduce the first and second moment of the ∇E to the update rule respectively, given by Equations 2.30 and 2.31.

Both of the aforementioned techniques are presented in greater detail in Chapter 2, in Section 2.2.3.

3.4.5 Error Function

The error, or loss, function is the function through which the model accuracy is evaluated and by whose gradient the optimization, and consequent weight coefficients update, takes place. The choice of an appropriate function is deeply influenced by the nature of the task and the used activation function; for categorization problems the *cross-entropy* error function is used, while for regression tasks the *mean squared* error function is employed.

Consequently, for all the trained models. the mean squared error function was the chosen loss function, as given by the following equation

$$MSE = \frac{1}{N} \sum_{i}^{N} (\hat{y}_i - y_i)^2 \tag{3.9}$$

where N is the total number of points, $\hat{y}_i$ is the predicted output and y_i the desired outcome, both values referring to the training dataset.

3.4.6 Learning Rate

The success of the chosen optimization algorithm is highly affected by the learning rate, η. This parameter controls the size of the step that is taken every iteration, towards the loss function minimization. Too big of a step, and the algorithm transitions to new values sharply, causing it to converge to suboptimal solutions or completely diverge altogether. Conversely, if the learning rate is not sufficiently large, the number of steps needed for convergence is drastically increased, slowing down the training process significantly.

The optimal size of the η is usually model and optimizer dependant; it is sometimes chosen through a trial-and-error approach [81].

The model convergence with regards to the choice of the learning rate is presented in Figure 3.23[84]. Three different cases are illustrated: one with too large η, one with too small η and one with an optimal η parameter.

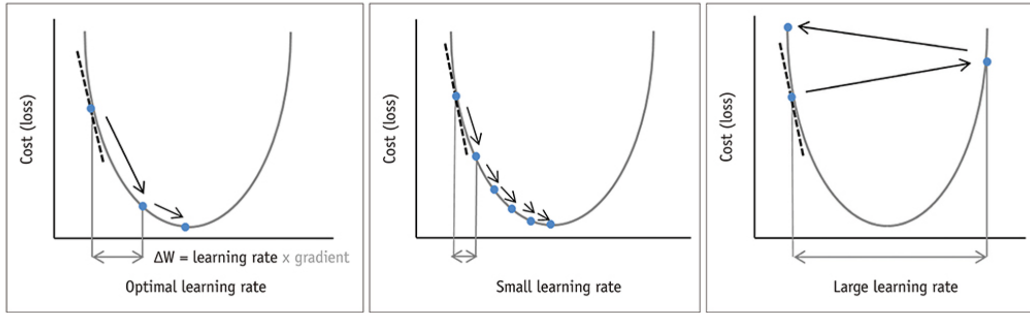

Figure 3.23: Depiction of the effect of the learning rate parameter on the model training, in three different scenarios.

For the premises of this thesis, three different learning rate values were tested during the training of Set A models: $\eta = 0.1$, $\eta = 0.01$ and $\eta = 0.001$. The last, smallest value of $\eta = 0.001$ was chosen for the Set B models.

3.4.7 Batch size and Epochs

Batch size refers to the number of elements that comprise the fragmented parts of the initial dataset. As mentioned before, the optimization techniques update the model weights after evaluating a portion of the dataset. If the size of the portion is equal to the size of the dataset, then the entire set is processed before updating the coefficients, while should the size be equal to 1, then the weights are updated after the evaluation of the error function on each element of the set. It has been proven that using the entire set during each step of the optimization yields worse results than using smaller batches of data [4]. Consequently, it could be surmised that the smallest sized batches produce the best result, albeit taking longer time to train; this line of thinking can be considered true in some cases, however the optimal batch size is yet again deeply case specific.

In any case, a total of eight different batch sizes were studied: 1, 2, 4, 8, 16, 32, 64 and 128, with only the values of 8 to 128 being used in Set B.

Epochs are the total rounds of weight updates that the optimizer iterates over, in order to reach the loss function minimization point. Broadly speaking, larger learning rates require fewer iterations, thus smaller epoch

values, to reach the convergence, while on the other hand, smaller learning rates need a greater amount of training cycles to reach the error function minimum. Having said that, incorrect selection of the number of epochs can be a detriment to the model. If the epochs are lacking, then the weights are not updated sufficiently and the network finishes the training stage without actually learning from the input data. Similarly, if the number is too steep, then the danger of overfitting the training data is further exasperated.

All the models were trained in three different epochs scenarios: with 100, 500 and 1000 epochs.

3.4.8 Model Accuracy Metrics

In order to assess the effectiveness of the trained models, via utilization of the test dataset, the deviation between the predicted output and the actual value had to be calculated.

To this end, three different metrics were implemented:

- *Mean Absolute Error*, or *MAE*, as given by the formula

$$MAE = \frac{1}{N}\sum_{i}^{N}|\hat{y}_i - y_i| \tag{3.10}$$

  where N is the total number of points, $\hat{y}_i$ the estimated by the model output and y_i the actual value. The MAE formula takes the average of all the differences between the two values, therefore it comes with its own unit of measurement, that being equal to the one the predicted values have.

- *Mean Absolute Percentage Error*, or *MAPE*, as given by the formula

$$MAPE = \frac{1}{N}\sum_{i}^{N}\frac{|\hat{y}_i - y_i|}{y_i} \tag{3.11}$$

  where the parameters are similarly defined as the ones in the MAE Equation 3.10 above. Since the the MAPE formula deals not with the absolute differences between the estimated and desired outcome, but with the relative to the actual value percentile deviation, it produces easier to evaluate results; a 0.2 MAPE, for example, indicates a 20% divergence of the network output with regards to the real value. Its chief disadvantage, however, is the fact that it cannot be used for data that includes a zero-value target value, as division with zero issues arise. As a result, the MAPE metric is unusable when dealing with normalized data, making the utilization of the MAE or a similar metric a necessity.

- *R-Squared*, or R^2, as given by the formula

$$R^2 = \frac{\sum_i^N (\hat{y}_i - y_i)^2}{\sum_i^N (y_i - \bar{y}_i)^2} \tag{3.12}$$

  where once again the parameters N, $\hat{y}_i$ and y_i are defined as in the MAPE and MAE description, and $\bar{y}_i$ is the average value of y_i. The R-squared, also known as *coefficient of determination*, is a form of measurement of how well a mathematical model fits a set of data. It takes values in the [0,1] range; the closer the coefficient lies to 1, the more accurate the fitting of the model is, and vice-versa.

3.4.9 Model Nomenclature

Due to the seer size of the different model hyperparameters combinations that were explored in this thesis, a proper naming convention had to be established during the network training and saving process. Thus, it was decided that the parameters that defined each model were to be reflected in its file name.

Figure 3.24 presents an example name of a trained network, and Figure 3.25 bellow explains the content of each coded parameter.

Act leakyrelu-Lr0.01-Opt ADAM-H.Layers 3-Neurons 15 18 17-VarST-Ep500-Btch4

Figure 3.24: Example of a trained model name.

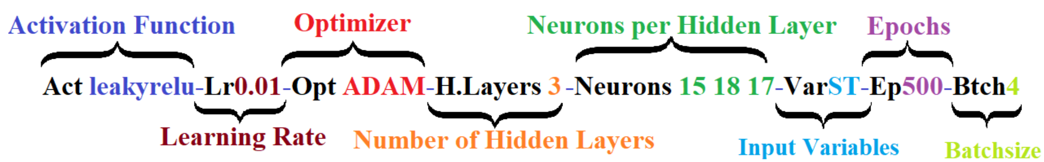

Figure 3.25: Analysis of the above model name.

Chapter 4

Results and Discussion

In this chapter, the evaluation of the trained models takes place. The test subset of the two datasets, consisting of 30% of the total points that were not applied for the training process, was used to produce predictions of the peak cylinder pressure. Out of the entire spectrum of models, only the best one for each set of inputs are being presented; in total four different ones for the Dataset A and two for Dataset B. After the showcase of the best performing networks, a brief summary of the observations regarding the effect of the various model hyperparameters on the outcome takes place.

4.1 Model Set A

In this model set, as Section 3.2 explained, four combinations of inputs were employed: models that used Speed and Torque (ST) as input, models that used Speed, Torque and Lambda (STL), models that used Speed, Torque and BSFC (STB) and finally models that used all of them in conjunction, Speed, Torque, Lambda and BSFC (STLB).

The best model of each input group is presented in Table 4.1, while the following section gives more details on them.

Model Name	Inputs	Accuracy
Act leakyrelu-Lr0.001-Opt Descent-H.Layers 5-Neurons 150 180 170 170 170-VarST-Ep500-Btch4	ST	99.31%
Act leakyrelu-Lr0.001-Opt Descent-H.Layers 4-Neurons 15 18 17 17-VarSTL-Ep500-Btch2	STL	99.32%
Act leakyrelu-Lr0.001-Opt Descent-H.Layers 3-Neurons 15 18 17-VarSTB-Ep1000-Btch4	STB	99.25%
Act leakyrelu-Lr0.001-Opt Descent-H.Layers 3-Neurons 45 54 51-VarSTLB-Ep500-Btch1	STLB	99.29%

Table 4.1: Comparison of the best models for each combination of input variables - Set A

4.1.1 Best performing networks, per Set of Inputs

Speed-Torque (ST) Models

For networks that used the Speed-Torque vector as training input, error of less than 1 bar was achieved, as shown in 4.3, amounting to an accuracy of 99.31%. The hyperparameters of this model are presented in Table 4.2.

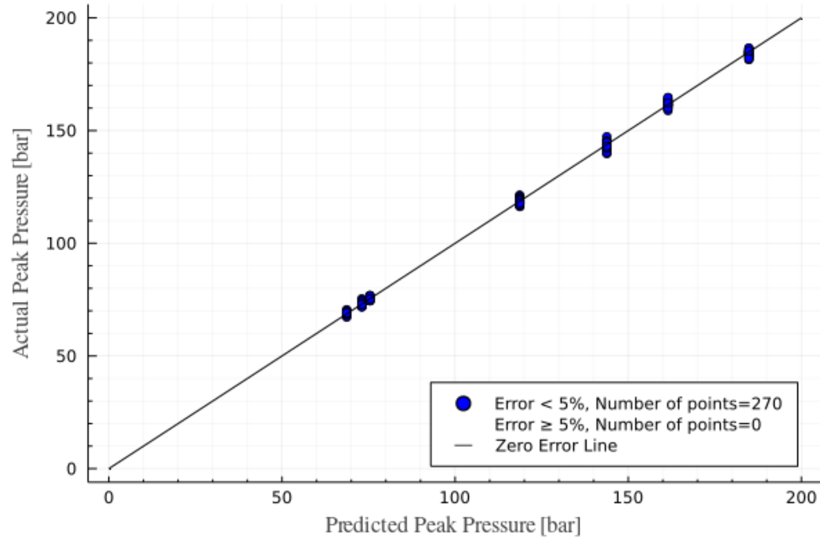

Figure 4.1: The zero-error-line diagram of the best performing model of Set A, with Speed-Torque as input.

Model Parameters	
Hidden Layers	5
Number of Neurons	150 180 170 170 170
Optimizer	Descent
Learning Rate	0.001
Activation function	leakyReLU
Epochs	500
Batchsize	4

Table 4.2: Hyperparameters of the best model, for ST input - Set A

Model Accuracy	
Mean Absolute Error, MAE [bar]	0.8184
Mean Absolute Percentage Error, MAPE [%]	0.6898
R^2	0.999175

Table 4.3: Error metrics of the best model, for ST input - Set A

Speed-Torque-Lambda (STL) Models

Regarding networks that used the Speed-Torque-Lambda vector as training input, the model accuracy was 99.32%; making it the best overall model. In Table 4.5 the error metrics are presented, and the model parameters in Table 4.4.

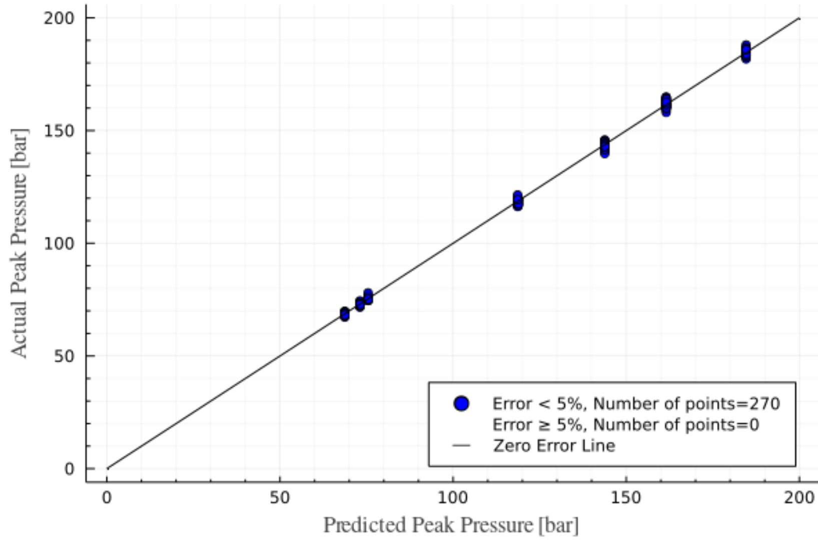

Figure 4.2: The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda as input.

Model Parameters	
Hidden Layers	4
Number of Neurons	15 18 17 17
Optimizer	Descent
Learning Rate	0.001
Activation function	leakyReLU
Epochs	500
Batchsize	2

Table 4.4: Hyperparameters of the best model, for STL input - Set A

Model Error Metrics	
Mean Absolute Error, MAE [bar]	0.8303
Mean Absolute Percentage Error, MAPE [%]	0.6820
R^2	0.999219

Table 4.5: Error metrics of the best model, for STL input - Set A

Speed-Torque-BSFC (STB) Models

Models that instead of the Lambda input variable used the Brake Specific Fuel Consumption one, the input vector thus being Speed-Torque-BSFC, an accuracy value of 99.25% was achieved, with the Mean Absolute Error being 0.88 bar, as shown in Table 4.7. The hyperparameters of this model are once again presented in Table 4.6.

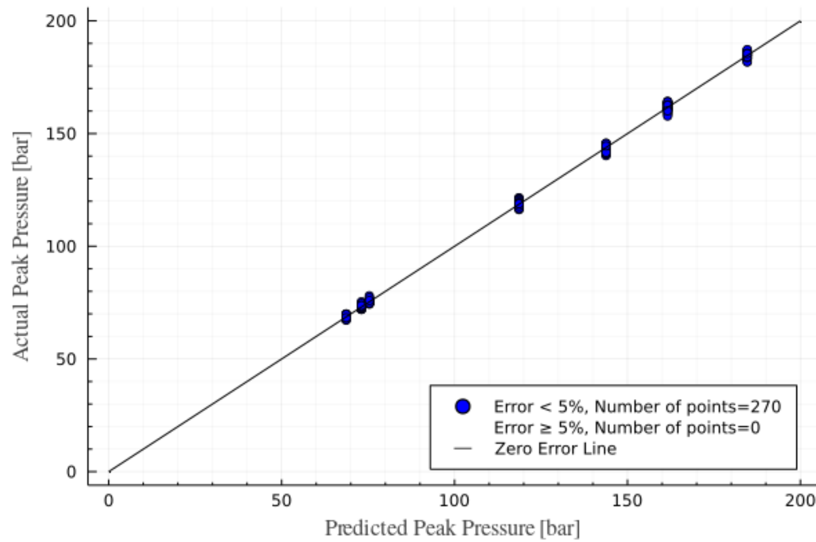

Figure 4.3: The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-BSFC as input.

Model Parameters	
Hidden Layers	3
Number of Neurons	15 18 17
Optimizer	Descent
Learning Rate	0.001
Activation function	leakyReLU
Epochs	1000
Batchsize	4

Table 4.6: Hyperparameters of the best model, for STB input - Set A

Model Error Metrics	
Mean Absolute Error, MAE [bar]	0.8830
Mean Absolute Percentage Error, MAPE [%]	0.7480
R^2	0.999120

Table 4.7: Error metrics of the best model, for STB input - Set A

Speed-Torque-Lambda-BSFC (STLB) Models

Finally, for networks that used every available input data, specifically Speed-Torque-Lambda-BSFC, an accuracy value of 99.29% was reached. The error metrics and parameters are shown in Table 4.9 and 4.8 respectively.

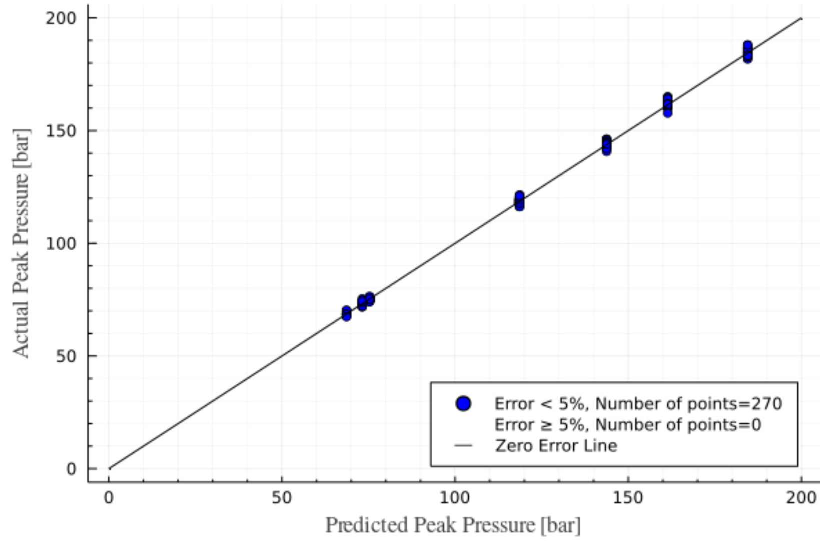

Figure 4.4: The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda-BSFC as input.

Model Parameters	
Hidden Layers	3
Number of Neurons	45 54 51
Optimizer	Descent
Learning Rate	0.001
Activation function	leakyReLU
Epochs	500
Batchsize	4

Table 4.8: Hyperparameters of the best model, for STLB input - Set A

Model Error Metrics	
Mean Absolute Error, MAE [bar]	0.8502
Mean Absolute Percentage Error, MAPE [%]	0.7074
R^2	0.999147

Table 4.9: Error metrics of the best model, for STLB input - Set A

4.1.2 Model Set A Summary

This particular model set achieved quite high accuracy values; this was to be expected since the training dataset was both small in size and narrow in range. As a result, most models fitted to these data points faired better at predicting the target values of the test subset.

Between the best models of the four separate input variable combinations, as presented in Table 4.1, the one that performed the best was the Speed-Torque-Lambda group, with the Speed-Torque one taking the second place by just 0.01% lower accuracy. On the other hand, the most proficient model of the Speed-Torque-BSFC category scored the lowest, albeit having just under 0.07% lower prediction accuracy comparatively to the Speed-Torque-Lambda one. This indicates that the Specific Fuel Consumption parameter shares a weak relation to the in-cylinder peak pressure, or at the very least weaker than the one between the peak pressure and Lambda coefficient.

Models of this group that utilized a small batchsize, equal to 1, 2 or 4 for example, in conjunction with the Descent optimizer produced the best results; as the batchsize increased, the effectiveness of the optimizer rapidly deteriorated. On the other hand, similarly good predictions were also made with models that combined the larger batchsizes of 16, 32 or 64 with the ADAM optimizer. This behaviour of the two optimizers is depicted in figures 4.5 to 4.7; the model parameters that were used for these figures were the ones presented in Table 4.4, since it was the most accurate of this group, however similar outcomes are received for most other models.

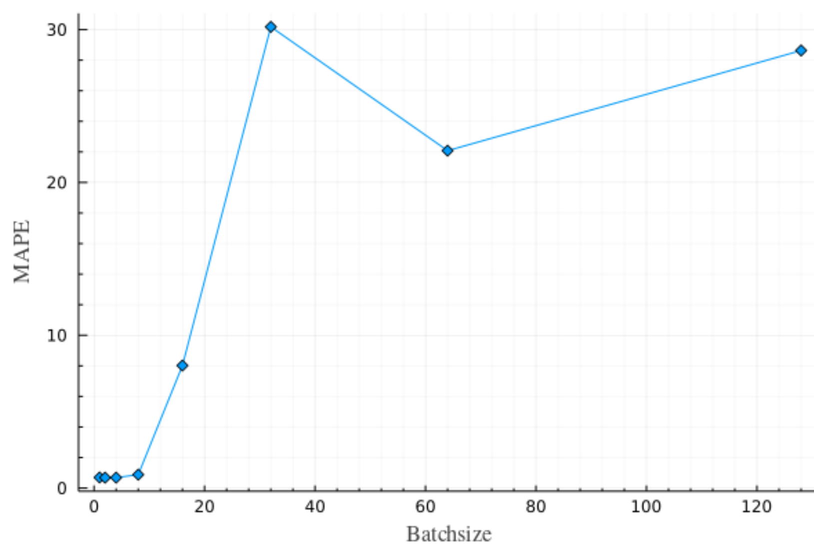

Figure 4.5: The MAPE of the best model of the Set A with the Descent optimizer, for various epochs and batchsizes.

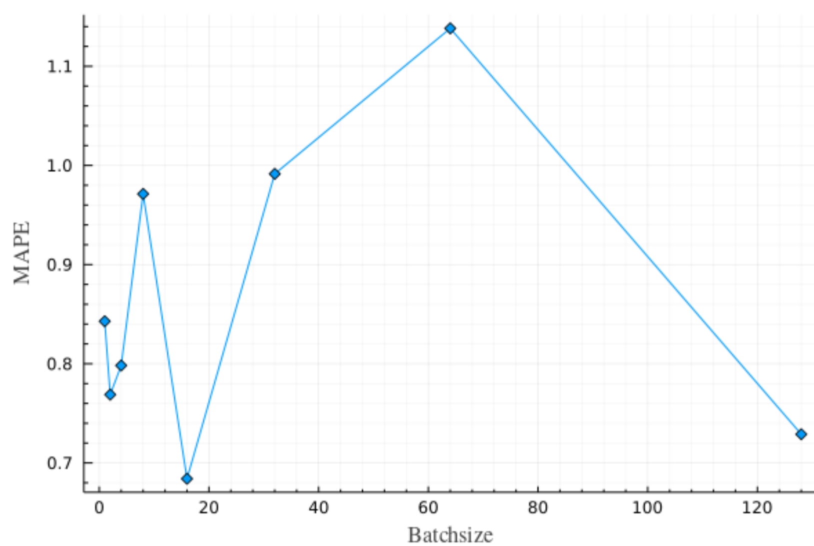

Figure 4.6: The MAPE of the best model of Set A with the ADAM optimizer, for various epochs and batchsizes.

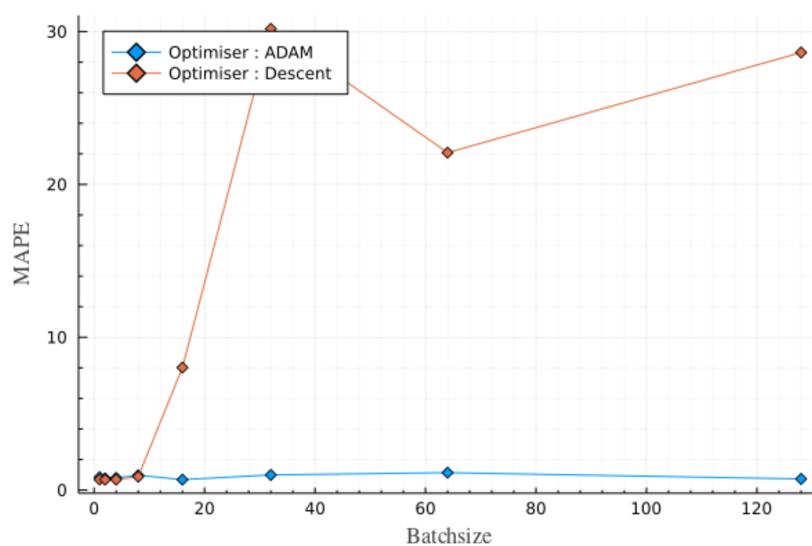

Figure 4.7: Comparison of the Descent and ADAM optimizer of the best Set A model, for various batchsizes.

In addition, while some of the top models of this group were ones with 4 or 5 layers and a large number of neurons, smaller in depth models scored equally well; this is further proof that due to the simplicity of Dataset A, lighter networks were sufficient for precise mapping of the data. For example, Figure 4.8 depicts the variation of the MAPE of the best group A model, as described in Table 4.4.

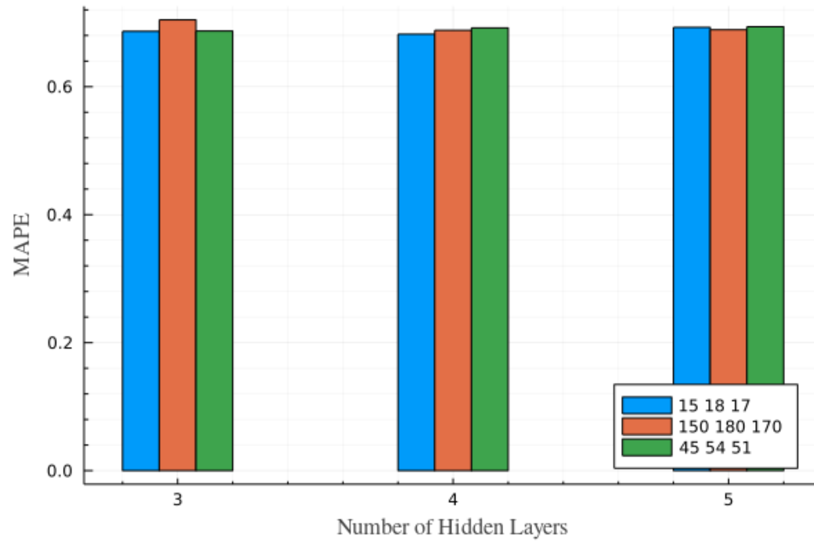

Figure 4.8: Comparison of variations to the number of hidden layers and neurons for the best model of Set A.

Note that in the above Figure 4.8, for the model with 4 and 5 hidden layers, the number of neurons is the one shown in the legend, but with the last number repeated once and twice respectively. For example, for a set of neurons equal to 15 18 17, the 4-hidden layered model has 15 18 17 17 neurons and the 5-hidden layered model 15 18 17 17 17.

On the other hand, varying the learning rate from 0.001 to 0.01 and 0.1 yields increasingly poorer results, as shown in Figure 4.9. Similarly, any other option of activation function, specifically sigmoid (σ) or tanh, also produced models with less predictive power, indicating that the leakyReLU function is the optimal choice, which is highlighted in Figure 4.10.

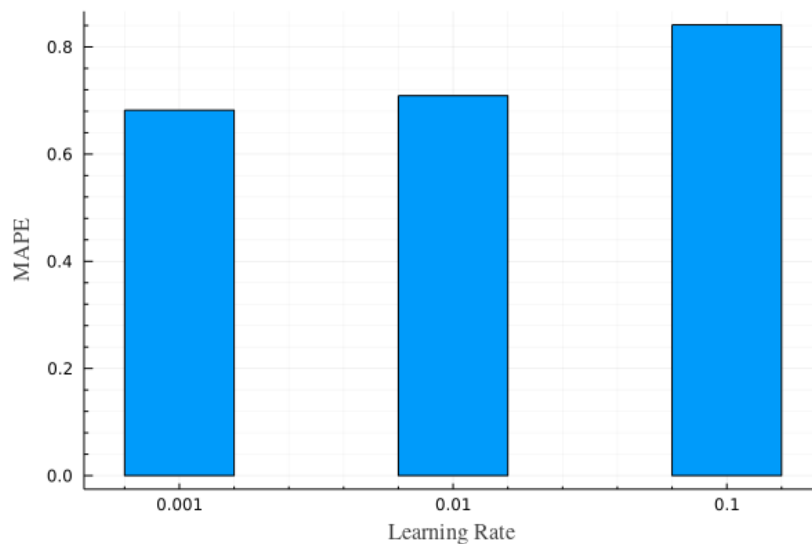

Figure 4.9: Comparison of different learning rates for the best model of Set A.

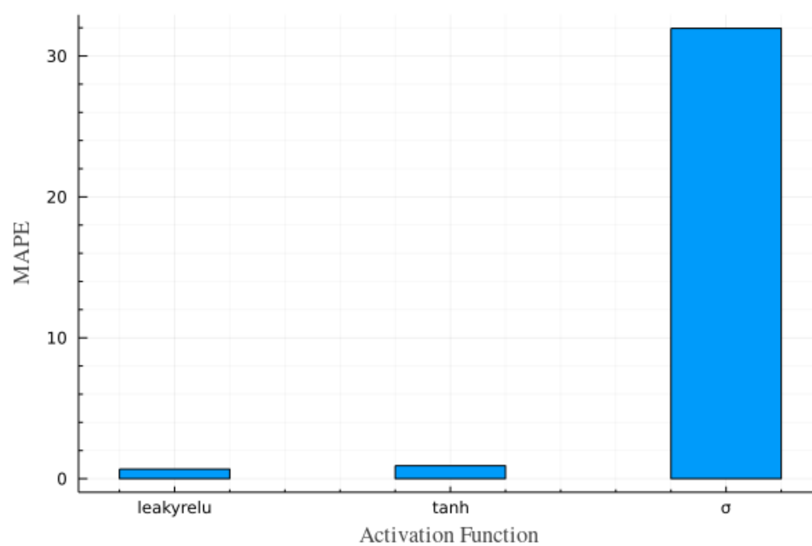

Figure 4.10: Evaluation of different activation functions for the best model of Set A.

Finally, when examining the different training epochs options, its becomes clear that 100 epochs were not enough for any network to fit the data input. However, there aren't any significant discrepancies 500 and 1000 iterations,

both of which producing mostly equally acceptable results, as is highlighted by Figure 4.11.

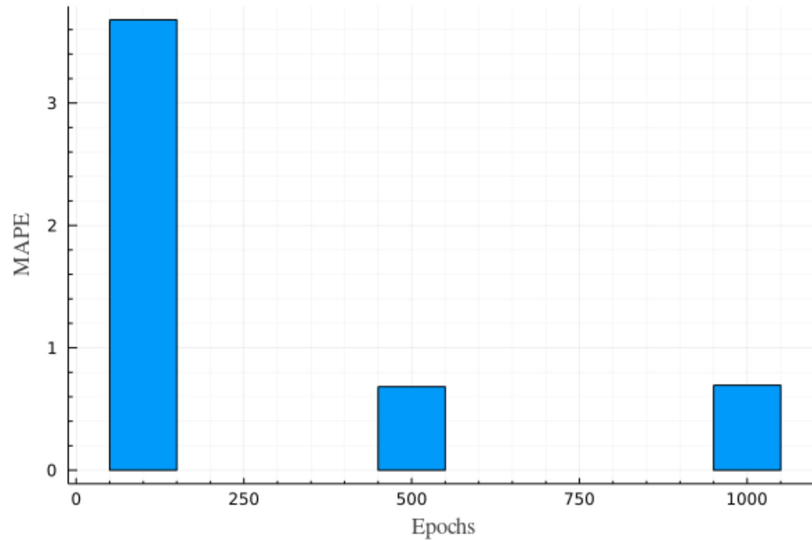

Figure 4.11: Comparison of different training epochs for the best model of Set A

To sum up, the best model of this group is the one that used the Speed-Torque-Lambda input vector and is defined by the hyperparameters of Table 4.4. It is important to note, however, that for someone to receive subjective and accurate predictions using this model, the input data provided will have to fall inside the range on which the training process took place, as described by Table 3.1. Any input that deviates too far from the aforementioned table will produce poor outcomes.

The rest of the diagrams that show the effect of variations to the hyperparameters of the models on the prediction accuracy are in Appendix A.

4.2 Model Set B

In this Model Set B, as opposed to the Set A, models were trained with two variations of the input parameters: one group used Speed and Torque (ST) as input, while the other Speed, Torque and Lambda (STL).

Table 4.10 contains the best models of the two groups, while the following section gives more details on their error metrics and parameters.

Model Name	Inputs	Accuracy
Act leakyrelu-Lr0.001-Opt ADAM-H.Layers 5-Neurons 150 180 170 170 170-VarST-Ep1000-Btch128	ST	95.21%
Act leakyrelu-Lr0.001-Opt ADAM-H.Layers 5-Neurons 150 180 170 170 170-VarSTL-Ep1000-Btch64	STL	97.04%

Table 4.10: Comparison of the best models for each combination of input variables - Set B

4.2.1 Best performing networks, per Set of Inputs

Speed-Torque (ST) Models

For Set B models that used the Speed-Torque vector as training input, an accuracy of 95.21% was reached. The error metrics are presented in 4.12, while its hyperparameters in Table 4.11.

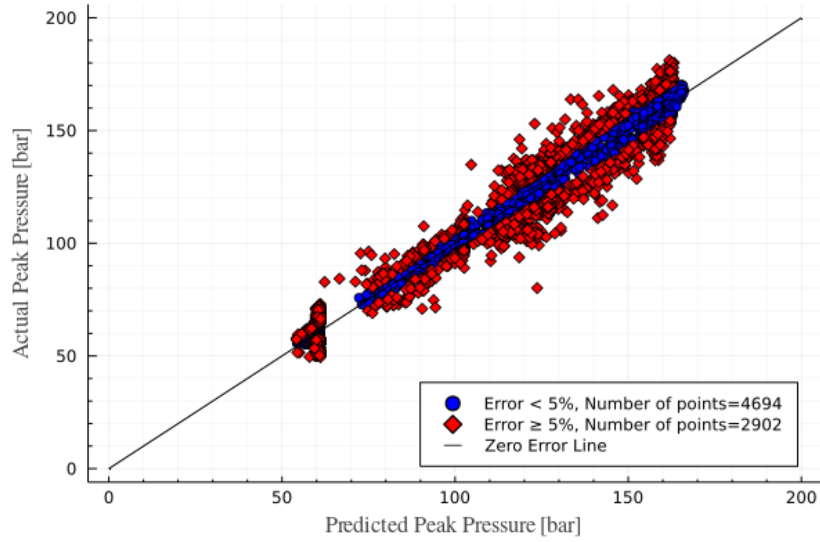

Figure 4.12: The zero-error-line diagram of the best performing model of Set B, with Speed-Torque as input.

Model Parameters	
Hidden Layers	5
Number of Neurons	150 180 170 170 170
Optimizer	ADAM
Learning Rate	0.001
Activation function	leakyReLU
Epochs	1000
Batchsize	128

Table 4.11: Hyperparameters of the best model, for ST input - Set B

Model Error Metrics	
Mean Absolute Error, MAE [bar]	4.984
Mean Absolute Percentage Error, MAPE [%]	4.7930
R^2	0.962679

Table 4.12: Error metrics of the best model, for ST input - Set B

Speed-Torque-Lambda (STL) Models

Regarding networks that used the Speed-Torque-Lambda vector as training input, the model accuracy was 97.04%; making it the best overall model of this Set. In Table 4.14 the error metrics are presented, and the model parameters in Table 4.13.

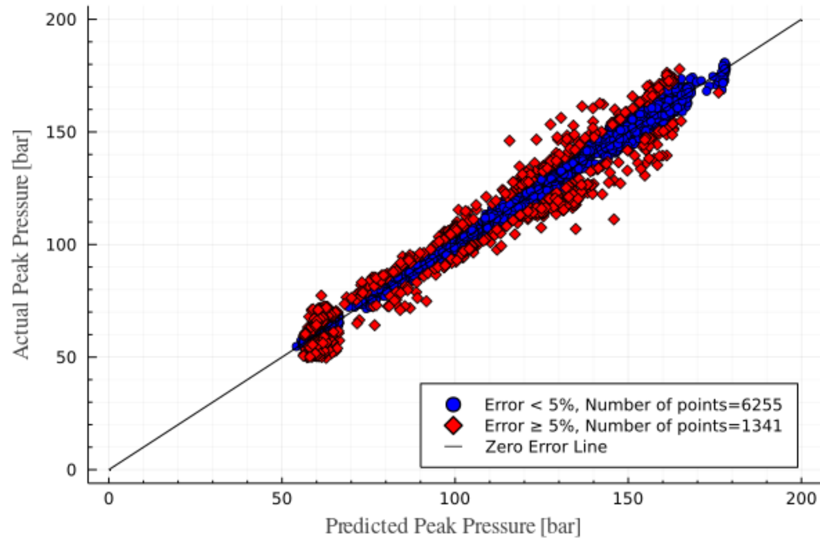

Figure 4.13: The zero-error-line diagram of the best performing model of Set A, with Speed-Torque-Lambda as input.

Model Parameters	
Hidden Layers	5
Number of Neurons	150 180 170 170 170
Optimizer	ADAM
Learning Rate	0.001
Activation function	leakyReLU
Epochs	1000
Batchsize	64

Table 4.13: Hyperparameters of the best model, for STL input - Set B

Model Error Metrics	
Mean Absolute Error, MAE [bar]	2.8221
Mean Absolute Percentage Error, MAPE [%]	2.9602
R^2	0.985495

Table 4.14: Error metrics of the best model, for STL input - Set B

4.2.2 Model Set B Summary

In contrast to Model Set A, this group of models reached a lower peak accuracy; nevertheless, with the mean percentage error being less that 3%, the results can undoubtedly be considered satisfactory. The cause behind the lowered accuracy when compared to Set A is the characteristics of the Dataset B; as explained in Section 3.1.2 of Chapter 2, the data spectrum is extensive and the points are spread out considerably. As a result, it is harder for the network to fit the data with extremely high precision, but at the same time the produced model will certainly boast superior robustness, as it has been exposed to a varied input range and thus can react better for a larger test subset.

Out of the two different input variables combinations, the Speed-Torque-Lambda one scored the lowest average error, as shown in Table 4.10; the one that used only the Speed-Torque inputs fell behind by almost 2%. This yet again shows the strong relation between the lambda coefficient with the peak internal pressure, a conclusion that was reached with the previous Set A. Also, based on the results of the previous group, the lack of Specific Fuel Consumption values most probably didn't impact negatively the prediction power of the models, as it was shown that the inclusion of BSFC worsened the outcome.

This model set used only the ADAM optimizer; similarly to the Set A, models with higher batchsize values produced more accurate prediction than their smaller-batchsize counterparts. Figure 4.14 demonstrates this property for the best performing model described in Table 4.13.

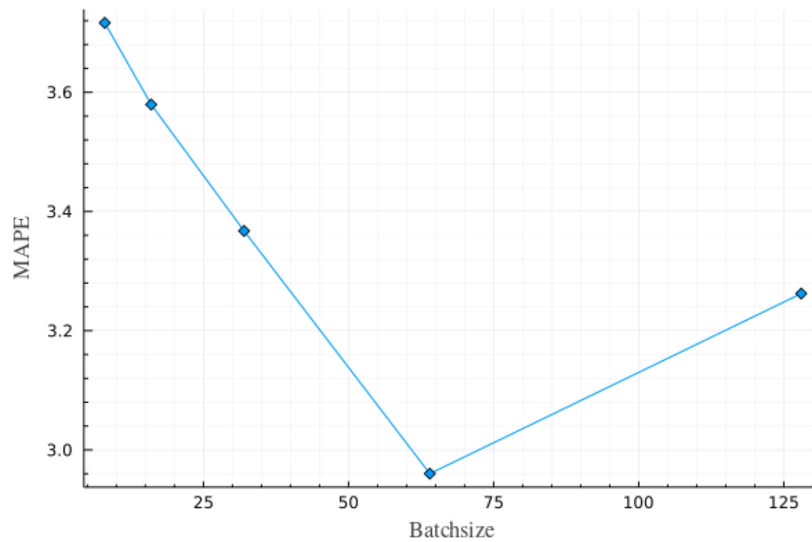

Figure 4.14: The MAPE of the best model of Set B with the ADAM optimizer, for various batchsizes.

Regarding the number of hidden layers and neurons per layer, contrary to the previous set of models, the upper echelon of Set A was populated almost exclusively with "heavier" networks. The best performing networks incorporated in their design a large amount of neurons, while most of them also leaned towards the 4 or 5 hidden layers. Once again, due to the extensive size of the Dataset B, numerous processing units are required to correctly map the input data. This behaviour is showcased in the following Figure 4.15.

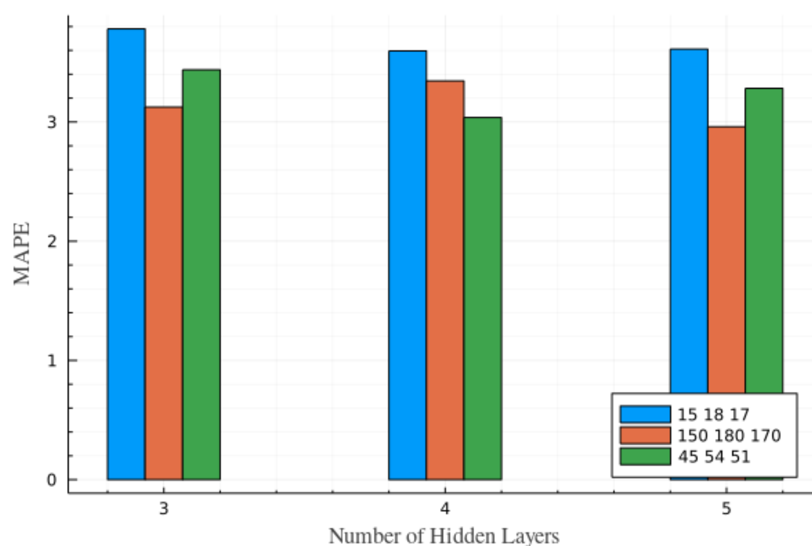

Figure 4.15: Comparison of variations to the number of hidden layers and neurons for the best model of Set B.

Lastly, the number of training epochs had a significant impact on the predictive capabilities of the models; the higher the epochs count, the more accurate the predictions became. The best results were almost always received at the highest epochs number: 1000, as the figure bellow shows.

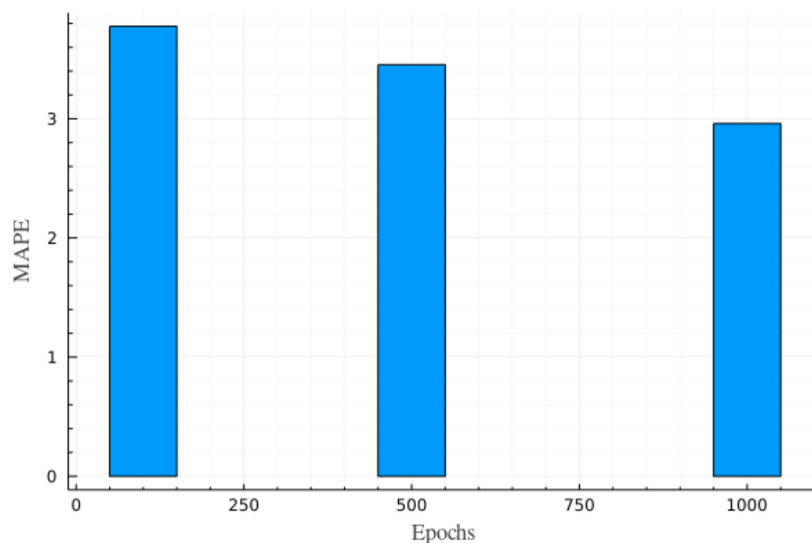

Figure 4.16: Comparison of different training epochs for the best model of Set B

Summing up for the Model Set B, the highest accuracy reached was 97.04%, which belonged to the Speed-Torque-Lambda input subset of models, with the best overall being the one as defined by the hyperparameters of Table 4.13. The average error values of this group were higher than the ones in Model Set A, something that can be ascribed to the complexity of Dataset B, however the results are still well within the acceptable error area. As mentioned for the previous set, it is once again important to note that for the model to make good predictions in a possible requested task, the input variables must be inside, or adequately close, to the range on which it was trained, shown by the diagrams 3.10 to 3.12.

Once again, the rest of the diagrams are presented in Appendix B.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

The ever-increasing interest in the fields of engineering for optimized operation of internal combustion engines has inevitably lead to the search for better engine control methods. In order to ensure that optimal performance is followed, more and more sophisticated measuring tools are being developed; the ideal method being one that combines inexpensive procedures with accurate and trustworthy results.

To this end, this thesis took on the task of predicting the in-cylinder peak pressure, an important parameter of the combustion process and structural stress of an working engine, of a four-stroke marine diesel engine. Specifically, the viability of utilizing the latest trends in machine learning, in order to produce algorithms that can be further put to use for such predictive endeavours, was investigated.

Thus, artificial neural networks were employed; models that were tasked to effectively predict the aforementioned pressure based on other, easy to acquire engine operating parameters; in essence, the engine Speed, Torque, Lambda and Brake-Specific Fuel Consumption (BSFC) were the used input variables.

Two groups of artificial neural networks were created and trained on two distinct datasets, extracted from experimental operations of two different four-stroke diesel engines: Model Set A and Model Set B. The outcome showed that both groups managed to reach the designated goal: group A achieved pressure prediction with accuracy of 99.32% and mean absolute error of 0.83 bar, while the second group B reached a comparatively lower, but nonetheless satisfactory, score of 97.04% and mean absolute error of 2.8 bar.

Between the input variables combinations that were tested, it was remarked that by using as input the engine Speed, Torque and Lambda, the most precise predictions were produced; on the other hand, incorporating the Specific Fuel Consumption parameter as input not only didn't improve the model accuracy, but it also hindered the learning process.

Regarding the hyperparameters of the models, during their training stage several trends were observed. Models that used the ADAM optimizer in conjunction with a large batchsize produced the best results; additionally the optimal learning rate was found to be equal to 0.001. Large numbers of neurons and multiple hidden layers were deemed necessary when dealing with expansive datasets, similar to Dataset B, while smaller in size models were acceptably good in cases of smaller datasets, like Dataset A; likewise, a sufficiently large number of epochs is needed to correctly fit broad sets of data. Lastly, out of the three activation function that were investigated, the leakyReLU counterpart of the ReLU function faired the best.

Consequently, by the successful results of the models of this thesis, it has been proven that artificial neural networks lend themselves to be utilized as a cheap alternative for in-cylinder peak pressure prediction in internal combustion engines.

5.2 Future Work

In this thesis a number of artificial neural networks that fell into the category of deep learning were used to predict in-cylinder peak pressure; the same concept behind these networks could be used to build models for predicting different engine parameters, like peak cylinder temperature, for instance.

Furthermore, different types of machine learning models could be investigated for use in similar tasks; Support Vector Regressors, for example, could be one of such models.

Finally, developing models for pure transient loads could also be a possible subject of research. For this type of tasks, were predictions of continuous signals is required, recurrent neural networks could be a suitable network type, as they keep the information of previous states and use it to predict subsequent ones.

Bibliography

[1] M. El-Ghamry, J. Steel, R. Reuben, and T. Fog, "Indirect measurement of cylinder pressure from diesel engines using acoustic emission," *Mechanical Systems and Signal Processing*, vol. 19, no. 4, pp. 751–765, 2005.

[2] D. Moro, N. Cavina, and F. Ponti, "In-Cylinder Pressure Reconstruction Based on Instantaneous Engine Speed Signal ," *Journal of Engineering for Gas Turbines and Power*, vol. 124, pp. 220–225, 03 2001.

[3] J. Heywood, *Internal Combustion Engine Fundamentals 2E.* McGraw-Hill Education, 2018.

[4] C. Bishop, *Pattern Recognition and Machine Learning*, ch. 1,3,4,5. New York, NY: Springer, 2006.

[5] E. Alpaydın, *Introduction to Machine Learning, Third Edition.* MIT Press, 2014.

[6] S. J. Russell and P. Norvig, *Artificial Intelligence a Modern Approach, Third Edition*, ch. 20.4. Pearson, 2010.

[7] Y. Fu and B. Xiao, "Online prediction of the piston maximum temperature in dual-fuel engine," *Advances in Mechanical Engineering*, vol. 9, no. 2, 2017.

[8] Y. Çay, "Prediction of a gasoline engine performance with artificial neural network," *Fuel*, vol. 111, p. 324–331, 09 2013.

[9] Y. Çay, I. Korkmaz, A. Çiçek, and F. Kara, "Prediction of engine performance and exhaust emissions for gasoline and methanol using artificial neural network," *Energy*, vol. 50, no. C, pp. 177–186, 2013.

[10] V. Cocco Mariani, S. Hennings Och, L. dos Santos Coelho, and E. Domingues, "Pressure prediction of a spark ignition single cylinder

engine using optimized extreme learning machine models," *Applied Energy*, vol. 249, pp. 204–221, 2019.

[11] H. Yaşar, G. Çağıl, O. Torkul, and M. Şişci, "Cylinder pressure prediction of an hcci engine using deep learning," *Chinese Journal of Mechanical Engineering*, vol. 34, January 2021.

[12] N. J. Nilsson, *Introduction to Machine Learning*, pp. 1–3. Stanford, CA: Robotics Laboratory Department of Computer Science Stanford University, 1998.

[13] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning, second edition.* Adaptive Computation and Machine Learning series, MIT Press, 2018.

[14] Vortarus Technologies LLC, "Simulation vs. machine learning." `https://vortarus.com/simulation-vs-machine-learning/`. Accessed: 2021-12-10.

[15] Copeland, B.J , "Alan Turing," *Encyclopedia Britannica*, 2021. Accessed: 2021-12-10.

[16] A. M. Turing, "I.—COMPUTING MACHINERY AND INTELLIGENCE," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[17] The Editors of Encyclopaedia Britannica, "Turing test," *Encyclopedia Britannica*, 2020. Accessed: 2021-12-11.

[18] J. McCarthy and E. A. Feigenbaum, "In memoriam: Arthur samuel: Pioneer in machine learning," *AI Magazine*, vol. 11, no. 3, 1990.

[19] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, pp. 71–105, 1959.

[20] Foote, Keith D. , "A brief history of machine learning." `https://www.dataversity.net/a-brief-history-of-machine-learning/`. Accessed: 2021-12-10.

[21] B. Swaminathan, R. Vaishali, and T. S. R. subashri, *Analysis of Minimax Algorithm Using Tic-Tac-Toe*, pp. 528–532. IOS Press, 2020.

[22] F. Rosenblatt, "THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[23] G. B. Ronsivalle, S. Carta, V. Metus, and M. Orlando, "Artificial neural networks, evaluation and complexity: Information technology and nonlinear algorithms to measure knowledge systems," in *INTED2014 Proceedings*, pp. 4175–4185, IATED, 2014.

[24] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Buffalo 21, New York: Spartan Books, 1961.

[25] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, p. 85–117, Jan 2015.

[26] M. Hema R and I. Alexy G, *Inductive Learning Algorithms for Complex Systems Modeling.* Boca Raton, Florida: CRC Press, Inc, 1994.

[27] Langley, Pat, "The changing science of machine learning," *Machine Learning*, vol. 82, no. 3, p. 275, 2011.

[28] Les Earnest , "Stanford cart." `https://web.stanford.edu/~learnest/sail/oldcart.html`. Accessed: 2021-12-11.

[29] Sejnowski, Terrence J. and Rosenberg, Charles R. , "Parallel networks that learn to pronounce english text," *Complex Systems*, no. 1, pp. 145–168, 1967.

[30] Greenemeier, Larry , "20 Years after Deep Blue: How AI Has Advanced Since Conquering Chess." `https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess/`. Accessed: 2021-12-11.

[31] IBM, "The first meeting between kasparov and deep blue made chess history." `https://web.archive.org/web/20081212043535/http://www.research.ibm.com/deepblue/watch/html/c.10.html`. Accessed: 2021-12-12.

[32] IBM, "Deep blue." `https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/`. Accessed: 2021-12-12.

[33] "About imagenet." `https://www.image-net.org/about.php`. Accessed: 2021-12-12.

[34] Gershgorn, Dave, "The data that transformed ai research—and possibly the world." `https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/`, 2017. Accessed: 2021-12-12.

[35] Rodriguez, Raul V. , "Kinect sensor: The ai tool you did not know you had." `https://analyticsindiamag.com/kinect-sensor-the-ai-tool-you-did-not-know-you-had/`, 2020. Accessed: 2021-12-12.

[36] Nath, Aditya, "What is google brain?." `https://www.geeksforgeeks.org/what-is-google-brain/`, 2021. Accessed: 2021-12-12.

[37] Simonite, Tom, "Facebook creates software that matches faces almost as well as you do." `https://www.technologyreview.com/2014/03/17/13822/facebook-creates-software-that-matches-faces-almost-as-well-as-you-do/`, 2014. Accessed: 2021-12-12.

[38] Nath, Aditya, "Google achieves ai 'breakthrough' by beating go champion." `https://www.bbc.com/news/technology-35420579`, 2016. Accessed: 2021-12-13.

[39] "Deepmind." `https://deepmind.com/about`. "Accessed: 2021-12-13".

[40] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[41] M. Kulin, T. Kazaz, I. Moerman, and E. De Poorter, "A survey on machine learning-based performance improvement of wireless networks: Phy, mac and network layer," *Electronics*, p. 63, 2020.

[42] M. Haenlein and A. Kaplan, "A brief history of artificial intelligence: On the past, present, and future of artificial intelligence," *California Management Review*, vol. 61, 2019.

[43] O. Campesato, *Artificial Intelligence Machine Learning and Deep Leraning*. Mercury Learning and Information LLC, 2020.

[44] V. Sindhu, S. Nivedha, and M. Prakash, "An empirical science research on bioinformatics in machine learning," *JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES*, pp. 86–94, 2020.

[45] IBM Cloud Education, "Deep learning." `https://www.ibm.com/cloud/learn/deep-learning`. Accessed: 2021-12-13.

[46] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[47] D. Bzdok, N. Altman, and M. Krzywinski, "Statistics versus machine learning," *Nature Methods*, vol. 15, no. 4, pp. 233–234, 2018.

[48] M. Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," *Reinforcement Learning: State of the Art*, pp. 3–42, 2012.

[49] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," 2014.

[50] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, vol. 52, p. 67, 07 2010.

[51] C. Gallo, *Artificial Neural Networks: tutorial*, vol. 10, ch. Neural Netorks, pp. 179–189. IGI Global, 2015.

[52] S. Agatonovic-Kustrin and R. Beresford, "Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 22, no. 5, pp. 717–727, 2000.

[53] TIBCO, "What is a Neural Network?." `https://www.tibco.com/reference-center/what-is-a-neural-network`. Accessed: 2021-12-24.

[54] R. Bridgelall, "Lecture notes, introduction to support vector machines." `https://www.ugpti.org/smartse/resources/downloads/support-vector-machines.pdf`, 2017. Accessed: 2021-12-21.

[55] V. Jakkula, "Tutorial on support vector machine (svm)," *School of EECS, Washington State University*, vol. 37, 2006.

[56] P. P. Ippolito, "SVM: Feature Selection and Kernels." `https://towardsdatascience.com/svm-feature-selection-and-kernels-840781cc1a6c`, 2019. Accessed: 2021-12-22.

[57] L. Rokach and O. Maimon, *Decision Trees*, vol. 6, pp. 165–192. Springer, Boston, MA, 2005.

[58] B. Jijo and A. Mohsin Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, pp. 20–28, 2021.

[59] M. Hornỳ, "Bayesian networks," *Boston University School of Public Health*, vol. 17, 2014.

[60] A. N. of Loc Nguyen, "Overview of bayesian network," *Science Journal of Mathematics & Statistics*, vol. 2013, pp. 1–99, 07 2013.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[62] "The Julia Programming Language." `https://julialang.org/`.

[63] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Learn Julia For Beginners – The Future Programming Language of Data Science and Machine Learning Explained." `https://julialang.org/blog/2012/02/why-we-created-julia/`, 2012. Accessed: 2022-6-2.

[64] L. Kilpatrikc, "Learn Julia For Beginners – The Future Programming Language of Data Science and Machine Learning Explained." `https://www.freecodecamp.org/news/learn-julia-programming-language/`, 2021. Accessed: 2022-6-2.

[65] V. Singh Rao, "Julia Programming Language – A True Python Alternative." `https://www.technotification.com/2018/08/julia-programming-language.html`, 2018. Accessed: 2022-6-2.

[66] "Flux: The Julia Machine Learning Library." `https://fluxml.ai/Flux.jl/stable/`. Accessed: 2022-6-2.

[67] "CUDA.jl." `https://juliagpu.gitlab.io/CUDA.jl/`. Accessed: 2022-6-2.

[68] "BSON." `https://github.com/JuliaIO/BSON.jl`. Accessed: 2022-6-2.

[69] R. Salehi and A. Stefanopoulou, "Effective component tuning in a diesel engine model using sensitivity analysis," p. 8, 10 2015.

[70] I. Hand, Mike J., E. Hellström, D. Kim, A. Stefanopoulou, J. Kollien, and C. Savonen, "Model and Calibration of a Diesel Engine Air Path With an Asymmetric Twin Scroll Turbine," vol. Volume 1: Large Bore Engines; Advanced Combustion; Emissions Control Systems; Instrumentation, Controls, and Hybrids of *Internal Combustion Engine Division Fall Technical Conference*, 10 2013.

[71] A. Stefanopoulou, I. Kolmanovsky, and J. Freudenberg, "Control of variable geometry turbocharged diesel engines for reduced emissions," *IEEE Transactions on Control Systems Technology*, vol. 8, no. 4, pp. 733–745, 2000.

[72] M. D. . Turbo, "L16/24 Project Guide - Marine Four-stroke GenSet compliant with IMO Tier II."

[73] "Test Cell." `https://www.lme.ntua.gr/facilities-1/test-cell-1/test-cell`. Accessed: 2022-26-2.

[74] "Engine Room." `https://www.lme.ntua.gr/testcell2.jpg/view`. Accessed: 2022-26-2.

[75] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *Nuclear Science, IEEE Transactions on*, vol. 44, pp. 1464 – 1468, 07 1997.

[76] Y. Verma, "A Complete Understanding of Dense Layers in Neural Networks." `https://analyticsindiamag.com/a-complete-understanding-of-dense-layers-in-neural-networks/`, 2021. Accessed: 2022-1-9.

[77] C.-F. Wang, "The Vanishing Gradient Problem." `https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484`, 2019. Accessed: 2022-1-11.

[78] S. Sharma, "Activation Functions in Neural Networks." `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`, 2017. Accessed: 2022-1-11.

[79] P. Baheti, "12 Types of Neural Network Activation Functions: How to Choose?." `https://www.v7labs.com/blog/neural-networks-activation-functions`, 2022. Accessed: 2022-1-11.

[80] keshav, "Hyperbolic Tangent (tanh) Activation Function [with python code]." `https://vidyasheela.com/post/hyperbolic-tangent-tanh-activation-function-with-python-code`. Accessed: 2022-1-11.

[81] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[82] Y. LeCun, "Activation and loss functions (part 1)." https://atcold.github.io/pytorch-Deep-Learning/en/week11/11-1/, 2020. Accessed: 2022-1-11.

[83] "Build a simple Neural Network with TensorFlow.js — Deep Learning for JavaScript Hackers (Part III)." https://curiousily.com/posts/build-a-simple-neural-network-with-tensorflow-js/, 2019. Accessed: 2022-1-12.

[84] S. Do, K. D. Song, and J. W. Chung, "Basics of deep learning: A radiologist's guide to understanding published radiology articles on deep learning," *Korean J Radio*, vol. 21, 2020.

Appendix A

Model Set A Diagrams

In this Appendix, diagrams depicting the rest of the four best per input models of Set A are presented. They include variations to the model optimizer, batchsize, learning rate, activation function and epochs. This section provides the complimentary to Section 4.1 of Chapter 4 figures of the best ST, STB and STLB models of Set A.

A.1 Speed-Torque (ST) Models

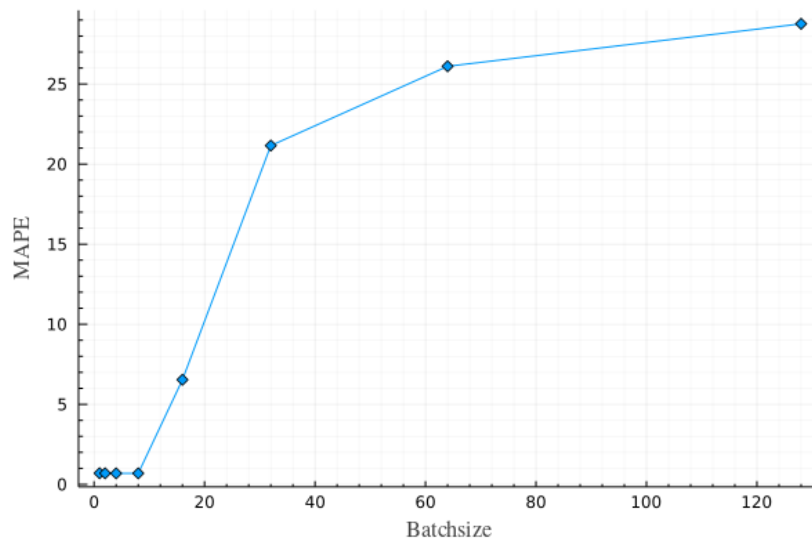

Figure A.1: The MAPE of the best ST model of the Set A with the Descent optimizer, for various epochs and batchsizes.

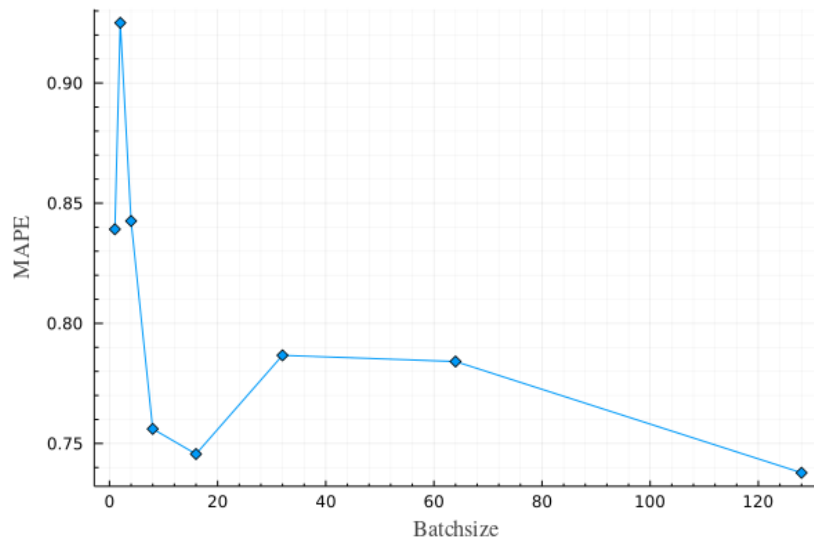

Figure A.2: The MAPE of the best ST model of Set A with the ADAM optimizer, for various epochs and batchsizes.

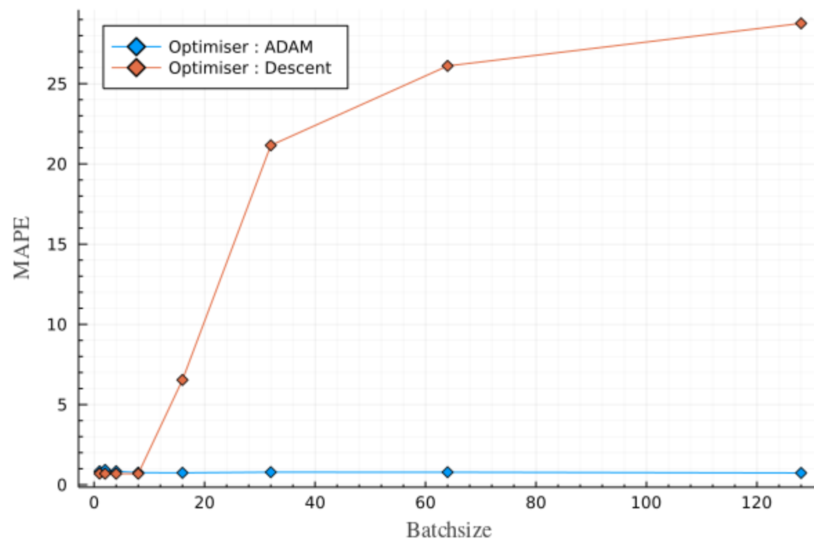

Figure A.3: Comparison of the Descent and ADAM optimizer of the best ST model, for various batchsizes.

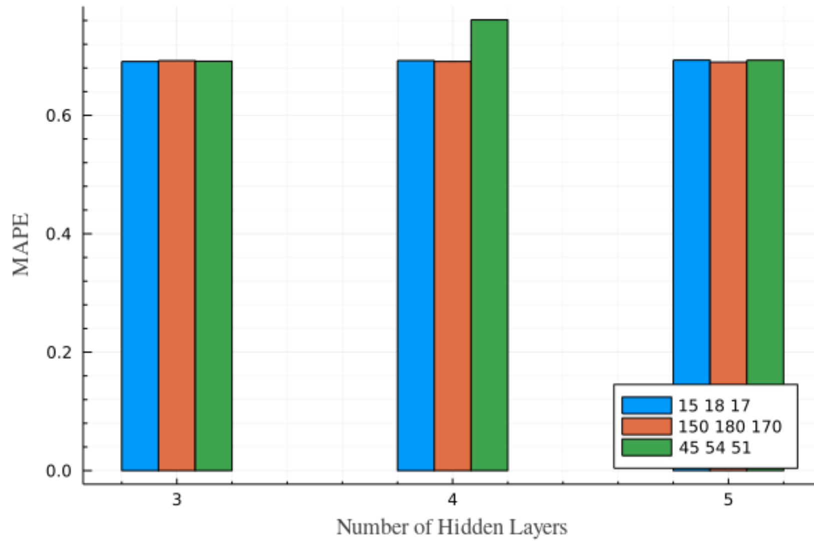

Figure A.4: Comparison of variations to the number of hidden layers and neurons for the best ST model of Set A.

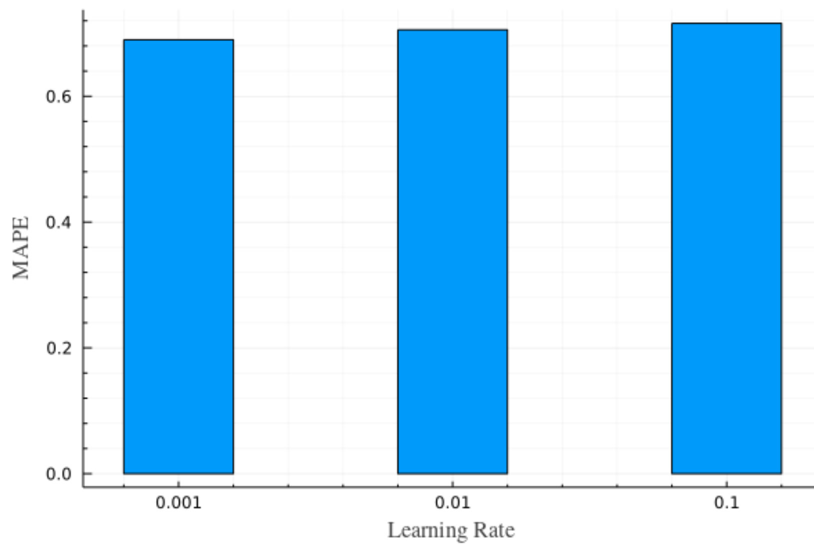

Figure A.5: Comparison of different learning rates for the best ST model of Set A.

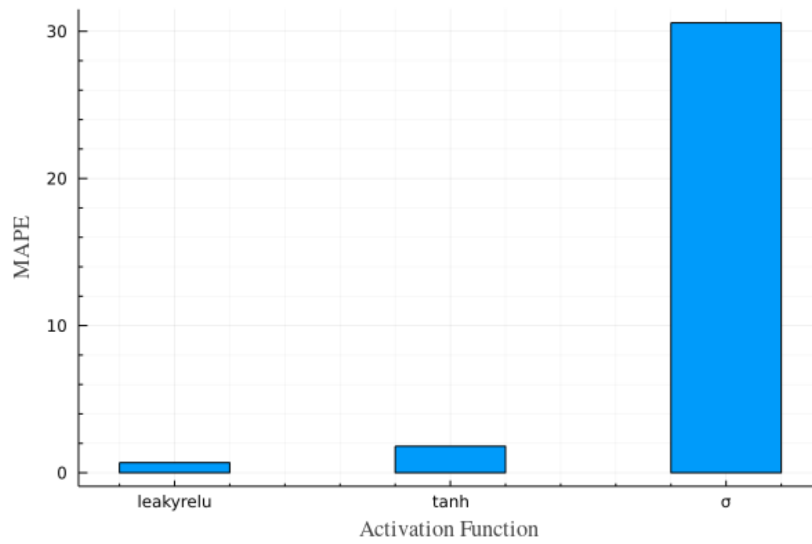

Figure A.6: Evaluation of different activation functions for the best ST model of Set A.

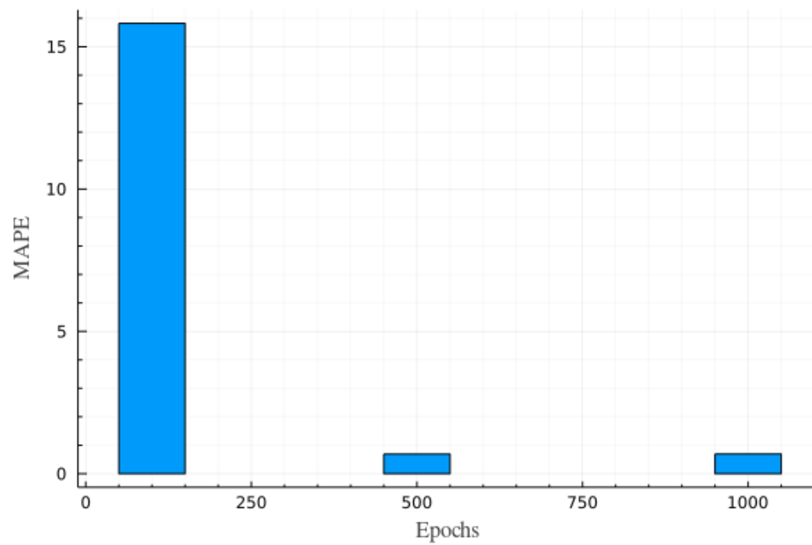

Figure A.7: Comparison of different training epochs for the best ST model of Set A

A.2 Speed-Torque-BSFC (STB) Models

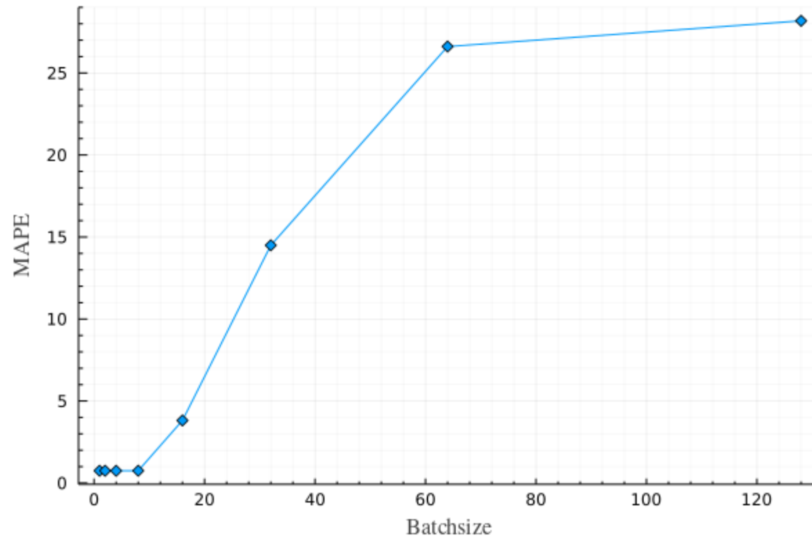

Figure A.8: The MAPE of the best STB model of the Set A with the Descent optimizer, for various epochs and batchsizes.

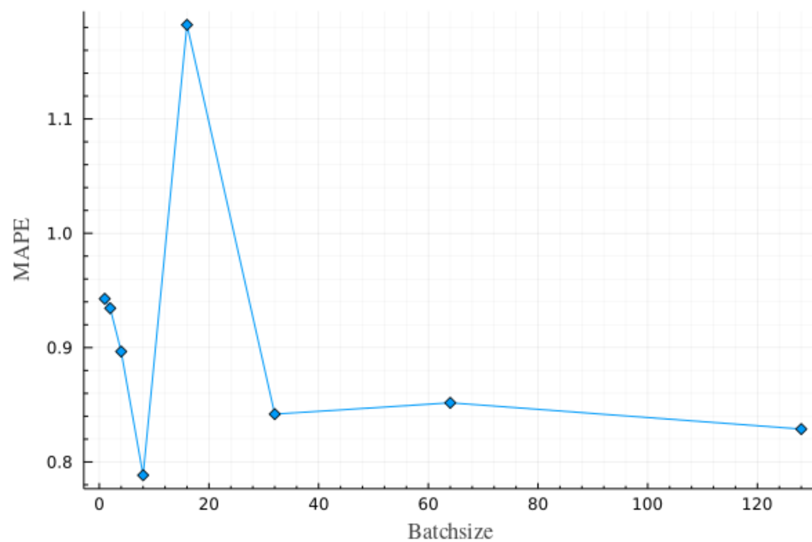

Figure A.9: The MAPE of the best STB model of Set A with the ADAM optimizer, for various epochs and batchsizes.

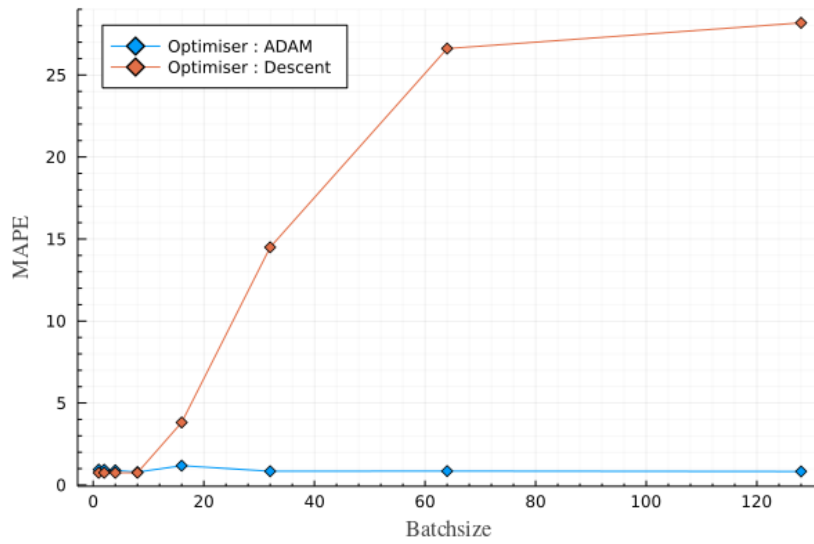

Figure A.10: Comparison of the Descent and ADAM optimizer of the best STB model, for various batchsizes.

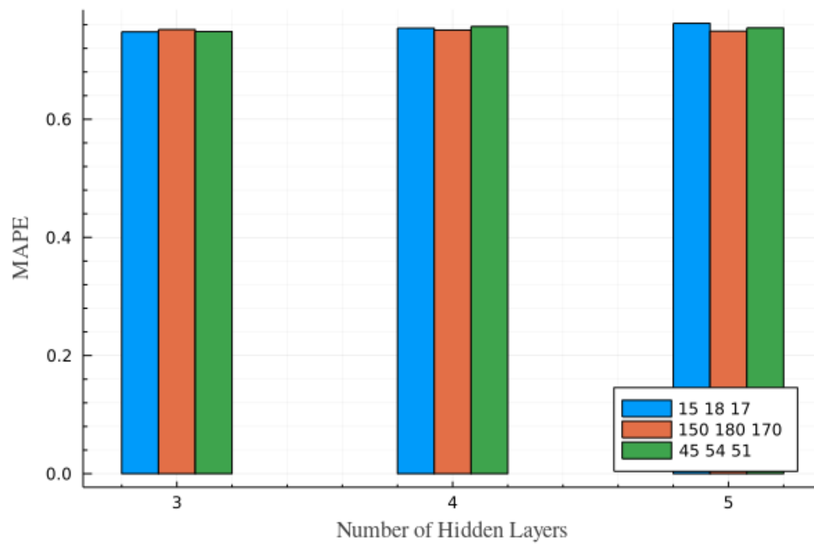

Figure A.11: Comparison of variations to the number of hidden layers and neurons for the best STB model of Set A.

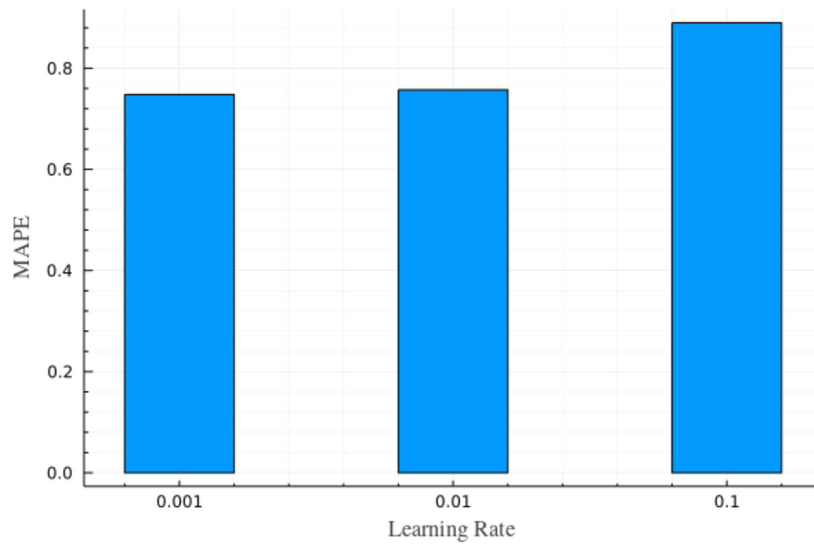

Figure A.12: Comparison of different learning rates for the best STB model of Set A.

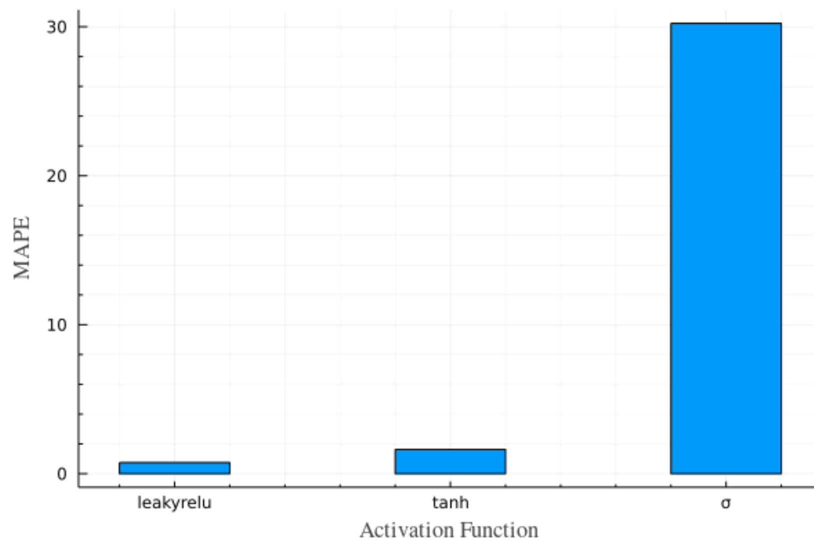

Figure A.13: Evaluation of different activation functions for the best STB model of Set A.

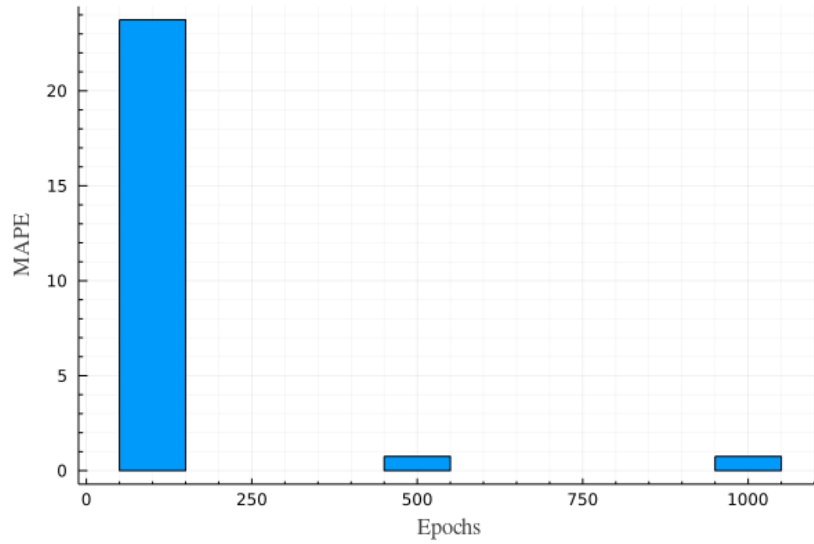

Figure A.14: Comparison of different training epochs for the best STB model of Set A

A.3 Speed-Torque-Lambda-BSFC (STLB) Models

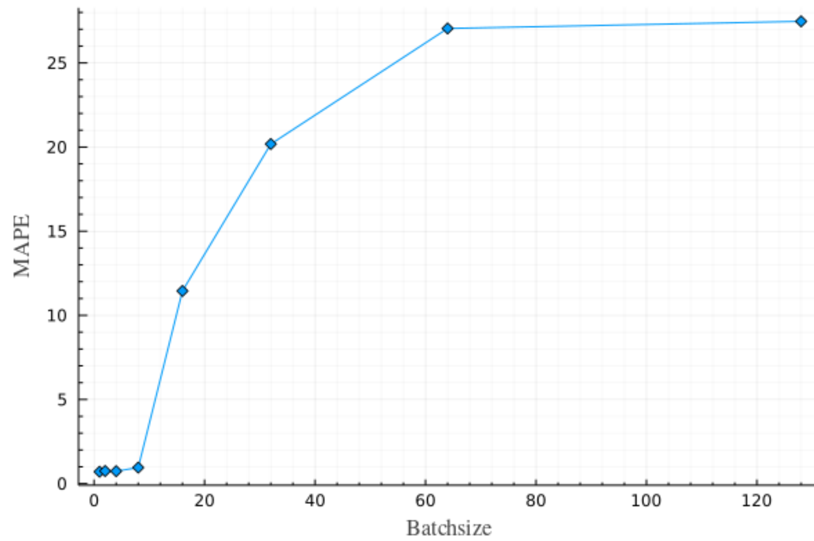

Figure A.15: The MAPE of the best STLB model of the Set A with the Descent optimizer, for various epochs and batchsizes.

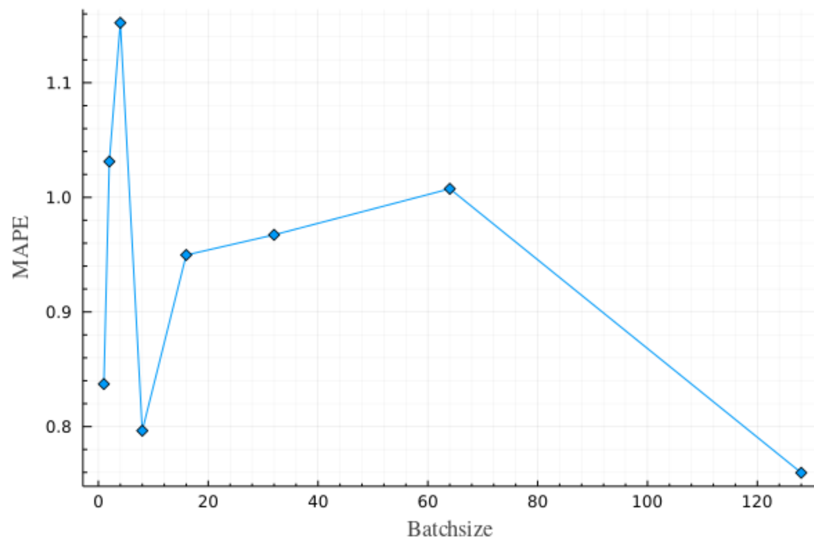

Figure A.16: The MAPE of the best STLB model of Set A with the ADAM optimizer, for various epochs and batchsizes.

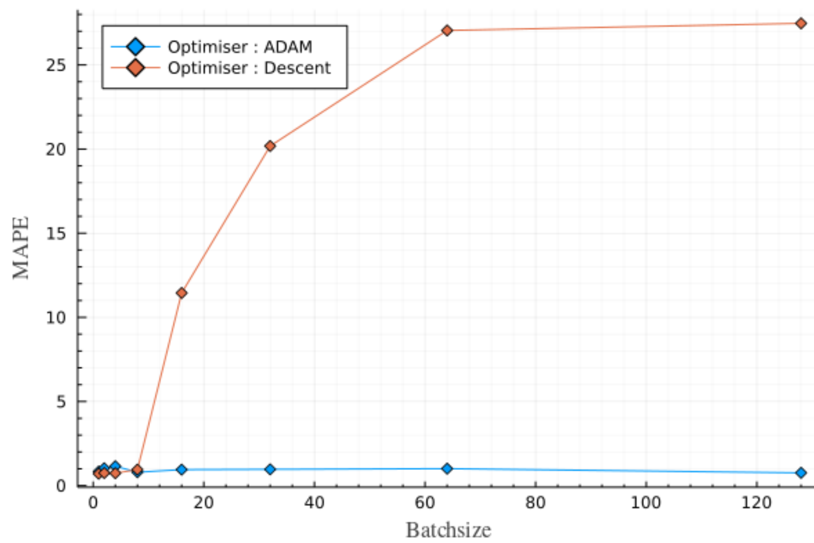

Figure A.17: Comparison of the Descent and ADAM optimizer of the best STLB model, for various batchsizes.

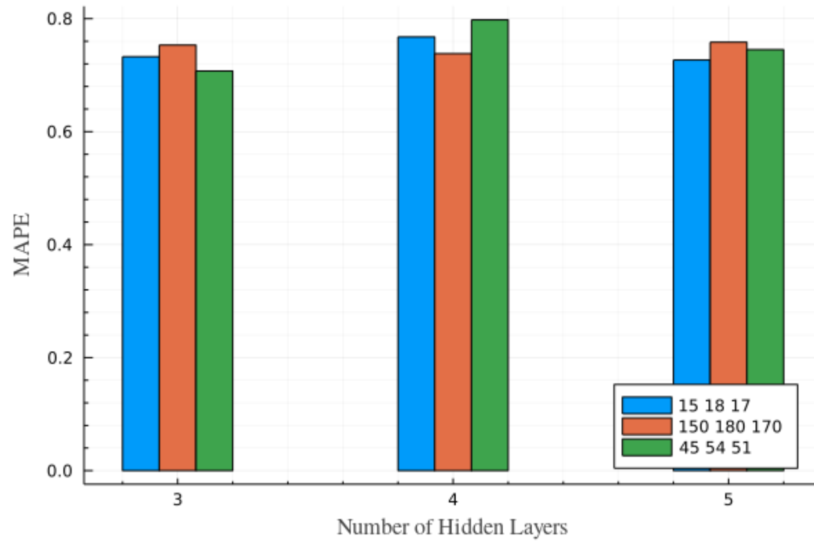

Figure A.18: Comparison of variations to the number of hidden layers and neurons for the best STLB model of Set A.

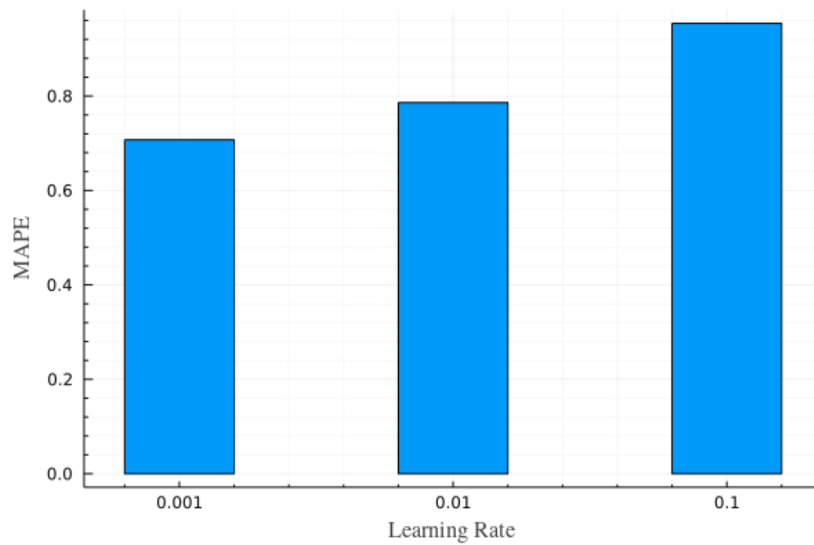

Figure A.19: Comparison of different learning rates for the best STLB model of Set A.

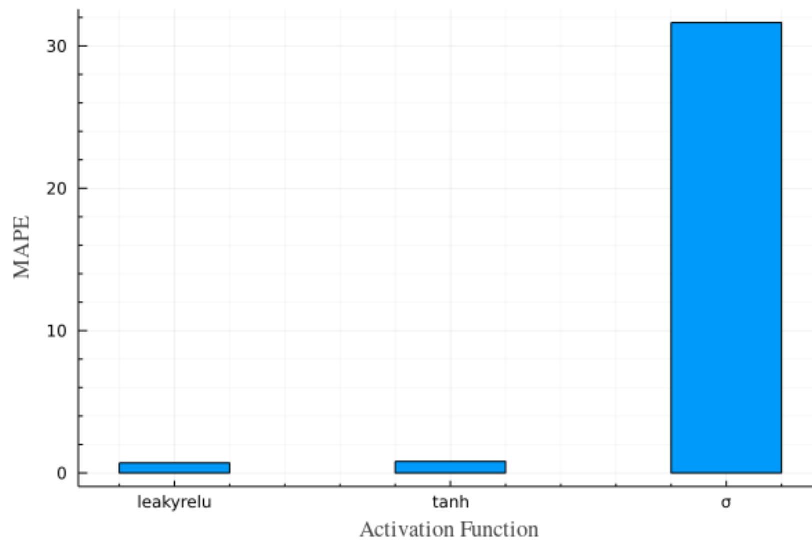

Figure A.20: Evaluation of different activation functions for the best STLB model of Set A.

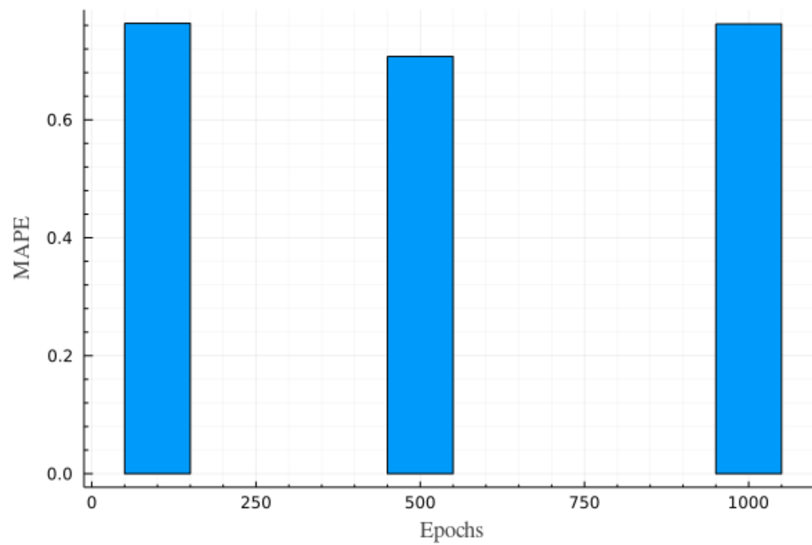

Figure A.21: Comparison of different training epochs for the best STLB model of Set A

Appendix B

Model Set B Diagrams

Following Appendix A, this Appendix presents the rest of the diagrams of Section 4.2, that belong to the best ST model of group B.

B.1 Speed-Torque (ST) Models

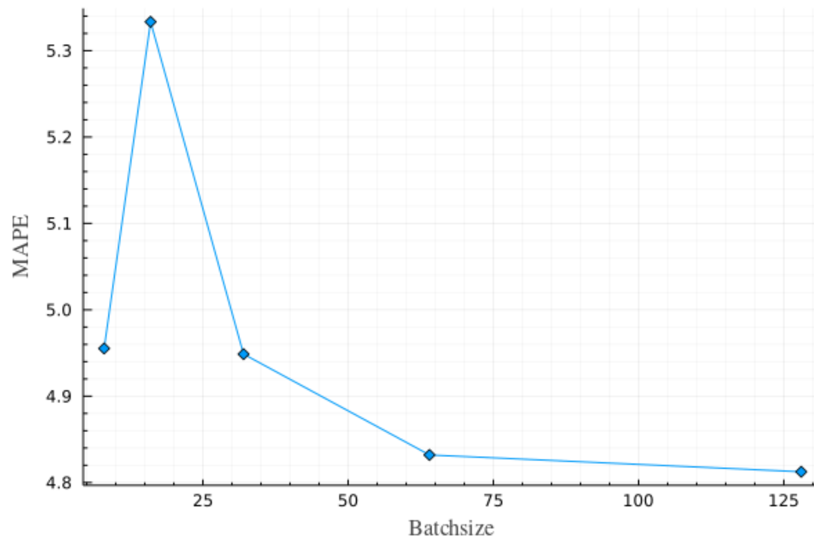

Figure B.1: The MAPE of the best ST model of Set B with the ADAM optimizer, for various batchsizes.

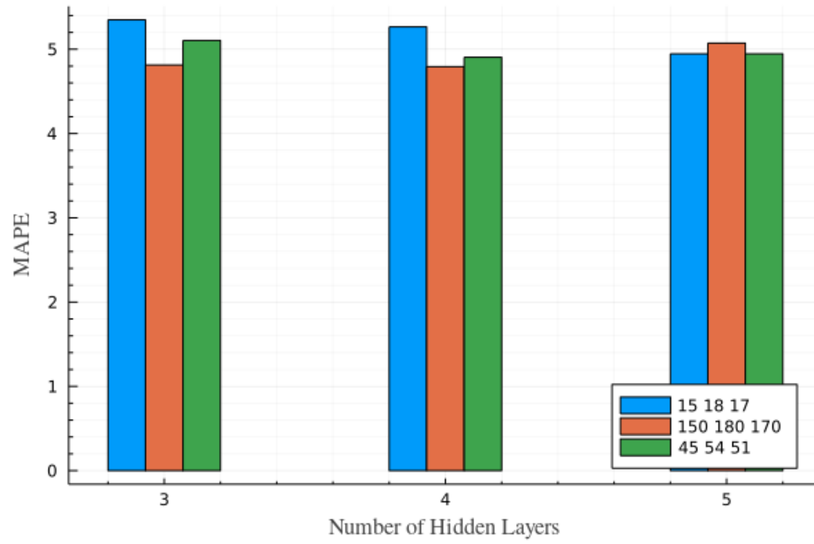

Figure B.2: Comparison of variations to the number of hidden layers and neurons for the best ST model of Set B.

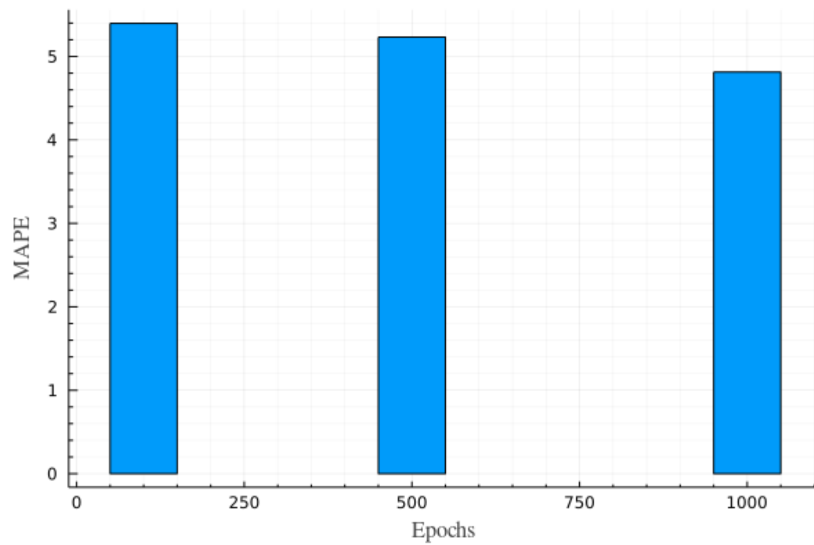

Figure B.3: Comparison of different training epochs for the best ST model of Set B