ΕΘΝΙΚΟ & ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
Τμήμα Φυσικής

National and Kapodistrian University of Athens
Department of Physics

Bachelor Thesis

A Physics-Informed Neural Network Approach to Predicting Dynamics and Identifying Unknown Parameters in Diesel Engines

Dimitrios Kalkantzis
AM 202000051

Supervisor:
Aris Moustakas

Athens, July 2025

Περίληψη

Η παρούσα πτυχιακή εργασία πραγματεύεται τη δυναμική ανάλυση ενός μέσου μοντέλου κινητήρα ντίζελ μέσω Φυσικά Ενσωματωμένων Νευρωνικών Δικτύων (Physics - Informed Neural Networks). Συγκεκριμένα, το πρόβλημα που εξετάζεται αφορά τον προσδιορισμό της δυναμικής ορισμένων μεταβλητών κατάστασης (state variables) καθώς και ορισμένων αγνώστων παραμέτρων σημαντικής φυσικής σημασίας για ένα μέσο μοντέλο του κινητήρα. Οι μεταβλητές κατάστασης ,των οποίων οι δυναμικές επιδιώκεται να προσδιοριστούν, είναι σε ελεύθερη μετάφραση οι εξής: η πίεση πολλαπλής εισαγωγής, η πίεση πολλαπλής εξαγωγής, η ταχύτητα τουρμπίνας , οι θέσεις των θέσων βαλβίδων ανακυκλοφορίας των καυσαερίων (EGR), η θέση ενεργοποιητή μεταβλητής γεωμετρίας του υπερσιμπιεστη (VGT) , η θερμοκρασία κατά το κλείσιμο της βαλβίδας μετά την φάση εισαγωγής και το κλάσμα υπολειπόμενων καυσαερίων. Οι φυσικές παράμετροι που προσπαθούμε να προσδιορίσουμε είναι : η μέγιστη ενεργή επιφάνεια της βαλβίδας EGR, ο συντελεστής αντιστάθμισης μη ιδανικών κύκλων του κινητήρα, ο συντελεστής συνολικής μετάδοσης θερμότητας των σωλήνων εξαγωγής και η μέγιστη επιφάνεια ροής αερίων στην τουρμπίνα.

Η φυσική του συστήματος περιγράφεται από ένα σύστημα 6 συνήθων διαφορικών εξισώσεων , η επίλυση των οποίων μας δίνει τη δυναμική των πρώτων έξι προαναφερθέντων μεταβλητών κατάστασης, ενώ οι άλλες δύο μεταβλητές δύνανται να προσδιοριστούν μέσω της επίλυσης ενός συστήματως δύο αλγεβρικών εξισώσεων. Η αδυναμία επίλυσης του προβλήματος αναλυτικά είναι αυτή που καθιστά την ανάγκη αξιοποίσης υπολογιστικών μεθόδων με σκοπό την καλύτερη δυνατή προσέγγιση των λύσεων. Εκ πρώτης όψεως , μία προφανής προσέγγιση είναι η επίλυση των δεδομένων εξισώσεων μέσω μεθόδων αριθμητικής ολοκλήρωσης. Η μεγάλη διαφορά στις τάξεις μεγέθους των μεταβλητών κατάστασης και στη μεταβλητότητά τους ως προς το χρόνο δημιουργούν την ανάγκη χρήσης πολύ μικρών βημάτων χρόνου . Επιπλέον, η πολυπλοκότητα του συτήματος έχει ως αποτέλεσμα την δημιουργία σωρευτικών σφαλμάτων (τα σφάλματα μεταφέρονται σε μεγάλο αριθμό διαδοχικών πράξεων) τα οποία οδηγούν σε μεγάλες αποκλίσεις. Έτσι, για την προσέγγιση των λύσεων, κατασκευάσαμε ένα μοντέλο αποτελούμενο από Φυσικά Ενσωματωμένα Νευρωνικά δίκτυα. Η διαφορά ενός τέτοιου μοντέλου με ένα κλασικό μοντέλο μηχανικής μάθησης , το οποίο βασίζεται αποκλειστικά σε δεδομένα, είναι ότι έχει τη δυνανότητα να ενσωματώνει τους φυσικούς νόμους που διέπουν ένα σύστημα και να τους αξιοποιεί ως πρόσθετη πληροφορία προκειμένου να προβλέψει το επιθυμητό αποτέλεσμα.

Αρχικά, προκειμένου να κατανοήσουμε την λειτουργηκότητα ενός τέτοιου μοντέλου είναι σημαντικό να εξετάσουμε τόσο τα βασικά στοιχεία της μηχανικής μάθησης, όσο και τον τρόπο με τον οποίο τα φυσικά ενσωματωματωμένα νευρωνικά δίκτυα (PINNs) εντάσσουν τη φυσική του συστήματος κατά την εκπαιδευτική διαδικασία. Έτσι, στο πρώτο κεφάλαιο , παρουσιάζεται η βασική θεωρία της μηχανικής μάθησης και ειδικότερα η μαθηματική θεμελίωση των νευρωνικών δικτύων. Βασικά στοιχεία της εκπαίδευσης αποτελούν η συνάρτηση κόστους καθώς και οι αλγόριθμοι βελτιστοποίησης, ενώ δίνεται ιδιαίτερη έμφαση στα προβλήματα παλινδρόμησης (Regression Problems), τα οποία συναντούμε επανειλημμένα κατά την επίλυση του βασικού μας προβήματος, και πιο συγκεκριμένα στα μη γραμμικά πολυωνυμικά μοντέλα. Ιδιαίτερα σημαντική καθίσταται η επεξήγηση των φαινομένων underfitting και overfitting αλλά και στους τρόπους αντιμετώπισής τους, έτσι ώστε να επιτευχθεί βέλτιση προσαρμογή στα δεδομένα.

Στη συνέχεια, στο δεύτερο κεφάλαιο, παρουσιάζεται η βασική θεωρία των PINNs καθώς και μία παραλλαγή τους τα Αντίστροφα ΠΙΝΝς (Inverse PINNs ή I-PINNs). Κατά την ανάλυση της αρχιτεκτονικής και της δομής των συγκεκριμένων τύπων νευρωνικών τονίζεται η μορφή της συνάρτησης κόστους (loss function) η οποία περιλαμβάνει όρους που σχετίζονται τόσο με την ακρίβεια ως προς τα δεδομένα (Data Loss) όσο και με την ικανοποίηση των διαφορικών εξισώσεων που περιγράφουν το σύστημα (Equation Loss) και των αρχικών συνθηκών (Initial Condition Loss). Στην περίπτωση των I-PINNs, τα οποία έχουν παρόμοια δομή με τα κλασικά PINNs δίνεται έμφαση στην αντιστροφή του προβλήματος, όπου οι άγνωστες παράμετροι ή ακόμη και

οι συντελεστές των εξισώσεων αντιμετωπίζονται ως μεταβλητές εκμάθησης. Έτσι, το δίκτυο δεν προβλέπει μόνο την εξέλιξη των μεταβλητών κατάστασης, αλλά και ταυτόχρονα εκτιμά παραμέτρους του ίδιου του φυσικού συστήματος, επιτρέποντας την εφαρμογή της μεθόδου ακόμη και σε περιπτώσεις περιορισμένης γνώσης των δυναμικών του μοντέλου.

Στο τρίτο κεφάλαιο, εξετάζεται η φυσική του υπό μελέτης συστήματος, δηλαδή του μοντέλου μηχανής ντίζελ, με στόχο την κατανόηση της φυσικής σημασίας των μεταβλητών κατάστασης και των φυσικών παραμέτρων των οποίων τις τιμές επιδιώκουμε να προσδιορίσουμε. Παράλληλα, επισημαίνεται η σημασία προεκπαιδευμένων νευρωνικών δικτύων Pre-trained Neural Networks τα οποία έχουν ως ρόλο να αντικαθαστίσουν πολύπλοκες αναλυτικές συναρτήσεις οι οποίες συμμετέχουν στη μαθηματική ανάλυση του συτήματος. Για την προσεγγιστική αναπαράσταση αυτών των συναρτήσεων, αξιοποιούνται εμπειρικές σχέσεις (empirical formulae) οι οποίες μπορούν να υπολογιστούν με λήψη δεδομένων για τις δυναμικές ορισμένων περεταίρω μεταβλητών που περιγράφουν χρονικά μεταβαλλόμενα φυσικά μεγέθη του κινητήρα. Στην πραγματικότητα, η κατασκευή τέτοιων νευρωνικών ισοδυναμεί με την επίλυση ενός προβλήματος παλινδρόμησης και έχει ως αποτέλεσμα την ελάττωση της πολυπλοκότητας των υπολογισμών σε πλαίσιο υλοποίσης και κατέπτέκταση την εξοικονόμηση υπολογιστικού χρόνου.

Η βασική πρόκληση μας για την πιο αποτελεσματική επίδοση του μοντέλου είναι η ελάττωση της πολυπλοκότητας κατά την εκπαιδευτική διαδικασία . Για τον σκοπό αυτό, αρχικά απομονώνουμε τις διαφορικές εξισώσεις των μεταβλητών οι οποίες δεν είναι αλληλοεξαρτώμενες και προσπαθούμε να υπολογίσουμε την δυναμική των εξισώσεων αυτών ξεχωριστά . Στο επόμενο στάδιο , δημιουργούμε μια αρχιτεκτονική με τα εναπομείναντα PINNs τα οποία εκπαιδεύονται ταυτόχρονα και προσπαθούν να προσεγγίσουν τις δυναμικές των αλληλοεξαρτώμενων μεταβλητών του συστήματος καθώς και τις πραγματικές τιμές των αγνώστων φυσικών παραμέτρων του κινητήρα. Ιδιαίτερη έμφαση δίνεται στη μορφή της ολικής συνάρτησης κόστους του μοντέλου η οποία αποτελείται από μεγάλο αριθμό επιμέρους συναρτήσεων κόστους. Συγκεκριμένα, καθε PINN θα εκπαιδεύεται με τουλάχιστον δύο επιμέρους συναρτήσεις κόστους οι οποίες θα προσπαθούν να ικανοποιήσουν τόσο την αρχική συνθήκη όσο και τη φυσική της εκάστοτε μεταβλητής, ενώ παράλληλα για PINNs τα οποία εκπαιδεύονται για να προσεγγίσουν μεταβλητές κατάστασης για τις οποίες έχουμε στη διάθεσή μας παρατηρήσιμα δεδομένα προσθέτουμε μια επιμέρους συνάρτηση κόστους δεδομένων . Η στρατηγική που εφαρμόζουμε προκειμένου η ολική συνάρτηση κόστους να ελαχιστοποιεί όλους τους επιμέρους όρους κόστους με τον ίδιο ρυθμό. Εφόσον οι μεταβλητές κατάστασης παρουσιάζουν μεγέλες διαφορές στις τάξεις μεγέθους μεταξύ τους, πολλαπλασιάζουμε τους επιμέρους όρους συνάρτησης κόστους με αντίστοιχα βάρη έτσι ώστε ο καθένας να συνεισφέρει κατά το δυνατόν ισάξια σε κάθε βήμα στη συνολικό κόστος.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα αποτελέσματα της μελέτης. Το μοντέλο εκπαιδεύτηκε υπό διαφορετικές συνθήκες θορύβου και δειγματοληψίας, προκειμένου να αξιολογηθεί η ευρωστία του. Συγκεκριμένα, εξετάστηκαν περιπτώσεις με προσθήκη λευκού θορύβου (white noise) στα δεδομένα, με επίπεδα 1%, 2%, 3% και 4%, καθώς και περιπτώσεις με αραιή δειγματοληψία (data sparsity), χρησιμοποιώντας χρονικά βήματα $\Delta t = 0.2$, 0.5, 1 και 2 δευτερόλεπτα. Τα αποτελέσματα αποτυπώνουν την ικανότητα του PINN μοντέλου να παραμένει ακριβές και σταθερό, ακόμα και υπό συνθήκες αυξανόμενης αβεβαιότητας και περιορισμένης πληροφορίας.

Η αξιολόγηση της απόδοσης του μοντέλου πραγματοποιήθηκε με βάση το μέσο τετραγωνικό σφάλμα (Mean Squared Error, MSE) μεταξύ των τιμών που παράγονται από το Simulink και των προβλέψεων του PINN. Παρατηρήθηκε ότι το MSE διατηρείται σε χαμηλά επίπεδα ακόμα και σε υψηλά επίπεδα θορύβου και μεγάλα χρονικά διαστήματα δειγματοληψίας. Επιπλέον, παρουσιάζονται γραφήματα σύγκρισης μεταξύ των πραγματικών και των προβλεπόμενων μεταβλητών κατάστασης, προκειμένου να αξιολογηθεί και ποιοτικά η ακρίβεια του μοντέλου.

Abstract

This Thesis proposes a PINN framework to simulate state dynamics and infer unknown parameters of a diesel engine model of six ordinary differential equations and two algebraic equations. There are eight state variables and four unknown parameters in the model. Normally, there are three components for the loss function: initial condition loss, equation residual loss, and data loss. One of the challenges is the very huge differences between the magnitudes of the parameters, which might lead to unbalanced optimization. Thus, to ensure that all three loss components contribute about equally to the initial training iteration, each was normalized by its order of magnitude.

It employs pretrained neural networks as surrogate models to substitute some analytically intractable functions inside equation loss terms, thus managing complex nonlinearities in governing equations. Training was performed sequentially in three structured stages to reduce complexity: first, trained PINNs to predict three uncoupled state variables; these networks were then reused in training five remaining coupled variables. Three of the five variables were accurately predicted in this second phase, while the remaining two were refined in a third phase by freezing the weights of the well-trained components and focusing optimization on the remaining ones.

The proposed method demonstrated strong performance in both clean and noisy data, with robustness maintained up to 4% noise. It also outperformed a benchmark PINN approach that did not incorporate loss normalization. Furthermore, two of the four unknown parameters were successfully inferred.

The thesis also includes a brief theoretical overview of machine learning and a mathematical analysis of deep learning fundamentals, including derivative computations for activation and loss functions, the backpropagation algorithm, and a structured methodology for regression problems.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my professor, Mr. Aris Moustakas, for his invaluable guidance, encouragement, and for the opportunity he gave me to work on such an interesting and challenging project. His mentorship not only helped me develop essential skills in machine learning, but also deepened my understanding of how these concepts are applied in real-world scenarios. Our collaboration gave me a strong sense of direction and motivation to continue exploring this field further in both academic and practical contexts. Also, I am sincerely thankful to my family for their constant support throughout this journey. Lastly, a very special thank you to my partner, Jessica, whose understanding and unwavering belief in me have been an endless source of strength and inspiration.

Contents

List of Figures

List of Tables

Introduction

Over the past few decades, the increasing availability of data, and thus the computing power of processing it, has led to a tendency to let systems learn by themselves instead of being programmed explicitly for a task. This procedure has been revolutionized with machine learning as numerous fields have benefited from intelligent decision making without explicit human programming such as medical sciences, environmental modeling, automotive engineering, natural sciences, etc.

At its core, Machine Learning relies on algorithms that learn from input data to perform various tasks with a wide range of techniques. These techniques are further enhanced by Deep Learning, which is a subset of ML that leverages multi-layered neural networks (NNs) to process vast amounts of unstructured data. Deep learning methods are particularly effective at capturing complex patterns, which has led to breakthroughs in image recognition, natural language processing, and, as we will conclude in the present study, physical system modeling.

Although data-driven models often fit observations well, their predictions can sometimes be unrealistic or inconsistent. This challenge highlights the need to integrate core physical laws and domain knowledge into ML models, essentially 'teaching' them about the fundamental principles governing real-world phenomena. Consequently, physics-informed machine learning emerges as a critical approach, allowing models to take advantage of empirical, physical, or mathematical insights to enhance their performance and reliability.

In the first chapter, we present a comprehensive overview of the fundamental concepts of machine learning and deep learning, with a particular emphasis on regression problems, which are central to the scope of this work. The second chapter introduces the theory of Physics-Informed Neural Networks (PINNs) and Inverse PINNs, establishing a connection between these frameworks and the core machine learning principles discussed in the preceding chapter. In the third chapter, we formulate the problem under investigation and provide a detailed methodology for constructing a PINN-based model capable of predicting the state dynamics and identifying unknown parameters of a mean diesel engine model. Additionally, we describe the development of pre-trained neural networks that serve as surrogates for analytical functions embedded within the physical laws governing the system. The model's robustness and performance are evaluated under conditions of noisy and sparse data. In the fourth chapter, we present the results, including data generation, the prediction accuracy of the state variables and unknown parameters, and the evolution of the PINNs' loss functions. Finally, we conclude with a discussion of the key findings and propose directions for future research.

Chapter 1

Theoretical Background

1.1 Brief theory of Machine Learning

Machine learning (ML), a branch of artificial intelligence, focuses on creating and analyzing algorithms that allow systems to learn from data and make predictions on new, unseen inputs , without being explicitly programmed for specific tasks. All machine learning algorithms consist of three key elements:

- **Representation:** A model must be represented in a way that is manageable for a computer. This defines the 'concepts' that the model can learn, forming the hypothesis space. Examples include a decision tree, a neural network, or a set of labeled data points.
- **Evaluation:** This involves an integral method to select one hypothesis over another based on an objective function, scoring function or loss function. For example, it might measure the difference between the correct output value and the model's predictions.
- **Optimization:** An efficient method for searching through the hypothesis space. The process typically begins with a simple hypothesis and adjusts it as needed if it doesn't fit the data. We start with an initial set of model parameters and gradually improve them.

Machine learning includes a broad range of algorithms, each designed for different types of tasks and data. Common methods include linear regression, decision trees, support vector machines (SVMs), k-nearest neighbors (KNN), and neural networks. These algorithms differ in terms of complexity, assumptions, and how they are trained. For example, neural networks require iterative optimization and are highly sensitive to hyperparameters such as the learning rate. Crucially, even a highly expressive model—such as a deep neural network—can fail to perform well if it is not optimized effectively. This underscores the fact that model capacity alone is not enough; successful learning depends just as much on the quality of the training process.

In the field of machine learning, problems are typically divided into classification and regression tasks.. Classification involves assigning input data to discrete categories or classes, such as identifying whether an email is spam or not. In contrast, regression focuses on predicting continuous values, such as temperature, velocity, or pressure over time. The work presented in this thesis falls under the regression category, as it aims to model the continuous dynamic behavior of a diesel engine using physics-informed neural networks (PINNs).

In many machine learning settings, the objective is to construct a rule that transforms observed data into a meaningful output or decision. Formally, we define a decision function f that maps an input $x \in \mathcal{X}$—representing the available observations—to an output $\hat{s} \in \mathcal{S}$, such that:

$$f : \mathcal{X} \to \mathcal{S} \tag{1.1}$$

This general formulation covers a wide range of problem types, including estimation, classification, regression, and control. The nature of the specific task determines both the structure of the input space $\mathcal{X}$ and

the output space $\mathcal{S}$. For instance, in classification problems, $\mathcal{S}$ consists of discrete labels, while in regression problems, it typically represents a continuous domain.

While the function f can, in theory, be any mapping from the input space $\mathcal{X}$ to the output space $\mathcal{S}$, practical implementations typically constrain f to a parameterized family of functions $\mathcal{F}$. These structured function classes enable efficient training, generalization, and optimization. In modern machine learning, particularly deep learning, $\mathcal{F}$ often consists of neural networks with predefined architectures and trainable parameters.

To train a model, we rely on a labeled dataset $\mathcal{D}$ that is typically partitioned into three disjoint subsets:

- **Training set:** $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N_{\text{train}}}$ is used to optimize the model parameters by minimizing a loss function.
- **Validation set:** $\mathcal{D}_{\text{val}} = \{(\mathbf{x}_j, y_j)\}_{j=1}^{N_{\text{val}}}$ is used to tune hyperparameters and monitor performance during training to prevent overfitting.
- **Test set:** $\mathcal{D}_{\text{test}} = \{(\mathbf{x}_k, y_k)\}_{k=1}^{N_{\text{test}}}$ is reserved for evaluating the final model's generalization performance on unseen data.

The decision function f is learned by minimizing a **cost function** $\mathcal{J}(f)$, which typically measures the **empirical risk**, i.e., the average loss on the training set:

$$\mathcal{J}(f) = \frac{1}{N_{\text{train}}} \sum_{i=1}^{N_{\text{train}}} \mathcal{L}(f(\mathbf{x}_i), y_i) \tag{1.2}$$

Here, $\mathcal{L}(\cdot, \cdot)$ is a task-specific loss function, such as mean squared error for regression or cross-entropy for classification. The goal of training is to find the function $f \in \mathcal{F}$ that minimizes this cost.

The expressiveness of $\mathcal{F}$ directly affects the model's capacity to fit data. A function class that is too simple may result in **underfitting**, where the model cannot capture important patterns in the data. On the other hand, a highly flexible $\mathcal{F}$ may lead to **overfitting**, where the model memorizes noise in the training set rather than learning generalizable patterns. As illustrated in Figure 1.2, achieving good generalization requires balancing model complexity with the available data. This balance is commonly maintained using regularization techniques, validation-based model selection, and careful architecture design.

During training, we monitor the behavior of the training and validation loss curves to assess the model's learning dynamics and detect potential overfitting or underfitting.These losses are typically plotted against the number of **epochs**, where each epoch corresponds to one complete pass through the entire training dataset. Ideally, both losses should decrease as the training progresses. However, their relative values and trends can provide important diagnostic information:

- If both training and validation losses remain high, the model may be **underfitting**, indicating insufficient capacity, poor feature representations, or inadequate training.
- If the training loss is low but the validation loss is significantly higher and begins to increase while training loss continues to decrease, this is a sign of **overfitting**. The model is learning noise or spurious correlations from the training data rather than general patterns.
- A good fit is typically characterized by both training and validation losses decreasing and stabilizing at low values, with a small gap between them.

It is generally expected that the validation loss will be slightly higher than the training loss, since the model is directly optimized to minimize the latter. However, if the gap is too large, it indicates overfitting. Conversely, if the gap is small but both losses are high, it suggests underfitting. All the above are summarized and illustrated in Figure 1.1.

Also, it is worth noting that in some cases, the validation loss may temporarily fall below the training loss, especially in the early stages of training or due to certain regularization techniques such as dropout. This behavior is not necessarily problematic. It can occur when regularization (applied during training) makes the model perform slightly worse on the training set than it would otherwise, while still improving generalization

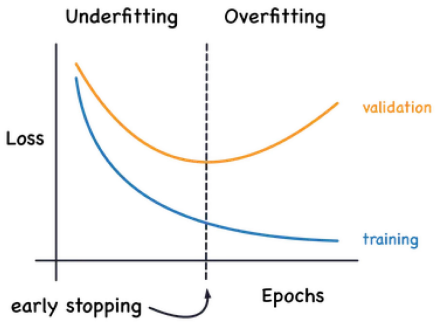


Figure 1.1: Illustration of underfitting and overfitting behavior during training. The training loss (blue curve) decreases consistently with more epochs, while the validation loss (orange curve) initially decreases but then starts increasing as the model overfits. The optimal stopping point is marked by early stopping, where the validation loss is minimized and generalization is maximized. Figure adapted from [12]

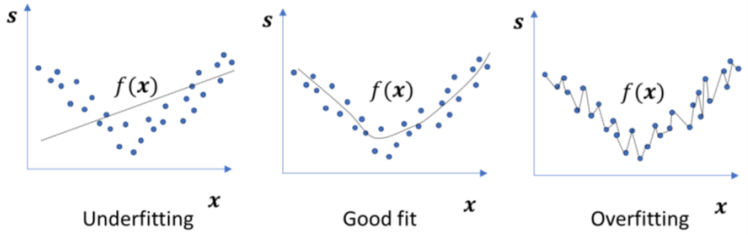


Figure 1.2: Illustration of model complexity and generalization. **Left:** Underfitting occurs when the model is too simple to capture the underlying structure in the data. **Center:** A well-fitted model accurately captures the true pattern while maintaining generalization. **Right:** Overfitting happens when the model is overly complex and captures noise in the training data, reducing its predictive performance on unseen data. Figure adapted from [12]

on unseen data. Moreover, randomness in mini-batch selection, data augmentation, or differences in how metrics are computed across datasets can lead to fluctuations where validation loss is lower than training loss. As long as the final model achieves good performance on the test dataset, such anomalies do not indicate overfitting and are generally acceptable.

To mitigate overfitting, one may employ techniques such as early stopping (halting training when validation loss stops improving), regularization (e.g., L2 penalty), dropout, or expanding the training dataset. These strategies help the model generalize better and maintain a healthy trade-off between training accuracy and validation performance.

1.2 Mathematical Foundations of Deep Learning

Deep learning is a subset of machine learning that leverages neural networks to carry out tasks like classification, regression etc. Inspired by the structure of the human brain, deep learning involves arranging artificial neurons into multiple layers and "training" them to interpret and process data. The term "deep" highlights the fact that these networks consist of layers, ranging from three to several hundred or thousands.

1.2.1 Feed-Forward Neural Networks (FFNNs)

1.2.1.1 A single neuron

In the field of deep learning , a Feed - Forward Neural Network (FFNN) is a multi - layer perceptron which is used to distinguish data that is not linearly separable. In order to understand further the definition

and the functionality of a FFNN we will explore mathematically the concept of a single **neuron** in a neural network. A single neuron, often referred as node, is connected with a sample of inputs $(x_1, x_2, \ldots, x_n) \in \mathbb{R}$, while each one is multiplied by its corresponding weight $(w_1, w_2, \ldots, w_n) \in \mathbb{R}$. The neuron takes the weighted sum of these inputs added with a bias b. This process is described mathematically with a function $\sigma : \mathbb{R} \to \mathbb{R}$ which is called **activation function**. . Taking the above into consideration, output a of a neuron is described by the formula :

$$a = \sigma\left(\sum_{i=1}^{n} x_i w_i + b\right) \tag{1.3}$$

The forementioned process can also be described in a more understandable way by using dot products of an input vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and weights' vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$. These vectors can be expressed as matrices. By transposing one of the two we can then multiply them and rewrite the expression of the neuron's output as:

$$a = \sigma\left(\mathbf{x}^T\mathbf{w} + b\right) \tag{1.4}$$

If we let z be the input of the activation function σ often referred as **pre-activation value** then

$$z = \mathbf{x}^T\mathbf{w} + b \tag{1.5}$$

$$a = \sigma(z) \tag{1.6}$$

An example of a single neuron's functionality is illustrated in Figure 1.3

We have already given mathematical insight about a neurons activation a but no information is yet provided about the activation $\sigma(\cdot)$. In fact, there are multiple forms this function could take. The choice of an activation function plays a crucial role in the model's ability to learn complex, nonlinear relationships. Some remarkable examples of activation functions that are often used in deep learning architectures are presented in Table 1.1

Function	Formula	Output Range	Purpose
ReLU	$f(z) = \max(0, z)$	$[0, +\infty)$	Simple and efficient. Encourages sparsity.
Sigmoid	$f(z) = \frac{1}{1+e^{-z}}$	$(0, 1)$	Used in binary classification. Prone to vanishing gradients.
Tanh	$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$(-1, 1)$	Zero-centered. Preferred over sigmoid in hidden layers.
Softmax	$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$	$(0, 1)$	Converts outputs to probability distribution. Used in multi-class classification.

Table 1.1: Common activation functions used in neural networks, along with their mathematical formulas, output ranges, and typical applications.

1.2.1.2 A layer of neurons

A fully connected (dense) layer in a neural network consists of a set of neurons, each receiving inputs from all neurons in the previous layer. The goal is to compute the activations of each neuron in the current layer based on the outputs from the previous layer.

Let $a_k^{[l-1]}$ denote the activation of the k-th neuron in layer $l-1$, and $a_j^{[l]}$ denote the activation of the j-th neuron in layer l. The computation of the pre-activation value $z_j^{[l]}$ for neuron j in layer l is given by:

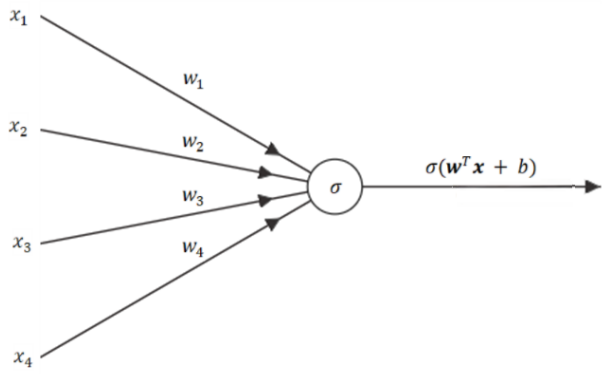


Figure 1.3: Example of a single neuron with 4 inputs

$$z_j^l = \sum_{k=1}^{n_{l-1}} w_{jk}^l a_k^{l-1} + b_j^l \tag{1.7}$$

where:

- w_{jk}^l is the weight connecting the k-th neuron of layer $l-1$ to the j-th neuron of layer l,
- b_j^l is the bias term of neuron j in layer l,
- n_{l-1} is the number of neurons in layer $l-1$.

Once the pre-activation value $z_j^{[l]}$ is computed, it is passed through an activation function $\sigma(\cdot)$ to produce the final activation:

$$a_j^l = \sigma(z_j^l) \tag{1.8}$$

For the entire layer, this computation can be compactly expressed using matrix notation. Let:

- $\mathbf{a}^{l-1} \in \mathbb{R}^{n_{l-1}}$: activation vector from layer $l-1$,
- $\mathbf{W}^l \in \mathbb{R}^{n_l \times n_{l-1}}$: weight matrix,
- $\mathbf{b}^l \in \mathbb{R}^{n_l}$: bias vector,
- $\mathbf{z}^l \in \mathbb{R}^{n_l}$: pre-activation vector,
- $\mathbf{a}^l \in \mathbb{R}^{n_l}$: activation vector.

Then the layer computation becomes:

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l \tag{1.9}$$

$$\mathbf{a}^l = \sigma(\mathbf{z}^l) \tag{1.10}$$

where σ is applied element-wise.

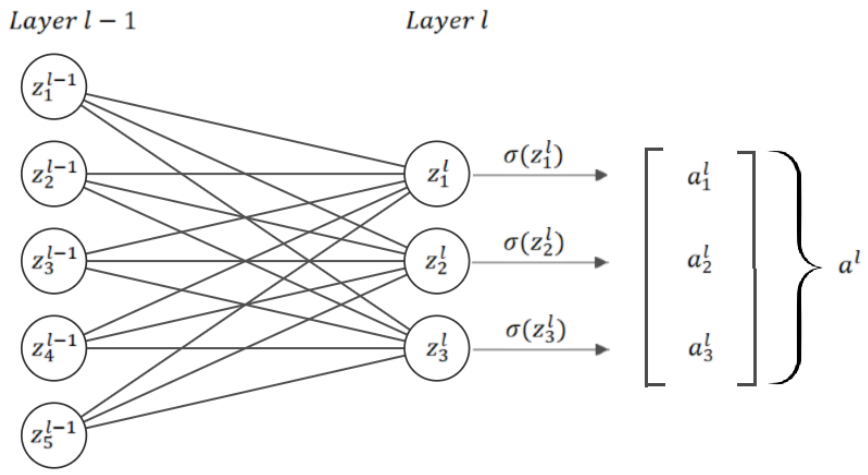


Figure 1.4: Feedforward computation from layer $l-1$ to layer l in a neural network. Each neuron in layer l computes a pre-activation value z_j^l as a weighted sum of the activations from the previous layer a_i^{l-1}, followed by the application of an activation function σ to produce the output activation a_j^l. The resulting activations form the vector $\mathbf{a}^l$, which serves as input to the next layer.

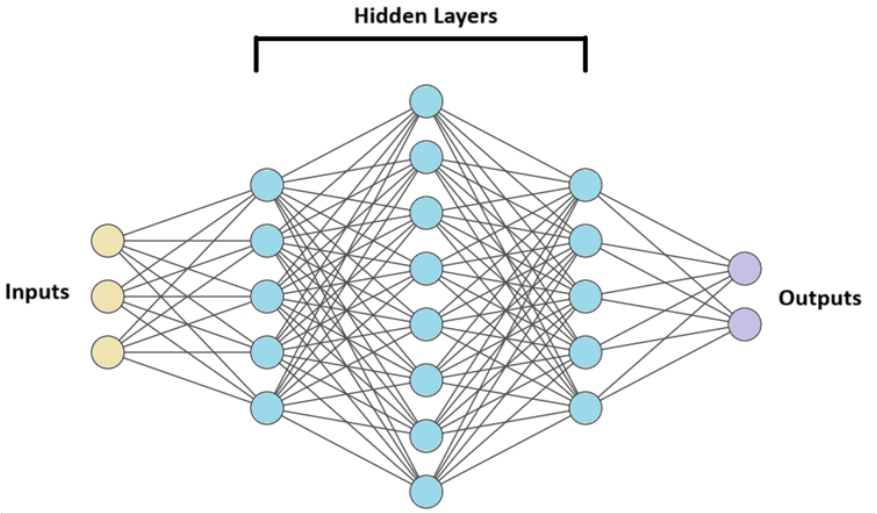


Figure 1.5: Structure of a multi-layer perceptron (MLP) consisting of 3 input nodes (yellow), 3 hidden layers composed of neurons (blue), and 2 output nodes (purple). While the input and output nodes are depicted similarly to neurons for visualization purposes, they do not perform neuron-like computations. They can be either scalars or vectors depending on the specific application.

1.2.1.3 Multi-layer perceptron

The above formulation allows efficient computation of all neuron outputs in parallel and forms the basis of feedforward computations in deep neural networks. A ***feedforward neural network*** (FFNN) is constructed by stacking multiple such layers, where the output (activation) of one layer becomes the input to the next. This architecture is commonly referred to as a ***multi-layer perceptron***, which is a class of FFNNs consisting of an input layer, one or more hidden layers, and an output layer. An example of a multi-layer perceptron is depicted in Figure 1.5.

Each layer in an MLP performs a linear transformation of its input followed by a non-linear activation function. These successive transformations allow the network to learn hierarchical representations of the input data and approximate complex functions. The input propagates strictly forward from layer to layer, which is a defining characteristic of FFNNs.

The choice of architecture—such as the number of layers, the number of neurons per layer, and the activation functions—plays a crucial role in the network's capacity to model the underlying structure of the data. Ultimately, the final layer produces the output corresponding to the network's prediction.

1.2.2 The Cost Function

Generally, a **cost function** $\mathcal{C}$ (also known as a **loss function**) quantifies the difference between the predicted output $\hat{y}$ and the true target y. In our case, the output of a cost function is a scalar value which measures the error between a feed - forward neural network's predictions and actual targets of training data. Its purpose is to measure how well the model is performing and to guide the learning process by updating the network's weights and biases during training. More specifically, the network is trying to minimize the value of the cost function using optimization algorithms. There are multiple types of cost functions the choice of which depends on the type of the problem we are trying to solve.

In this section, we will focus on calculating the derivatives of a cost function with respect to each of the weights and biases and observing the change these cause to the cost function. Also, we will talk about how we can use these derivatives to create an algorithm to automatically find the lower cost.

First of all, we want to calculate the derivatives of neurons activation with respect to weight and bias. If we let H be a matrix whose elements arise from the Hadamard product (or element-wise multiplication) computed by multiplying the corresponding elements of matrices $\mathbf{w}$ and $\mathbf{x}$, , S is a function whose input is H and gives the sum of each element H_{ij} we have the following expression considering equations 1.3 - 1.6:

$$a = \sigma(\mathbf{w}^T\mathbf{x} + b) = a(z) = \sigma\left(S(H) + b\right) \tag{1.11}$$

$$S(H) = sum(H) = \sum_{i,j} H_{ij} = \sum_{i,j} w_{ij}x_{ij} \tag{1.12}$$

$$H = \mathbf{w} \circ \mathbf{x} \tag{1.13}$$

Now, we will focus on calculating the partial derivatives of activation a with respect to weight and bias. According with the chain rule the derivatives are calculated as it follows:

$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial S}\frac{\partial S}{\partial H}\frac{\partial H}{\partial w} \tag{1.14}$$

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b} \tag{1.15}$$

We prove in Appendix A that $\frac{\partial H}{\partial w} = diag(x_1, x_2, \ldots, x_n)$ and $\frac{\partial S}{\partial H} = \mathbf{1}^T$, where $\mathbf{1}^T$ denotes a row vector of ones. Also, it is obvious that $\frac{\partial z}{\partial S} = \frac{\partial z}{\partial b} = 1$. With these assumptions :

$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial z}\mathbf{x}^T \tag{1.16}$$

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial z} \tag{1.17}$$

The complete mathematical expressions of the above derivatives depend on the choice of an activation function σ, while $a = \sigma(z)$. We have already discussed the functionality of different activation functions in Table 1.1.

Now, we will follow the same procedure for the derivatives of the cost function $C(\mathbf{s}, \hat{\mathbf{s}})$ with respect to the parameters $\mathbf{w}$ and b. If we assume that we feed our model with multiple vector inputs $X = \mathbf{x}^T = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}]^T$ for training ,with target vector $\mathbf{s} = [target(\mathbf{x_1}), target(\mathbf{x_2}), \ldots, target(\mathbf{x_N}) = [s_1, s_2, \ldots, s_N]$ where each s_i is scalar. For our analysis we also make the assumption that our cost function has is the mean squared error between the predicted outputs of our model and the target values.Mathematically, the cost function is then defined as :

$$C = \frac{1}{N}\sum_{i=1}^{N}(s_i - \hat{s_i})^2 \tag{1.18}$$

where $\hat{s_i}$ denotes the ith predicted output of our model. In the case of a feed-forward neural network, the model's output for the i-th input sample is denoted by $\hat{s}_i$, and it corresponds to the activation value a_i^L of the final layer L in the network. That is $\hat{s}_i = a_i^L$. If we also let $v = s_i - a_i^L$ then:

$$C = \frac{1}{N}\sum_{i=1}^{N} v^2 \tag{1.19}$$

Given that form for the cost function we can compute the derivatives $\frac{\partial C}{\partial \mathbf{w}}$, $\frac{\partial C}{\partial b}$ as follows:

$$\frac{\partial C}{\partial \mathbf{w}} = \frac{\partial}{\partial \mathbf{w}}\frac{1}{N}\sum_{i=1}^{N} v^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial \mathbf{w}}v^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial v^2}{\partial v}\frac{\partial v}{\partial \mathbf{w}} = \frac{1}{N}\sum_{i=1}^{N} 2v\frac{\partial v}{\partial \mathbf{w}} \tag{1.20}$$

$$\frac{\partial C}{\partial b} = \frac{\partial}{\partial b}\frac{1}{N}\sum_{i=1}^{N} v^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial}{\partial b}v^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial v^2}{\partial v}\frac{\partial v}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} 2v\frac{\partial v}{\partial b} \tag{1.21}$$

These gradients are used to iteratively update the model parameters during training using the gradient descent algorithm.

1.2.3 Backpropagation

Backpropagation is a fundamental algorithm used to train feed-forward neural networks by efficiently computing the gradients of the cost function with respect to the network's parameters. It applies the chain rule of calculus to propagate the error from the output layer backward through the network, layer by layer. These gradients are then used to update the weights and biases during training, typically via gradient descent. The main goal of backpropagation is to enable deep networks to learn complex functions from data by minimizing the cost function. There are four fundamental equations that describe how to compute gradients efficiently for all parameters in the network.

First of all, we introduce the error δ_j^l of neuron j in output layer L , which is computed as following:

$$\delta_j^l = \frac{\partial C}{\partial z_j^L} = \sum_k \frac{\partial C}{\partial a_k^L}\frac{\partial a_k^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{1.22}$$

It is important to mention for the computations above that the output activations a_k^L of the k^{th} neuron depends only on the weighted input z_j^L for the j^{th} neuron when $k = j$. As a result, gradients $\partial a_k^L/\partial z_j^L$ vanish when $k \neq j$.

For the corresponding matrix-based form we have:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{1.23}$$

In this equation, $\nabla_a C$ is defined to be a vector whose components are the partial derivatives $\partial C/\partial a_j^L$

Next, we are going to compute the error d_j^l of neuron j in layer l considering that we have knowledge of the error δ^{l+1} in the next layer. More specifically, this erro can be computed as following:

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{1.24}$$

For the computation of the derivative $\partial z_k^{l+1}/\partial z_j^l$ we have the following definition based on equation 1.7:

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{1.25}$$

By differentiating with respect to z_j^l we obtain:

$$\frac{\partial z_k^{l+1}}{\partial z_k^l} = w_{kj}^{l+1} \sigma'(z_j^l) \tag{1.26}$$

By substituting to equation 1.24 we have the following component from for δ_j^l:

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \tag{1.27}$$

The corresponding matrix-based expression for this form is the following:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{1.28}$$

We are also interested in calculating the rate of change of the cost C with respect to any bias b_j^l in the network. In other words the derivative $\partial C/\partial b_j^l$ is calculated according with the chain rule as following:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \tag{1.29}$$

It is obvious from equation 1.7 that $\partial z_j^l/\partial b_j^l = 1$. Also, from the definition of δ_j^l in eq 1.24 we conclude that:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{1.30}$$

Respectively, we would like to define the rate of change of the cost C with respect to any weight w_{jk}^l in the network which is expressed mathematically by the derivative $\partial C/\partial w_{jk}^l$. From equation 1.7 we have:

$$z_{jk}^l := w_{jk}^l a_k^{l-1} \tag{1.31}$$

$$\frac{\partial z_{jk}^l}{\partial w_{jk}^l} = a_k^{l-1} \partial w_{jk}^l \tag{1.32}$$

According to the chain rule the derivative $\partial C/\partial w_{jk}^l$ can be calculated as following:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_{jk}^l} \frac{\partial z_{jk}^l}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_{jk}^l} a_k^{l-1} \tag{1.33}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{1.34}$$

Equations 1.23,1.28,1.30,1.34 are considered to be the four fundamental equations for describing mathematically the backpropagation process for a feed-forward neural network.

Now, we will describe how the above equations contribute to the evaluations of the backpropagation algorithm. To begin with, the last begins by feeding the input $\mathbf{x}$ forward through the network to compute activations a_j^l for each layer $l = 1, 2, \ldots, L$, beginning with activation a^1 for the input layer . At the output layer L, the algorithm measures how far the network's output a^L is from the desired result s, and calculates the initial error δ^L. This error is then propagated backward through the network, layer by layer, using

the chain rule to determine how much each neuron in earlier layers contributed to the final error. These calculations can be achieved by utilizing equation 1.28. As we move backward, we compute gradients of the cost C with respect to each weight and bias by utilizing equations 1.34 and 1.30 respectively. These gradients are then used to update the parameters during training. The reason we go backward is because the cost depends on the output a^L, and to understand how earlier layers influence that cost, we need to trace their effects in reverse. This step-by-step backward computation of error δ^l in each layer l is what gives backpropagation its name.

1.2.4 Optimization Algorithms

Once the gradients of the cost function with respect to each weight and bias are computed using backpropagation, optimization algorithms are used to update these parameters in a way that reduces the overall cost. The most common method is gradient descent, where each weight and bias is adjusted slightly in the opposite direction of its gradient. This means that if a weight increases the cost, the algorithm will reduce it, and vice versa. Regardless of the method, the goal is to iteratively update weights and biases so the network learns to make more accurate predictions over time.

Gradient Descent The most basic optimization method is gradient descent, which updates the parameters in the direction opposite to the gradient. The general update rule is:

$$\theta \leftarrow \theta - \eta \nabla_\theta C,$$

where θ represents a weight or bias, η is the learning rate, and $\nabla_\theta C$ is the gradient of the cost function with respect to θ. Choosing the right learning rate is important: too large may lead to divergence, while too small results in slow training.

Momentum Momentum is used to accelerate the gradient descent process by incorporating an exponentially weighted moving average of past gradients. This helps smooth out the trajectory of the optimization allowing the algorithm to converge faster by reducing oscillations.

The update rule with momentum is:

$$w_{t+1} = w_t - \alpha m_t \tag{1.35}$$

where m_t is the moving average of the gradients at time t, α is the learning rate and w_t and w_{t+1} are the weights at time t and time $t+1$, respectively.

The momentum m_t is updated recursively as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t} \tag{1.36}$$

where β_1 is the momentum parameter and $\partial L / \partial w_t$ is the gradient of the loss function with respect to the weights at time t

RMSprop Root Mean Square Propagation(RMSprop) is an adaptive learning rate method that uses an exponentially weighted moving average of squared gradients, which helps overcome the problem of diminishing learning rates. The update rule for RMSprop is:

$$w_{t+1} = w_t - \frac{a_t}{\sqrt{v_t + \epsilon}} \frac{\partial L}{\partial w_t} \tag{1.37}$$

where v_t is the exponentially weighted average of squared gradients:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left(\frac{\partial L}{\partial w_t} \right)^2 \tag{1.38}$$

and ϵ is a small constant.

Adam Optimizer Adaptive Moment Estimation or Adam optimiser combines the momentum and RMSprop techniques to provide a more balanced and efficient optimization process. The key equations governing Adam are equations 1.36 and 1.38 which denote the first(moment) and second(moment) estimate respectively.

Since both m_t and v_t are initialized at zero, they tend to be biased toward zero, especially during the initial steps. To correct this bias, Adam computes the bias corrected estimates:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{1.39}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{1.40}$$

Adam is an optimization algorithm that addresses several limitations of traditional gradient descent methods. One of its key features is the use of dynamic, adaptive learning rates for each parameter, which are adjusted based on the history of past gradients and their magnitudes. This allows Adam to navigate the loss landscape more effectively, reducing oscillations and improving its ability to escape local minima. Overall, Adam is known for its efficiency and typically requires less tuning of hyperparameters compared to other methods, making it a popular and practical choice for many machine learning and deep learning tasks.

1.2.5 Regression Problems

Regression problems are a fundamental class of tasks in machine learning, where the goal is to predict continuous numerical values while given input data. Although there are numerous traditional machine learning techniques for solving regressions— which will not be described in this thesis— more complex mappings between input and target variables often require more advanced models. In such cases, deep learning extends regression capabilities by using neural networks with multiple layers to automatically learn intricate, non-linear relationships in large and high-dimensional datasets.

In the context of our work, we will be particularly concerned with solving regression problems. It then of outmost importance to provide the basic theory behind regression analysis and how with of deep learning and specifically the usage of feed-forward neural networks we can approach regression problems and finally be able to solve them.

A general form of a regression model can be expressed as:

$$Y_i = f(X_i, \beta) + e_i \tag{1.41}$$

where Y_i is the output for the i-th observation, X_i is the input vector, $\boldsymbol{\beta}$ denotes the model parameters, and e_i is the random error term. The main task in this problem is to estimate the function $f(X_i, \beta)$ that most closely fits the data. Relying on deep learning applications we are going to approximate the function $f(X_i, \beta)$ with a neural network $\mathcal{N}(X_i, \boldsymbol{\beta})$, where parameters $\boldsymbol{b}$ stand for the entire set of learnable parameters — all weights and biases in the network. In other words $\mathbf{b}$ is the mathematical set $\boldsymbol{\beta} = \{\mathbf{W}, \mathbf{b}\}$ where $\mathbf{W}$ and $\mathbf{b}$ stand for the weight matrix and the bias vector of the neural-network $\mathcal{N}(X_i, \boldsymbol{\beta})$ respectively. In order to make conclusions for the performance of the model used to solve a regression problem we evaluate our results using different metrics. Some frequently used metrics in regression problems are the following:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left(Y_i - \hat{Y}_i \right)^2 \quad (\text{ Mean Square Error})$$

$$MAE = \frac{1}{N} \sum_{i=1}^{N} \left| Y_i - \hat{Y}_i \right| \quad (\text{ Mean Absolute Error})$$

$$R^2 = 1 - \frac{\sum_{i=1}^{N} \left(Y_i - \hat{Y}_i \right)^2}{\sum_{i=1}^{N} \left(Y_i - \bar{Y} \right)^2} \quad (\text{Determination coefficient})$$

$$L_2 = \frac{\|Y - \hat{Y}\|_2}{\|Y\|_2} \quad (\text{L2 relative error})$$

where N is the total number of data points(observations) in the data set. A good fit of a neural network is characterized by low values of the Mean Squared Error (MSE) and Mean Absolute Error (MAE), while the coefficient of determination (R^2) should be as close to 1 as possible. However, it is also important to compare these metrics between the training and validation (or test) datasets. Large discrepancies between training and test errors may indicate overfitting, where the model learns the training data too well but fails to generalize to unseen data. Conversely, high errors on both training and test sets suggest underfitting, meaning the model is too simple to capture the underlying patterns in the data. Since training progresses over multiple epochs, the fitting behavior depends strongly on the number of epochs: too few may lead to underfitting, while too many may cause overfitting if proper regularization is not applied. The following table illustrates the expected values of common regression metrics on unseen data, highlighting how they indicate whether a neural network is well-fitted, overfitted, or underfitted.

Scenario	**MSE**	**MAE**	$\mathbf{R^2}$	**Interpretation**
Good Fit	Low	Low	Close to 1	Model generalizes well
Overfitting	High	High	Lower than expected	Poor test performance
Underfitting	High	High	Low	Model too simple to capture patterns

Table 1.2: Empirical formulae for calculation of labelled data of pre-trained NNs.

Example To make the concept more concrete, we consider a simple illustrative case. Let $\mathbf{y}$ denote the observed output values for a set of input data points $\mathbf{x}$. The true underlying relationship is assumed to follow the function $f(x, \beta) = \sin(x, \beta)$, with the observations subject to additive uniform noise $e_i = \mathcal{U}(0, 0.3)$. Accordingly, equation 1.41 becomes:

$$Y_i = \sin(X_i, \beta) + \mathcal{U}(0, 0.3).$$

In our example, we have 100 scalar input data points X_i , each one correspond to a scalar noisy observation Y_i. Our main task is to construct a feed-forward neural network with appropriate network parameters β for achieving a good fit between input and output data.

The training process of a feedforward neural network (FFNN) for this regression task begins by passing the input data $\mathbf{x}$ through the network's layers. Each layer computes a weighted sum of its inputs plus a bias term, followed by a nonlinear activation function, which enables the network to approximate complex, non-linear relationships such as the $\sin(x)$ function.

Initially, the weights $\mathbf{W}$ and biases $\mathbf{b}$ are randomly initialized. During training, the network generates predictions, which are compared to the true noisy target values using a suitable loss function—in this case, the Mean Squared Error (MSE) is typically used for regression tasks.

To minimize this loss, the backpropagation algorithm computes the gradient of the loss function with respect to each weight and bias. An optimization algorithm—Adam optimizer was used in our case— updates the weights and biases iteratively using gradients. This update process is repeated for multiple *epochs*, where each epoch corresponds to one complete pass through the training dataset.

Throughout training, the model adjusts its parameters to reduce prediction errors on the training data, while the validation loss indicates how well the network generalizes to unseen data. If the network has sufficient capacity (e.g., enough hidden layers and neurons) and training is stopped at an appropriate number of epochs, the FFNN can learn to approximate the underlying $\sin(x)$ function despite the presence of noise.

This process highlights the balance between model complexity, data quality, the choice of loss function, and the effectiveness of the optimization method. By carefully monitoring loss curves and evaluation metrics during training, it is possible to detect underfitting or overfitting and ensure that the trained model generalizes well to new, unseen data.

Three different scenarios are shown in Figure 1.6 to demonstrate underfitting, a good fit, and overfitting. Each scenario includes the predicted function compared to the true target function and the noisy data, as well as the corresponding training and validation loss curves plotted against the number of epochs.

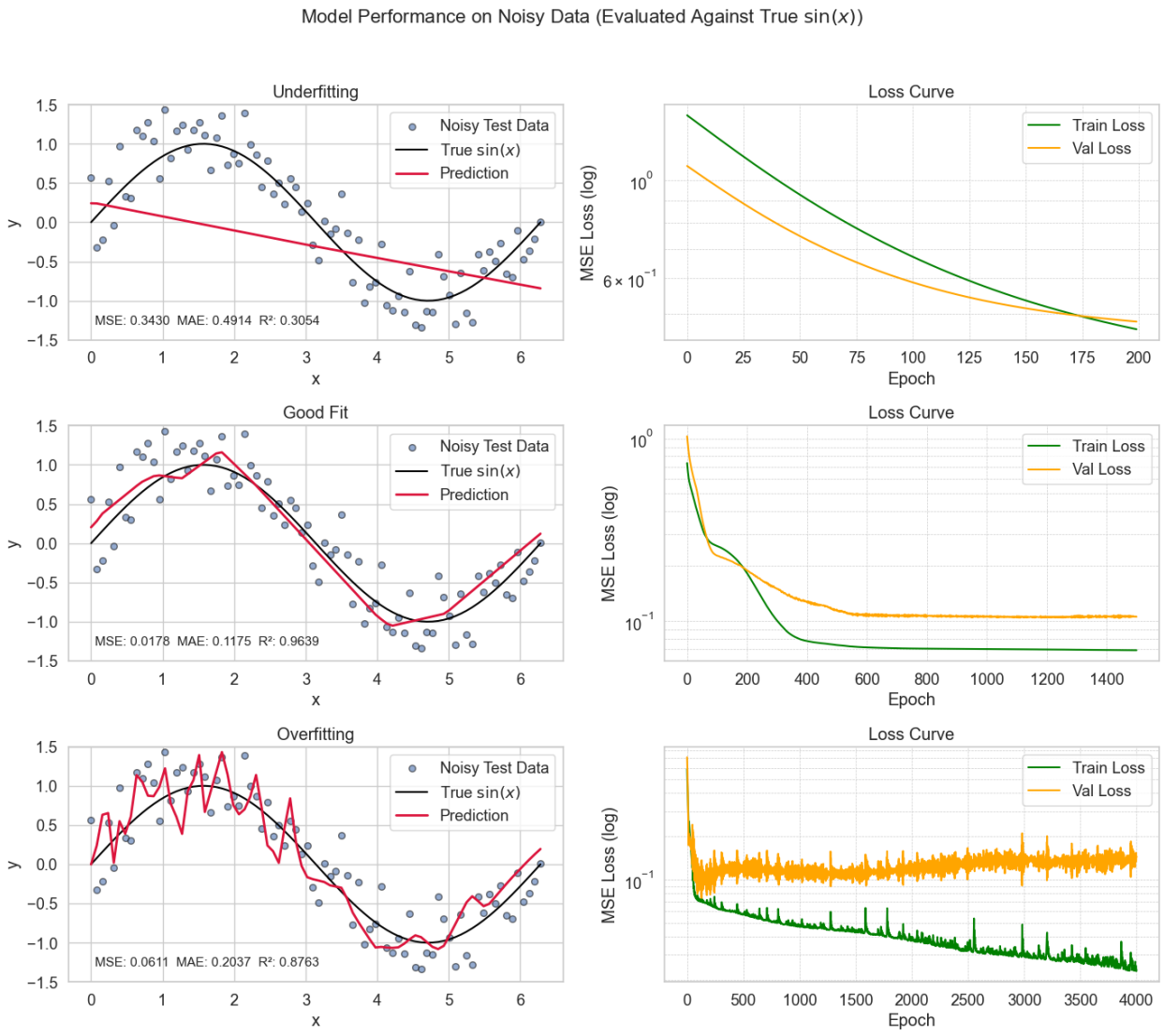


Figure 1.6: Example of underfitting, good fit, and overfitting on noisy $\sin(x)$ data using 30 training samples. **Left**: Predicted function (red) compared to the true $\sin(x)$ function (black) and noisy observations (blue points). The models' performances are quantified using MSE, MAE, and R^2 metrics. **Right**: Corresponding training and validation loss curves illustrate how loss evolves over training epochs. Model architectures used were: Underfit — [1, 10, 1], Good fit — [1, 32, 32, 1], Overfit — [1, 256, 256, 128, 64, 1], all with ReLU activations.

As shown in Figure 1.6, underfitting is characterized by high error metrics and poor alignment with the true function, while both training and validation losses decrease slowly and plateau at relatively high values. In contrast, a well-fitted model closely matches the true function and noisy data, yielding low MSE and MAE, high R^2, and training and validation losses that decrease and stabilize together with a small gap. Overfitting is evident when the prediction follows the noise in the training data too closely, resulting in low training loss but significantly higher validation loss and increased fluctuations during training. This example clearly demonstrates how analyzing predictions, error metrics, and loss curves together provides valuable insight into the model's generalization performance.

Chapter 2

Physics-Informed Neural Networks (PINNs)

In recent years, deep learning has demonstrated remarkable success in a wide range of applications, from computer vision and natural language processing to scientific computing and engineering. However, standard neural networks often rely on large amounts of labeled data and lack the ability to incorporate known physical laws explicitly. This limitation has driven the development of Physics-Informed Neural Networks (PINNs), a class of deep learning models that integrate governing physical equations into neural network architecture. PINNs have emerged as a powerful framework for solving forward and inverse problems in partial and ordinary differential equations (PDEs and ODEs), offering an efficient and flexible alternative to traditional numerical methods.

For this thesis, we will only focus on explaining the fundamentals of PINNs, with references only to their significance when used to approximate solutions of ODEs as our problem, which is going to be described in chapter 4, is governed by a system of coupled ODEs.

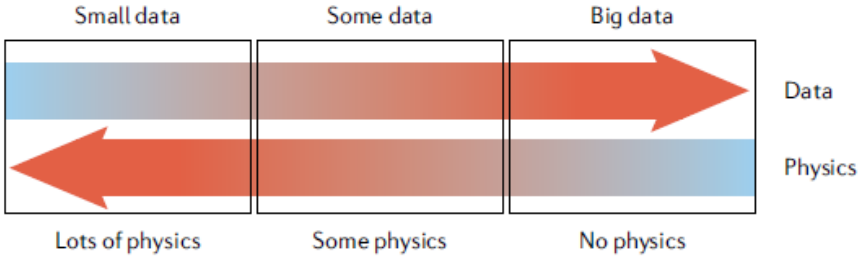


Figure 2.1: Illustration of the interplay between data availability and the reliance on physical modeling. In small-data regimes, physics-based modeling is predominant. As data availability increases, the reliance shifts toward data-driven approaches, with physics playing a diminishing role. The diagram emphasizes the continuum between physics-informed and data-driven methods depending on the amount of data and domain knowledge available. Figure adapted from [6]

2.1 Mathematical Formulation of PINNs

2.1.1 Forward PINNs

2.1.1.1 Loss function of PINNs

This results in a composite loss function that not only accounts for available data but also enforces compliance with governing differential equations. To better understand this, we now explore the specific structure of the loss function used in PINNs and how it integrates both data and physics-based constraints.

More specifically, the total loss function of a PINN can be defined as the sum of three quantities, the initial condition loss, the equation loss and the data loss. In fact, data mismatch term is applied only in case there is data provided for the variable we want to predict. In other cases, in can be neglected. Considering that data is necessary as long as the solution of an ODE cannot be solved analytically, the total loss is given by:

$$\mathcal{L}_{Total} = \mathcal{L}_{IC} + \mathcal{L}_{Equation} + \mathcal{L}_{Data} \tag{2.1}$$

where $\mathcal{L}_{IC}$, $\mathcal{L}_{Equation}$, $\mathcal{L}_{Data}$ represent the initial condition loss, the equation loss and the data loss respectively.

Let's assume that the actual value of a variable the neural network is trying to predict is y. Then, we symbolize as $\hat{y}$ the predicted value of the network. If the input of the neural network is time t and its trainable parameters are $\boldsymbol{\theta}$ then $\hat{y} = \hat{y}(t;\boldsymbol{\theta})$.All three constraints of the total loss of a PINN have the form of means squared error function. More specifically, **initial condition loss** is defined as:

$$\mathcal{L}_{IC} = (\hat{y}(t_0;\boldsymbol{\theta}) - y_0)^2 \tag{2.2}$$

where y_0 is the initial condition at a specific time point t_0 and $\hat{y}(t_0;\boldsymbol{\theta})$ the predicted value of the neural network at time t_0.

Also, **equation loss** is given by the form:

$$\mathcal{L}_{equation} = \frac{1}{N}\sum_{i=1}^{N} |r(t_i;\boldsymbol{\theta})|^2 \tag{2.3}$$

where $r(t_i;\boldsymbol{\theta})$ is the residual of the corresponding differential equation, N is the number of residual points. In a PINN, the neural network $\hat{y}(t;\boldsymbol{\theta})$ approximates $y(t)$ (when referring to ODEs). If F is a differential operator defining the ODE, then we would like $\mathcal{F}(y(t;\boldsymbol{\theta})) = 0$. In reality, we want $\mathcal{F}(y(t;\boldsymbol{\theta}))$ to approximate zero, as equality to zero can never be achieved. The closer to the zero the more informed the neural network is about the physics of the system.

Finally, **data loss** is defined as:

$$\mathcal{L}_{Data} = \frac{1}{M}\sum_{i=1}^{M} |\hat{y}(t_i;\boldsymbol{\theta}) - y(t_i)|^2 \tag{2.4}$$

where M represents the total number of measurements(data points).
By replacing 2.2, 2.3 and 2.4 to 2.1 the total loss of the PINN can be expressed as:

$$\mathcal{L}_{Total} = (\hat{y}(t_0;\boldsymbol{\theta}) - y_0)^2 + \frac{1}{N}\sum_{i=1}^{N} |r(t_i;\boldsymbol{\theta})|^2 + \frac{1}{M}\sum_{i=1}^{M} |\hat{y}(t_i;\boldsymbol{\theta}) - y(t_i)|^2 \tag{2.5}$$

Note that the number of residual points N might differ from the number of measurements M.
This loss term is used to approximate the solution of a single ODE. However, in most cases, the governing equations a physical system are more than one. The problem we are going to solve in chapter 3, includes 6 coupled differential equations. So, for systems with multiple unknown fields the loss function can then be rewritten as:

$$\mathcal{L}_{\text{Total}} = \sum_{l=1}^{L}\left[(\hat{y}_l(t_0;\boldsymbol{\theta}) - y_{0,l})^2 + \frac{1}{N_l}\sum_{i=1}^{N_l} |r_l(t_i;\boldsymbol{\theta})|^2\right] + \sum_{k=1}^{K}\left[\frac{1}{M_k}\sum_{i=1}^{M_k} |\hat{y}_k(t_i;\boldsymbol{\theta}) - y_k(t_i)|^2\right] \tag{2.6}$$

where K and L denote the number of variables that can be measured as well as the equations, respectively; M_k and N_l are the number of measurements for the kth variable and the number of residual points for the lth equation, respectively.

While this mathematical formulation seems logical often gives good results when dealing with multiple ODEs , in cases such as the one we are working on this expression for the loss function appears to be problematic. The reason is that the variables involved in the ODEs may differ significantly in scale. As a result, the gradients associated with the loss terms of large-scale variables dominate those of smaller-scale variables. This leads the model to focus primarily on approximating the large-scale dynamics, while the small-scale components are effectively neglected during training. There are many possible methods that could be implemented for tackling this problem. The approach we chose in this work is to multiply each loss term in equation 2.6 so that in the first epoch of the training process the total loss function will be as closer to 1 a possible. With this adjustment the expression for the total loss function will be :

$$\mathcal{L}_{\text{Total}} = \sum_{l=1}^{L} \left[\lambda_{l,IC} \left(\hat{y}_l(t_0; \boldsymbol{\theta}) - y_{0,l} \right)^2 + \frac{1}{N_l} \lambda_{l,Eq} \left| r_l(t_i; \boldsymbol{\theta}) \right|^2 \right] + \sum_{k=1}^{K} \left[\frac{1}{M_k} \sum_{i=1}^{M_k} \lambda_{k,Data} \left| \hat{y}_k(t_i; \boldsymbol{\theta}) - y_k(t_i) \right|^2 \right] \quad (2.7)$$

where $\lambda_{l,IC}$, $\lambda_{l,Eq}$, $\lambda_{k,Data}$ are the corresponding weights for initial condition, equation and data loss terms respectively. The results and the effectiveness of this approach will be presented in chapter 4

2.1.1.2 Structure of PINNS

In most cases, a Physics – Informed Neural Network is a FFNN (Feed – Forward Neural Network), the structure of which was explained in Chapter 1 and usually the dense of its layers is stable. In other words, the number of neurons in each layer is the same.

PINNs that they are trained to predict the solution of an ODE take as input only time points. The number of the outputs is congruent with the number of the variables we are trying to predict and, by extension, the number of the governing equations of the system.

Another important consideration is that in the last layer, the output values range between 0 and 1 as we make the use of activation functions. This means that without any other modifications the networks predictions are not scaled appropriately, and they lack physical meaning. So, it is important that output values are multiplied by a factor so that the values are scaled appropriately, in agreement with the physical quantities they represent. In Figure 2.2 there is an illustration of a simple PINN that uses a loss function defined by 2.5 in order to approximate a solution of an ODE.

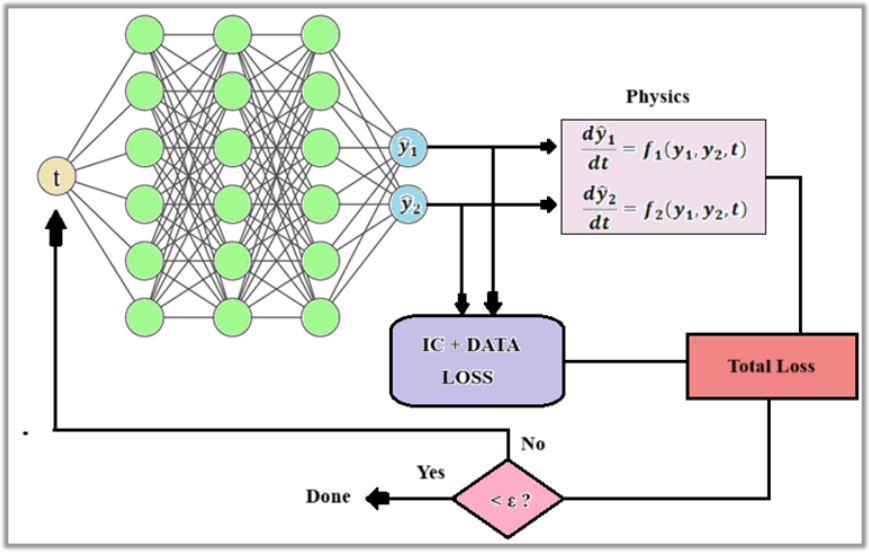


Figure 2.2: Illustration of a Physics Informed Neural Network that is trained to predict the solutions of two coupled ODES. The total loss is the summation of three losses: the initial condition loss, the data loss and the physics loss (or equation loss). The backpropagation process ends when the value of the loss function is smaller than a specific value ϵ.

2.1.2 Training Process

After a detailed mathematical analysis of how neural networks learn, particularly in the context of Physics-Informed Neural Networks (PINNs), we now present an example illustrating how a PINN is trained and ultimately learns to approximate the dynamics of a variable by incorporating the underlying physics of the system. In fact, we are going to explain how we can built a PINN, which actually has a structure of a FFNN, that approximates the solution of equation 3.6. The reason we choose this example is the simplicity of the equation and also the fact that state variable $\tilde{u}_{vgt}$ which describes the VGT actuator position of a diesel engine is independent from the other state variables. This means that we can construct a model with a single PINN, the loss function of which will include only an initial condition and an equation loss term while no data for VGT actuator position $\tilde{u}_{vgt}$ will be used during the training process. According to equation 2.7 with $L = 1$, as we are constructing a PINN for predicting the dynamics of a single variable by using a single ODE with its initial condition, and $K = 0$ as long as we have assumed that no data for $\tilde{u}_{vgt}$ are available for training , we have the following expression for this loss:

$$\mathcal{L}_{\tilde{u}_{vgt}} = \lambda_{IC,\tilde{u}_{vgt}}\mathcal{L}_{IC,\tilde{u}_{vgt}} + \lambda_{Eq,\tilde{u}_{vgt}}\mathcal{L}_{Eq,\tilde{u}_{vgt}}\lambda_{Eq,\tilde{u}_{vgt}} = \left(\mathcal{N}_{\tilde{u}_{vgt}}(t_0;\boldsymbol{\theta}) - \tilde{u}_{vgt,0}\right) + \lambda_{Eq,\tilde{u}_{vgt}} \sum_{i=1}^{N} \frac{d\mathcal{N}_{\tilde{u}_{vgt}}(t_i;\boldsymbol{\theta})}{dt} - \left(\frac{1}{\tau_{vgt}}(u_{egr}(t-\tau_{dvgt}) - \mathcal{N}_{\tilde{u}_{vgt}}(t_i;\boldsymbol{\theta})\right) \tag{2.8}$$

where $\mathcal{N}_{\tilde{u}_{vgt}}(t_i;\boldsymbol{\theta})$ and $\mathcal{N}_{\tilde{u}_{vgt}}(t_i;\boldsymbol{\theta})$ the PINN's prediction for variable $\tilde{u}_{vgt}$ at initial condition t_0 and the whole time domain respectively. The calculation of $\frac{d\mathcal{N}_{\tilde{u}_{vgt}}}{dt}$ is achieved with automatic differentiation.

The FFNN for predicting the dynamics of $\tilde{u}_{vgt}$ takes as input a time vector $\boldsymbol{t}$ with shape 301×1 and time points from 0 to 60s with time step equal to 0.2s. After the end of each epoch, the neural network's output vector $\boldsymbol{y}$ is being transformed appropriately in order to give the variable $\tilde{u}_{vgt}$ in its physical scale. More specifically, $\tilde{u}_{vgt} = 100 \times S(\boldsymbol{y})$. An illustration of this FFFN is given in the following figure.

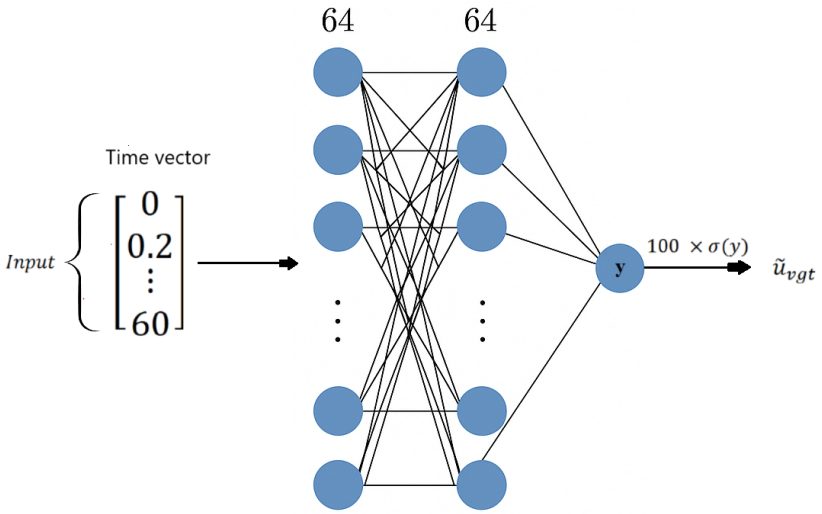
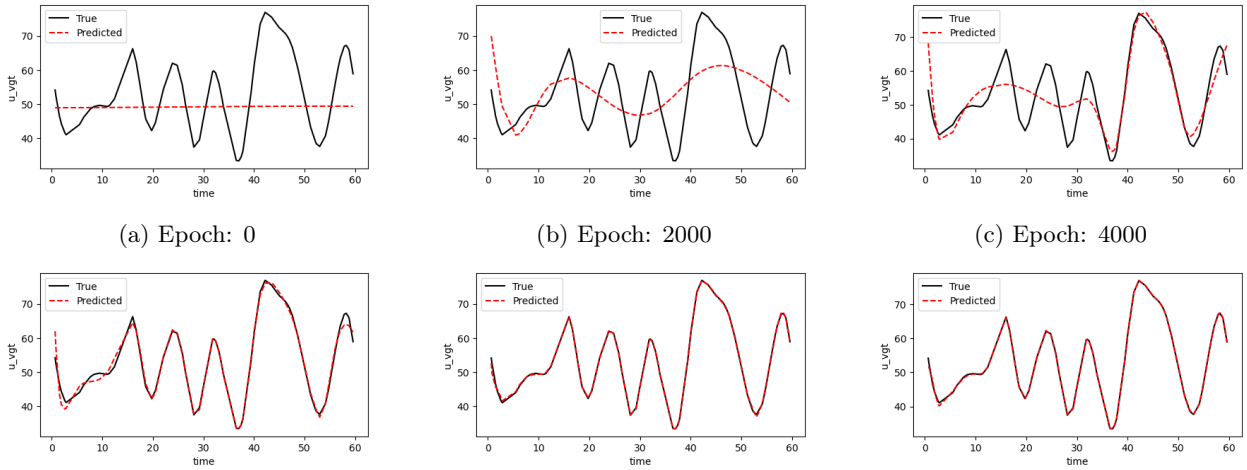


Figure 2.3: Feed-Forward Neural Network for predicting the dynamics of $\tilde{u}_{vgt}$. The input $\boldsymbol{x}$ of this network is a time vector $\mathbf{t}$ and the output is given by $\hat{y} = \mathcal{N}_{\tilde{u}_{vgt}}(\mathbf{t};\boldsymbol{\theta}) = 100 \times S(\boldsymbol{y})$, where $\boldsymbol{y}$ is the raw output of the FFNN and S is the sigmoid function. The network constructed for the approximation of the VGT actuator position dynamics consists of two hidden layers with 64 neurons each. For visualization purposes, only 16 neurons are depicted in each hidden layer.

The whole training process has been described analytically in subsection 1.2.5 where we had to solve a regression problem as well. The main difference here is that now we do not have observations as data , but our model is informed with the physics that governs our system. In an other point of view, we could say that

we want to minimize a loss(cost) function in order to satisfy the corresponding ODE of the variable $\tilde{u}_{vgt}$. However, mapping appropriately the inputs with the desired output by finding the unknown function that describes mathematically the relationship between the two, still remains the major goal. The loss function that is used for the training of the corresponding PINN is given by equation 2.8 while we make usage of Adam optimizer. Also the neurons' activation function used is tanh in every hidden layer. The initial condition for $\tilde{u}_{vgt}$ at time t=0 is $\tilde{u}_{vgt,0} = 90.0317\%$. The training progression of the PINN responsible for predicting the dynamics of $\tilde{u}_{vgt}$ is illustrated in 2.4f.



(a) Epoch: 0 (b) Epoch: 2000 (c) Epoch: 4000

(d) Epoch: 8000 (e) Epoch: 20000 (f) Epoch: 50000

Figure 2.4: Training progression of the PINN model over different epochs. The subfigures show the predicted solution $\tilde{u}_{vgt}$ at various training stages: from the initial epoch (0) to intermediate steps (2000, 4000, 8000, 20000) and the final result at epoch 50000. The results demonstrate how the model gradually improves its approximation of the solution of the ODE for predicting the dynamics of $\tilde{u}_{vgt}$.

The training loss over the iterations(epochs) during the training process is depicted in 2.5

Figure 2.5: Values of the loss function during the training process of the PINN responsible for predicting the dynamics of u_{vgt}

As we can see in the figure above, the loss generally decreases monotonically, indicating that the neural network progressively improves its approximation of the solution while better satisfying the governing equations. The initial rapid drop reflects the network's quick adaptation to initial conditions, followed by a slower reduction as the physics residuals are minimized.

Another significant fact that we should not overlook is the input's transformation. More specifically, it is common every input in a PINN to be rescaled appropriately for the training process as it results in better work for activation functions and gradient flow is being improved. To be more precise, activation function

$tanh(x)$ saturates to -1 or 1 for large $|x|$. In those regions gradients vanish and training slows down or stalls. At the same time, while losses involve derivatives of the neural network with respect to its inputs, inconsistent input scale will lead to difference in magnitude of loss gradients. Consequently, Proper scaling makes these derivatives numerically stable and helps the optimizer balance terms in the loss function. The time rescaling from [0,60] to [-1,1] is achieved as following:

$$\tilde{t} = 2\left(\frac{t}{60}\right) - 1 \tag{2.9}$$

$$t = 30(\tilde{t} + 1) \tag{2.10}$$

This rescaling requires careful handling of the governing equations. When the time domain is transformed, the chain rule must be applied consistently to the derivatives in the ODE to preserve the correct physical relationships. For instance, a first-order time derivative will introduce a scaling factor equal to the inverse of the scaling applied to the time variable. Neglecting this adjustment would result in the neural network learning an incorrect form of the underlying physics, ultimately leading to inaccurate predictions despite successful minimization of the loss function.

Mathematically, suppose the original time variable t is defined on the interval [0,T] and is rescaled to a normalized variable $\tilde{t} \in [-1, 1]$ via a linear transformation given by equation 2.9, . In this case, the relationship between the derivatives in the original and rescaled coordinates is determined by the chain rule. Specifically, a first-order time derivative of a variable u transforms as:

$$\frac{du}{dt} = \frac{du}{d\tilde{t}} \cdot \frac{d\tilde{t}}{dt} = \frac{2}{T}\frac{du}{d\tilde{t}} \Rightarrow \frac{du}{d\tilde{t}} = \frac{T}{2}\frac{du}{dt} \tag{2.11}$$

In our case , while T=60s ,every derivative of our variables with respect to time should be multiplied with a scaling factor equal to 30 according to the chain rule. This ensures that the PINN accurately approximates the true solution while benefiting from the numerical stability provided by input scaling.

2.1.3 Inverse PINNs (Parameter Identification)

While PINNs effectively solve forward problems where the governing equations and parameters are known, many real-world applications require solving inverse problems, where certain aspects of the system—such as unknown parameters, hidden functions, or even the governing equations themselves—must be inferred from observed data. This leads to the development of Inverse Physics-Informed Neural Networks (I-PINNs), an extension of PINNs designed to recover missing information by leveraging available physical constraints.

In addition, the difference in training between PINNS and I-PINNs is that the second networks, despite minimizing variables of the ODES, should also perform appropriately in order to predict some unknown parameters $\boldsymbol{\Lambda}$ that are included in the differential equations.

This extension results in change to the total trainable parameters of the network. More specifically, while there are more trainable parameters of the network, they can be defined as:

$$\boldsymbol{\theta'} = \{\boldsymbol{\theta}, \boldsymbol{\Lambda}\} = \{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{\Lambda}\} \tag{2.12}$$

Taking this change into account we have the following expression for the total loss term of a PINN which is constructed to approximate the solution of an ODE with unknown parameters $\boldsymbol{\Lambda}$:

$$\mathcal{L}_{\text{Total}}(\boldsymbol{\theta}, \boldsymbol{\Lambda}) = \left(\hat{y}(t_0; \boldsymbol{\theta}, \boldsymbol{\Lambda}) - y_0\right)^2 + \frac{1}{N}\sum_{i=1}^{N} \left|r(t_i; \boldsymbol{\theta}, \boldsymbol{\Lambda})\right|^2 + \frac{1}{M}\sum_{i=1}^{M} \left|\hat{y}(t_i; \boldsymbol{\theta}, \boldsymbol{\Lambda}) - y(t_i)\right|^2 \tag{2.13}$$

In full correspondence with the theory of PINNs in the previous subsection, the total loss function of an I-PINN used to approximate simutaenously the solutions of multiple odes is given by equation 2.7 with the modification $\boldsymbol{\theta} \rightarrow \boldsymbol{\theta'}$.

Chapter 3

Problem Description and Methodology

3.1 Physical system of a mean value diesel engine

3.1.1 Diesel Engine's components

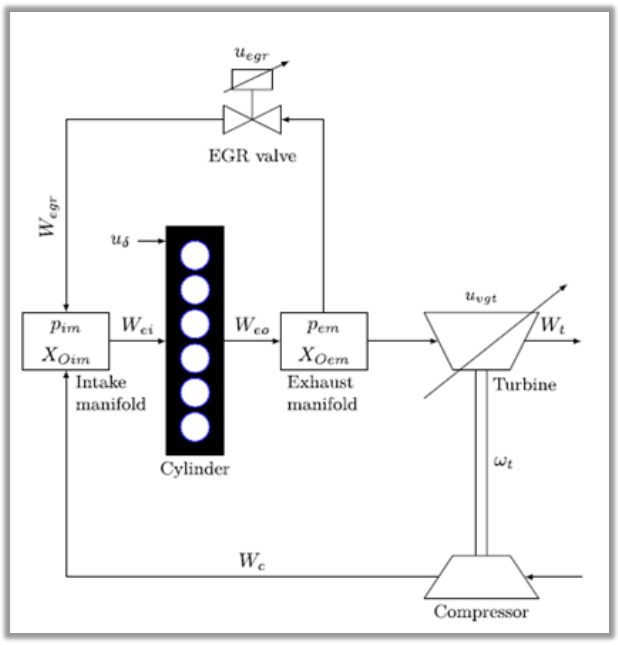


Figure 3.1: Schematic diagram of a mean value diesel engine.

The system we are going to deal with is a diesel engine, which is initially proposed in Ref[14]. In summary, the engine model mainly consists of a variable – geometry turbocharger (VGT), an exhaust gas recirculatory (EGR), the intake manifold, the exhaust manifold, the cylinder and an EGR valve system. An illustration of this system can be found on Figure 3.1. We will provide brief information for each of the six components of the engine:

- **Variable-Geometry Turbocharger:**It is a type of turbocharger that adjusts its turbine geometry dynamically to optimize performance across a wide range of engine speeds.
- **Exhaust gas recirculatory:**It is a nitrogen oxide (NO_x) emissions reduction technique. The goal of EGR is to reduce NO_x production by reducing the combustion temperatures.

- **Intake manifold:**responsible for distributing air evenly to the engine cylinders for combustion.
- **The exhaust manifold:**collects the exhaust gases from multiple cylinders into one pipe.
- **Cylinder:**It houses the piston, which moves up and down to convert fuel combustion into mechanical power. Diesel engines typically have 2 to 16 cylinders.
- **EGR valve system:**The EGR valve is the main component of the EGR system, and it's normally closed. It connects the exhaust manifold to the intake manifold.

The control input vector of the system is $u = \{u_\delta, u_{egr}, u_{vgt}\}$, -in which u_δ is the mass of injected fuel, u_{egr} is the EGR valve position and u_{vgt} is the VGT actuator position. The engine speed n_e is considered an input vector as well. Also, this model includes six states: the intake manifold pressure p_{im}, the exhaust manifold pressure p_{em}, the turbo speed ω_t, the VGT actuator dynamics $\tilde{u}_{vgt}$ and both mass fractions in the intake and exhaust manifold $X_{O_{im}}$, $X_{O_{em}}$ respectively. These mass fractions are assumed to be constant and known.

3.1.2 Governing equations

The governing ordinary differential equations that are used to describe the gas flow dynamics are the following:

$$\frac{dp_{im}}{dt} = \frac{R_a T_{im}}{Vim}(W_c + W_{egr} - W_{ei}) \tag{3.1}$$

$$\frac{dp_{em}}{dt} = \frac{R_e T_{em}}{Vem}(W_{eo} - W_t - W_{egr}) \tag{3.2}$$

$$\frac{d\omega_t}{dt} = \frac{P_t \eta_m - P_c}{J_t \omega_t} \tag{3.3}$$

$$\frac{d\tilde{u}_{egr1}}{dt} = \frac{1}{\tau_{egr1}}[u_{egr}(t - \tau_{degr}) - \tilde{u}_{egr1}] \tag{3.4}$$

$$\frac{d\tilde{u}_{egr2}}{dt} = \frac{1}{\tau_{egr2}}[u_{egr}(t - \tau_{degr}) - \tilde{u}_{egr2}] \tag{3.5}$$

$$\frac{d\tilde{u}_{vgt}}{dt} = \frac{1}{\tau_{vgt}}[u_{egr}(t - \tau_{dvgt}) - \tilde{u}_{vgt}] \tag{3.6}$$

The are also two important equations, that indicate a relationship between residual gas fraction x_r and the temperature T_1 when the inlet valve closes after the intake stroke and mixing. The two equations are significant for calculating terms of the forementioned equations of the state variables and also play a crucial role for the solution of our problem. They have the following expressions:

$$T_1 = x_r T_e + (1 - x_r) T_{im} \tag{3.7}$$

$$x_r = \frac{\Pi_e^{1/\gamma_\alpha}\, x_p^{-1/\gamma_\alpha}}{r_c\, x_v} \tag{3.8}$$

The physical meaning of the quantities included in equations 3.1 - 3.8 are presented in Table ... in Appendix ... Also further mathematical insight of these equations is provided in Appendix B while more interested readers are encouraged to consult Ref [14]

3.1.3 Parameters of interest

In this work, a Physics-Informed Neural Network (PINN) model is developed to estimate several unknown or hard-to-measure parameters in a diesel engine. These parameters are essential for accurately capturing the engine's performance and emissions behavior. The target variables include:

- $\boldsymbol{\eta_{sc}}$: the compensation factor of non-ideal cycles, representing the effectiveness of replacing the burnt gases with fresh air during the scavenging process.
- $\boldsymbol{h_{tot}}$: the total heat transfer coefficient of the exhaust pipes, which accounts for the heat losses through the cylinder walls and other engine components.
- $\boldsymbol{A_{vgtmax}}$: the maximum effective area of the Variable Geometry Turbocharger (VGT), influencing the boost pressure and turbine performance.
- $\boldsymbol{A_{egrmax}}$: the maximum effective area of the Exhaust Gas Recirculation (EGR) valve, affecting the recirculated exhaust gas fraction and NO_x emissions.

These unknown parameters correspond to the vector $\boldsymbol{\Lambda} = \{\eta_{sc}, h_{tot}, A_{vgtmax}, A_{egrmax}\}$ we discussed in the theory of I-PINNS in the previous subsection. A main task in our work includes the approximation of the theoretical values of these parameters given a specific diesel engine system as shown in Table 3.1.

Unknown	$\boldsymbol{\eta_c}$	$\boldsymbol{h_{tot}}$	$\boldsymbol{A_{egrmax}}$	$\boldsymbol{A_{vgtmax}}$
Value	1.1015	96.2755	4.0×10^{-4}	8.4558×10^{-4}

Table 3.1: Theoretical values of unknown parameters for the diesel engine system.

3.2 Methodology

The problem we commit to solve in this thesis is presented in [8]. More specifically, we are trying to construct a model that predicts the dynamics of the state variables for the diesel engine and also the values of unknown parameters we discussed in the previous section. During the training process of the model we possess data for intake manifold pressure p_{im}, exhaust manifold pressure p_{em}, turbo speed ω_t and $Wegr$. Of course, we also inform the system with the necessary physics , described by the forementioned equations. In addition, we are interested in testing the performance of the model in noisy and sparse data.

In our work we generate our own data and provide our own architectures for all the Neural Networks in the present study. Moreover, we implement strategies that seem to result in better performance from the model in [8] as we manage to deal successfully with the problem's complexity.

3.2.1 Generating the Data

Data used in this work was generated through a Simulink simulation built in MATLAB, the same simulation that was used for the work in [14]. This simulator is embedded with the physics of the diesel engine system and is used in order to provide two types of data:

- Data used for training the model used for the prediction of the state variables' dynamics and unknown parameters of the engine.
- Data for training pre-trained NNs that will be used as surrogates in order to replace analytical functions included in state variables' ODEs. The importance and utility of these pre-trained NNs is highlighted in the next subsection.

For running this simulation we at first have to provide the input signals u_{egr}, u_{vgt}, u_δ and n_e as well as the time domain , which is equal to 0-60s (1 minute).Input signal u_δ and n_e are considered to remain

constant with values 110 and 1500 respectively while we generate random signals for input variables u_{egr} and u_{vgt} with signal ranges between 0 and 100 provided that these two signals denote valve positions (0% indicates fully closed while 100% indicates fully opened). All input signals consist of 301 time-discrete values sampled over the interval $[0, 60]$ seconds, with a constant time step of $dt = 0.2\,\text{s}$. The input signals for input variables u_{egr} and u_{vgt} are depicted in the following figure:

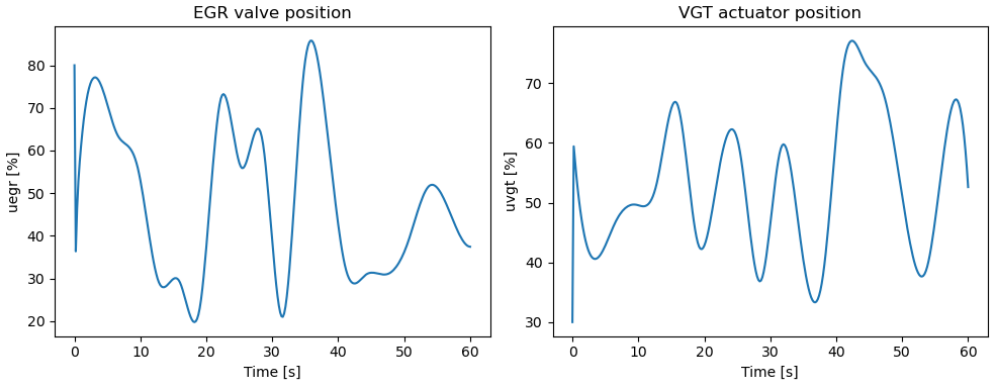


Figure 3.2: Input signals of EGR valve position u_{egr} and VGT actuator position u_{vgt}

Despite the fact that our simulator provides a satisfying amount of data for the physical quantities of our system it does not provide data for the state variables $\tilde{u}_{vgt}$, $\tilde{u}_{egr1}$, $\tilde{u}_{egr2}$, ω_t .However, data for the dynamics of these variables are necessary not only for training the model but also for validating its own performance. In other words, we have to find a method for generating data for these variables. The tactic we implement for this problem is solving equations 3.3 - 3.6 with numerical integration. To be more precised, the method we used to solve numerically these equations is the explicit Runge - Kutta of order 5(4). The selection of this method for solving this system occurs from the fact that not only handles stiff and non-stiff problems successfully but also provides higher accuracy compared to lower-order Runge – Kutta methods. Further mathematical analysis for this process is provided in Appendix A.

After running our simulation and solving numerically ODEs for extracting data for the missing values we have generated a clean dataset with a total of 2200 samples for each physical quantity. This means that data were not limited to the specific time points included in the input variables, but rather sampled densely across the entire time domain.

The initial conditions we obtain for the state variables in our case, after following the previous process are presented in the following table:

	State variable	**Initial Condition**
1	p_{im}	1.0134e+05
2	p_{em}	1.0237e+05
3	ω_t	1.5827e+03
4	$\tilde{u}_{egr1}$	18.2518
5	$\tilde{u}_{egr2}$	18.1813
6	$\tilde{u}_{vgt}$	90.0317
7	T_1	3.1495e+02
8	x_r	2.0222e-02

Table 3.2: Initial Conditions for the state variables

Table 3.3 presents key physical quantities obtained from the Simulink simulation, which are essential for training the pre-trained neural networks.

	Description	**Symbol**
1	Intake manifold pressure	p_{im}
2	Exhaust manifold pressure	p_{em}
3	Engine speed	ω_t
4	VGT position	$\tilde{u}_{vgt}$
5	EGR position	$\tilde{u}_{egr}$
6	Engine speed	n_e
7	Total mass flow from the intake manifold into the cylinders	W_{ei}
8	Intake manifold temperature	T_{im}
9	EGR mass flow	W_{egr}
10	Exhaust manifold temperature	T_{em}
11	Turbine mass flow	W_t
12	Compressor mass flow	W_c
13	Temperature after the compressor	T_c
14	Ambient temperature	T_{amb}
15	Ambient pressure	p_{amb}

Table 3.3: Variables required for training the pre-trained neural networks

Data for EGR position $\tilde{u}_{egr}$ is given by the expression:

$$\tilde{u}_{egr} = K_{egr}\tilde{u}_{egr1} - (K_{egr} - 1)\tilde{u}_{egr2} \tag{3.9}$$

where K_{egr} is constant. Also, ambient temperature T_{amb} and ambient pressure p_{amb} are considered constant with values 298.4636K and 10.111 Pa respectively.

From the whole clean dataset we create training and testing datasets both for the training of our model and also for the training of pre-trained NNs. The training dataset for model's training includes variable values for $dt = 0.2$s while the testing set consists of 80 random samples from the initial dataset. In the case of the pre-trained neural networks, the initial dataset was split into 80% for training and 20% for testing. It is important to note that, across all datasets, the values of different variables are temporally aligned — that is, the i-th element of each variable corresponds to the same time point.

As we have already mentioned, we are also concerned about the behavior of the model on noisy and sparse training data. For the noisy data sets we apply gaussian noise on the existed clean training dataset while the testing data set remains the same for all cases. To be more precise, if we let u be a signal that describes the dynamics of a variable in our problem then a noisy signal $\tilde{u}$ will occur by multiplying a standard deviation σ with signal's range:

$$\tilde{u} = \sigma \times \text{signal range}(u) \tag{3.10}$$

$$\text{signal range}(u) = max(u) - min(u) \tag{3.11}$$

Model's performance will be tested under 1%, 2%, 3% and 4% white noise.

For sparse datasets we simply increase the time distance dt between training points. Again , the model's behavior will be examined for 0.5s, 1s,2s, 5s time distance between training points.

3.2.2 Pre-Trained Neural Networks

The main reason for constructing pre-trained neural networks, that are used as approximators of functions and replace them during the training process, is that they can evaluate complex functions much faster than computing an analytical form directly. Moreover, it is a fact that despite the existence of analytical forms, in practice, we may have only sampled data or noisy measurements.As the analytical forms of the functions in the system of ODEs is considered to be engine-specific, for the approximation of the function will be used an empirical formulae.

The pre-trained neural networks in this work are also Feed-Forward Neural Networks (FFNNs)—the functionality of which was presented in Chapter 1—with network parameters symbolized with $\boldsymbol{\theta}^P$. The loss function considered for training the i-th pre-trained networks is given by the form:

$$\mathcal{L}_i(\boldsymbol{\theta}_i^P) = \frac{1}{n_i}\sum_{j=1}^{n_i}\left(y_i^{(j)} - \hat{y}_i^{(j)}\right)^2 = \frac{1}{n_i}\sum_{j=1}^{n_i}\left(y_i^{(j)} - \mathcal{N}_i^{(P)}(\mathbf{x}_i^{(j)};\boldsymbol{\theta}_i^P)\right)^2, \quad i = 1, 2, \ldots, 6 \tag{3.12}$$

Index i corresponds to one of the six in total pre-trained neural networks used as surrogates for approximating analytical functions, y_i are the target values for the i-th pre-trained neural network which are calculated according to empirical formulaes in Table 3.4, $\hat{y_i}$ is the prediction of each i-th pre-trained neural network and n_i is the number of labelled dataset for the i-th network.

#	Quantity	Symbol	Surrogate	Empirical Formula
1	Volumetric efficiency	η_{vol}	$\mathcal{N}_1^{(P)}$	$\frac{120\, W_{ei}\, R_a\, T_{im}}{p_{im}\, n_e\, V_d}$
2	EGR effective area ratio	f_{egr}	$\mathcal{N}_2^{(P)}$	$\frac{W_{egr}\sqrt{T_{em}R_e}}{A_{egr,max}\, p_{im}\, \Psi_{egr}}$
3	VGT area × choking	$f_{vgt} f_{\Pi_t}$	$\mathcal{N}_3^{(P)}$	$\frac{W_t\sqrt{T_{em}R_e}}{A_{vgt,max}\, p_{em}}$
4	Turbine mech. efficiency	η_{tm}	$\mathcal{N}_4^{(P)}$	$\frac{P_t\, \eta_m}{W_t\, c_{pe}\, T_{em}(1-\Pi_t^{1-1/\gamma_e})}$
5	Compressor efficiency	η_c	$\mathcal{N}_5^{(P)}$	$\frac{T_{amb}(\Pi_c^{1-1/\gamma_a}-1)}{T_c - T_{amb}}$
6	Volumetric flow	Φ_c	$\mathcal{N}_6^{(P)}$	$\frac{R_a T_{amb}}{p_{amb}\pi R_c^3 \omega_t} W_c$

Table 3.4: Empirical formulae used to generate labelled data for pre-trained surrogate neural networks.

The inputs and the analytical functions to be approximated for each of the six pre-trained neural networks are summarized in Table 3.5.

The training process of each surrogate is quite similar with the regression example we discussed in 1.2.5. The main difference is that the inputs $\boldsymbol{x}$ of the pre-trained NNs are not scalars but vectors. The input vector's dimension corresponds to the number of variables each neural network receives as input, while the shape of each element represents the number of time points from which data has been collected(in our case we have 2200 samples in total for each variable). However, the initial goal still remains the same with the

Variable	$\boldsymbol{\eta_{vol}}$	$\boldsymbol{f_{egr}}$	$\boldsymbol{F_{vgt,\Pi_t}}$	$\boldsymbol{\eta_{tm}}$	$\boldsymbol{\eta_c}$	$\boldsymbol{\Phi_c}$
Surrogate	$\mathcal{N}_1^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_1^P})$	$\mathcal{N}_2^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_2^P})$	$\mathcal{N}_3^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_3^P})$	$\mathcal{N}_4^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_4^P})$	$\mathcal{N}_5^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_5^P})$	$\mathcal{N}_6^{(P)}(\boldsymbol{x}, \boldsymbol{\theta_6^P})$
Input ($\boldsymbol{x}$)	$\{p_{im}, n_e\}$	$\{\tilde{u}_{egr}\}$	$\{\tilde{u}_{vgt}, \Pi_t\}$	$\{\omega_t, T_{em}, \Pi_t\}$	$\{W_c, \Pi_c\}$	$\{T_{amb}, \Pi_c, \omega_t\}$
Equation	(B.4)	(B.15)	(B.17)	(B.21)	(B.29)	(B.27)

Table 3.5: Pre-trained NN surrogates for the approximation of analytical functions. We provide the symbolisms and inputs of each surrogate as well as the analytical equations that each network is trained to approximate, that are analyzed in Appendix B

forementioned example : mapping appropriately the inputs with the desired output by finding an unknown function that describes mathematically the relationship between them. Last but not least, input variables are all rescale to [0,1].If we let x be an input variable in time space then we can take the undimensioned form $\tilde{x}$ of this input variable with the tranformation :

$$\tilde{x} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.13}$$

Proper input transformation of the input variables, ensures that features are on comparable scales, preventing dominance by variables with larger ranges and enhancing numerical stability during training. Additionally, transforming inputs can help the network capture complex patterns more effectively, leading to faster convergence and more accurate function approximation.

Our goal is for the surrogate models to accurately replicate the analytical functions that characterize specific engine models. Although the empirical formulas provide a general representation of the physical quantities in diesel engine models, it is obvious that they must also closely align with the unique characteristics of each individual engine.

The architecture of each pre-trained NN, $\mathcal{N}_i^P(\boldsymbol{x}_i^P; \boldsymbol{\theta}_i^P)$—specifically, the number of hidden layers and neurons per layer—is chosen to balance approximation capability and training stability. As we have already explained, a network that is too small may underfit, failing to capture the complexity of the solution, while an excessively large network may overfit limited data or introduce unnecessary training instabilities. In this work, the network depth and width were selected empirically by starting with simple architectures and gradually increasing complexity until the residual loss and prediction errors reached a satisfactory level without signs of overfitting, such poor generalization on validation points. The final architectures represent a trade-off between model expressiveness and computational efficiency.

During the training process we also apply a learning rate scheduler which monitors a specific performance metric during training, such as the validation loss or accuracy. When this metric stops improving for a defined number of epochs, known as the *patience*, the learning rate is reduced by a specified factor. Reducing the learning rate at plateaus helps the optimizer make smaller, more precise weight updates, which is crucial when approaching local or global minima. This mechanism prevents the training process from oscillating or overshooting once the model is near convergence. Overall, this scheduler provides an adaptive strategy to adjust the learning rate dynamically, balancing faster learning in the early stages with finer tuning in the later stages, thus often leading to better generalization and lower final loss.

It is notable from Table 3.7 that in some pre-trained neural networks $\mathcal{N}_2^P(\boldsymbol{x}_2^P; \boldsymbol{\theta}_2^P)$ and $\mathcal{N}_6^P(\boldsymbol{x}_2^P; \boldsymbol{\theta}_6^P)$ we applied an output transformation using the Sigmoid activation function. The reason is that the output of the variables f_{egr} and Φ_c are bounded in the range (0,1). This is because the sigmoid maps any real-valued input smoothly and monotonically into the $(0, 1)$ range, ensuring that model predictions remain physically meaningful and within valid limits.

Finally, every surrogate's prediction is compared with the corresponding analytical function that is considered to represent the same physical variable. Analytical mathematical analysis of these functions is provided in appendix B

Pre-Trained NNs	Network size	0utput	Output transformation	Learning rate
$\mathcal{N}_1^P(\boldsymbol{x}_1^P;\boldsymbol{\theta}_1^P)$	[2,10,10,10,1]	η_{vol}	-	10^{-3}
$\mathcal{N}_2^P(\boldsymbol{x}_2^P;\boldsymbol{\theta}_2^P)$	[1,32,32,32,1]	f_{egr}	$S(\boldsymbol{x_2^P})$	10^{-3}
$\mathcal{N}_3^P(\boldsymbol{x}_3^P;\boldsymbol{\theta}_3^P)$	[2,32,32,1]	F_{vgt,Π_t}	-	5×10^{-3}
$\mathcal{N}_4^P(\boldsymbol{x}_4^P;\boldsymbol{\theta}_4^P)$	[2,16,16,16,16,1]	η_{tm}	-	10^{-3}
$\mathcal{N}_5^P(\boldsymbol{x}_5^P;\boldsymbol{\theta}_5^P)$	[2,4,4,4,1]	η_c	-	10^{-3}
$\mathcal{N}_6^P(\boldsymbol{x}_6^P;\boldsymbol{\theta}_6^P)$	[3,20,20,20,1]	Φ_c	$S(\boldsymbol{x_6^P;\theta_6^P})$	10^{-3}

Table 3.6: Pre- trained Neural network architectures, outputs, transformations and learning rates.

3.2.3 Model Structure

After generating our data for (1) defining the data losses for the variables p_{im},p_{em},ω_t, and W_{egr} and (2) training the pretrained-neural networks with the method we explained in the previous chapter, we are able to build the model that consists of a number of PINNs and I-PINNs and finally solve our problem. We observe that equations 3.4 - 3.6 can be solved separately since the variables do not depend to the variables of the other equations. So, despite entering all equations simultaneously into the model, it is preferable that we first approximate with PINNs the solutions of these three equations and then commit to approximate the solutions of the other differential equations while keeping the dynamics of $\tilde{u}_{egr1}$,$\tilde{u}_{egr2}$ and $\tilde{u}_{vgt}$ unchanged. In this way, we reduce the complexity of the system as fewer gradients of the loss function need to be calculated during the training process. Details about the structure of the FFNNs that are used to approximate the state variables are shown in Table 3.7:

PINNs	Network size	Output	Outputs Transformation	Scaling
$\mathcal{N}_1(t;\theta_1)$	[1,128,128,128,2]	p_{im}, p_{em}	$S_p(p_{im})+0.5$, $S_p(p_{im})$	$\times10^5$
$\mathcal{N}_2(t;\theta_2)$	[1,128,128,1]	x_r	$S_p(x_r)$	$\times0.03$
$\mathcal{N}_3(t;\theta_3)$	[1,128,128,128,1]	T_1	$S_p(T_1)$	$\times300$
$\mathcal{N}_4(t;\theta_4)$	[1,64,64,64,2]	u_{egr1}, u_{egr2}	$S(\tilde{u}_{egr1}, \tilde{u}_{egr2})$	$\times100$
$\mathcal{N}_5(t;\theta_5)$	[1,64,64,1]	ω_t	$S_p(\omega_t)$	$\times5\times10^3$
$\mathcal{N}_6(t;\theta_6)$	[1,65,64,1]	$\tilde{u}_{vgt}$	$S(\tilde{u}_{vgt})$	$\times100$

Table 3.7: Neural network architectures, outputs, transformations and scaling factors.

Each one of the PINNs takes in as input the rescaled time vector $\tilde{\boldsymbol{t}} = [-1, -0.9933, \ldots, 1]$ which derives from the transformation of the time vector $\boldsymbol{t} = [0, 0.2, \ldots, 60]$ according to equation 2.9. The input shape of input vector $\tilde{\boldsymbol{t}}$ is 1×301, exactly the same with time vector $\boldsymbol{t}$. The reason for rescaling the input vector was discussed at the PINNs' example of the previous section. The criteria for constructing each neural network with the appropriate number of layers an neurons has been extensively and repeatedly explained already in our thesis.

The activation function we in every hidden layer of the PINNs is tanh while we apply an appropriate output tranformation in the last layer of each PINN. To be precised, we use two types of activations in the last layers, Softplus and Sigmoid activation functions. The Sigmoid function is used for predicting variables that are strictly positive but are bounded above. Such variables are $\tilde{u}_{egr1}$, $\tilde{u}_{egr2}$ and $\tilde{u}_{vgt}$, which denote coverage percentage of valve positions and their values range between 0 and 100. Softplus activation fuction is applied in the last hidden layers of variables that are considered to be positive physical quantities with no upper bounds. Scaling factors are applied to the outputs of the PINNs to transform the normalized predictions into variables with the appropriate physical units and realistic magnitudes.

To ensure that the neural network's output after the Softplus activation attains a desired minimum value and possesses the correct physical units, we apply two key transformations. First, since the Softplus function

$$y = \ln(1+e^z)$$

is always positive but can approach zero, we add a constant offset α to guarantee a minimum output value. Second, we multiply the resulting output by a scalar factor s to scale it into the appropriate physical units

of the modeled variable. Mathematically, the final output is given by

$$y_{\text{phys}} = s \times (\ln(1 + e^z) + \alpha),$$

which ensures the network output satisfies

$$y_{\text{phys}} \geq s\alpha$$

and accurately represents the physical quantity of interest. In fact, the values for scaling factor s and constant offset α are chosen appropriately in order the value $s\alpha$ to be equal to or just lower from the minimum value of the physical variable y_{phys} we posess from the the training data.

For the variables that we desire final outputs to be within a range $[0, b]$ we apply the Sigmoid function to the output of each NN. Since the Sigmoid function naturally produces outputs in $[0, 1]$, it can be directly scaled to match the desired range $[0, b]$. Specifically, if

$$y = \sigma(z) = \frac{1}{1 + e^{-z}},$$

then the physical output is given by

$$y_{\text{phys}} = b \times y = b \times \sigma(z).$$

where b is actually the scaling factor for each case.

This guarantees that the network prediction satisfies $y_{\text{phys}} \in [0, b]$, ensuring that the solution remains physically valid and within the required bounds.

The loss functions are calculated according to (2.6) and (2.8). The data loss terms are only used to approximate variables where data are given. In other words, we take into consideration those terms only for the variables p_{im},p_{em},ω_t, and W_{egr}. For the prediction of the other variables, we consider only initial condition and equation loss terms during the training process. We should also highlight that all the loss terms should be multiplied by appropriate weights. While there are many approaches to adjusting the weight losses in order to increase the model's performance, in this thesis we are focusing on adjusting the weights manually and keeping them constant during training.

For the part of model that no prediction of the variables $\tilde{u}_{egr1}$,$\tilde{u}_{egr2}$ and $\tilde{u}_{vgt}$ is required, as they will be approximated separately, the total loss term for the model that will predict the solutions of the coupled system will be given by:

$$\begin{aligned}
\mathcal{L}_{total,coupled} = &(\lambda_{p_{im}} \times \mathcal{L}_{Eq,p_{im}} + \lambda_{IC,p_{im}} \times \mathcal{L}_{IC,p_{im}} + \lambda_{Data,p_{im}} \times \mathcal{L}_{Data,p_{im}}) \\
&+ (\lambda_{p_{em}} \times \mathcal{L}_{Eq,p_{em}} + \lambda_{IC,p_{em}} \times \mathcal{L}_{IC,p_{em}} + \lambda_{Data,p_{em}} \times \mathcal{L}_{Data,p_{em}}) \\
&+ \left(\lambda_{W_{egr}} \times \mathcal{L}_{Eq,W_{egr}} + \lambda_{IC,W_{egr}} \times \mathcal{L}_{IC,W_{egr}} + \lambda_{Data,W_{egr}} \times \mathcal{L}_{Data,W_{egr}}\right) \\
&+ (\lambda_{IC,\omega_t} \times \mathcal{L}_{IC,\omega_t} + \lambda_{Eq,\omega_t} \times \mathcal{L}_{Eq,\omega_t} + \lambda_{Data,\omega_t} \times \mathcal{L}_{Data,\omega_t}) \\
&+ (\lambda_{IC,T_1} \times \mathcal{L}_{IC,T_1} + \lambda_{Eq,T_1} \times \mathcal{L}_{Eq,T_1}) \\
&+ (\lambda_{IC,x_r} \times \mathcal{L}_{IC,x_r} + \lambda_{Eq,x_r} \times \mathcal{L}_{Eq,x_r})
\end{aligned} \tag{3.14}$$

Also, the loss terms for training the networks $N_4(t;\theta_4)$ and $N_6(t;\theta_6)$ are respectively:

$$\mathcal{L}_{N_4(t;\theta_4)} = \lambda_{IC,\tilde{u}_{egr1}}\mathcal{L}_{IC,\tilde{u}_{egr1}} + \lambda_{Eq,\tilde{u}_{egr1}}\mathcal{L}_{Eq,\tilde{u}_{egr1}} + \lambda_{IC,\tilde{u}_{egr2}}\mathcal{L}_{IC,\tilde{u}_{egr2}} + \lambda_{Eq,\tilde{u}_{egr2}}\mathcal{L}_{Eq,\tilde{u}_{egr2}} \tag{3.15}$$

$$\mathcal{L}_{N_6(t;\theta_6)} = \lambda_{IC,\tilde{u}_{vgt}}\mathcal{L}_{IC,\tilde{u}_{vgt}} + \lambda_{Eq,\tilde{u}_{vgt}}\mathcal{L}_{Eq,\tilde{u}_{vgt}} \tag{3.16}$$

The challenge here is to adjust appropriately the weights λ of each loss term. Since the state variables have different physical units and magnitudes, the corresponding residuals of the ODEs — and thus the individual loss terms — naturally have different scales. There are many strategies that we could apply for ensuring stability during the training process and manage to treat every state variable equally such that every single loss term that corresponds to each state variable is progressively minimized. To ensure balanced training and prevent any single loss term from dominating the optimization process, we introduced scaling weights for each loss term. These weights remain constant during the training process and were chosen such that

the initial contributions of each term to the total loss are approximately equal, with the goal of normalizing each loss term to be of order 1 during the first training epoch. To state it clear, each weight $\lambda_{l,IC}$, $\lambda_{l,Eq}$ and $\lambda_{k,Data}$ in equation 2.7 that multiplty initial condition, equation and data loss terms respectively are calculated as :

$$\lambda_{l,IC} = \frac{1}{\mathcal{L}^{(0)}_{l,IC}}, \quad \lambda_{l,Eq} = \frac{1}{\mathcal{L}^{(0)}_{l,Eq}}, \quad \lambda_{k,Data} = \frac{1}{\mathcal{L}^{(0)}_{k,IC}} \tag{3.17}$$

where $\mathcal{L}^{(0)}_{l,IC}$, $\mathcal{L}^{(0)}_{l,Eq}$, $\mathcal{L}^{(0)}_{k,IC}$ are the initial condition, equation and data loss values respectively , described by equations 2.2- 2.4, at the first epoch of the training process.

The PINNs $\mathcal{N}_1(t;\theta_1)$ and $\mathcal{N}_5(t;\theta_5)$ that are responsible for predicting the dynamics of p_{im} , p_{em} and ω_t seem to perform well on clean data while the dynamics of the variables x_r and T_1 seem to differ significantly from the true values. In other words, the training of the PINNs $\mathcal{N}_2(t;\theta_2)$, $\mathcal{N}_3(t;\theta_3)$ has failed so far. This can also be concluded by the results in Chapter 4.

Summarizing, until now we have managed to make some PINNs that predict satisfactorily the dynamics of all the state variables except temperature T_1 and residual gas fraction x_r. The strategy we are going to implement in this stage , for finally manage to train our models and approximate the dynamics of the two forementioned values is continue the training of the model by freezing the trainable parameters of the networks $\mathcal{N}_1(t;\theta_1)$ and $\mathcal{N}_5(t;\theta_5)$ in order to increase the ability of our model to minimize the total loss function by decreasing the models complexity. More analytically, from an epoch and after, the only loss terms that will remain "active" and our optimizer will commit to minimize is the equation losses embedded with the physics of variables x_r and T_1. That means that from a specific stage of the training process and after, we the initial model with total loss that is described by equation 3.14 becomes equivalent to a model with the following loss function :

$$\mathcal{L}_{T_1,x_r} = (\lambda_{IC,T_1} \times \mathcal{L}_{IC,T_1} + \lambda_{Eq,T_1} \times \mathcal{L}_{Eq,T_1}) + (\lambda_{IC,x_r} \times \mathcal{L}_{IC,x_r} + \lambda_{Eq,x_r} \times \mathcal{L}_{Eq,x_r}) \tag{3.18}$$

The weights λ_{IC,T_1}, λ_{Eq,T_1}, λ_{IC,x_r}, λ_{Eq,x_r} are calculated with the method we have already discussed and described by equation 3.17. Judging by the results, we can see that , at east for clean data , our method has increased the performance of neural networks $\mathcal{N}_2(t;\theta_2)$ and $\mathcal{N}_3(t;\theta_3)$. The methodology we have discussed so far is also applied in cases where we have noisy and sparse data as well.

Another thing that is important to emphasize that the minimization of each individual PINN loss term does not necessarily guarantee that the Physics-Informed Neural Network has successfully learned the underlying physics governing the variable it aims to predict. This is because the total loss can be reduced primarily through the minimization of the initial conditions (IC) or data loss components, without adequately satisfying the physical constraints encoded by the governing differential equations or boundary conditions. Therefore, careful design of the loss function weighting and thorough validation are essential to ensure that the model truly captures the intended physical behavior rather than merely fitting observed data.

Finally, to draw conclusions about the performance of the proposed model, we present plots comparing the predictions of each PINN with the corresponding true dynamics of each variable. The reference dynamics are obtained from (1) the simulation framework described in [15] and (2) the numerical results discussed previously. To quantitatively assess the agreement between the predicted and true values, we employ the mean squared error (MSE) and the coefficient of determination (R^2) as evaluation metrics on the testing dataset. Moreover, we investigate the model's robustness and accuracy under scenarios with noisy and sparse data, aiming to identify the conditions and bounds within which the model remains reliable.

Chapter 4

Evaluation and Results

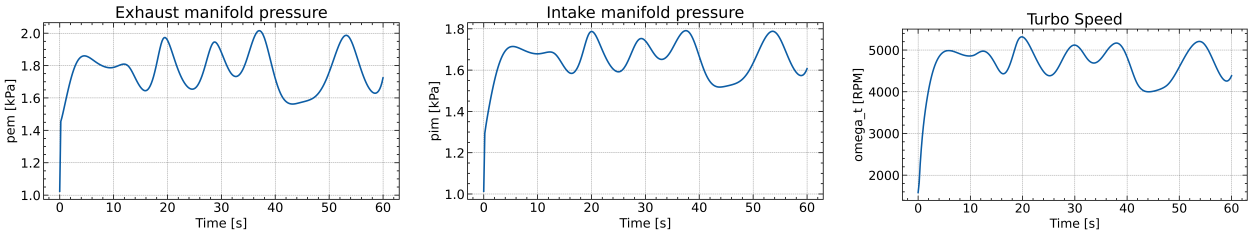
In this chapter we present our results for the diesel engine model we constructed in three different data cases: clean data , noisy data and sparse data. From the Simulink simulation we used from [15] we managed to gain 2200 samples in total for physical quantities used in this work in the time domain. For the training of the PINNs we managed to create a training and a testing data set that we utilized during the training process while the performance of the model was tested on an evaluation dataset (unseen data). For the training of the surrogates, due to limited data possession for training only training and testing data sets where constructed.

At first, for all three cases we are presenting the data generated each time by the Simulink model as well as the numerical calculations as we discussed in Chapter 3.Right after, we illustrate the performance of our pre-trained NNs that are functioning as surrogates of the analytical functions used in the embedding physics of the system during the training process of the diesel engine model. Although we plot only the performance of the pre-trained NNs in cases with clean data, 4% noisy data and sparsity where $\Delta_t = 2s$ we provide metrics to compare the predictions of these NNs with the actual true values we have successfully generated already for every case. More specifically we use MSE and R^2 metrics as together they provide a more complete evaluation of a each regression model's performance. While MSE shows the average magnitude of prediction errors in the original units of the target variable, R^2 indicates how well the model explains the variability in the data compared to a simple baseline. Combining them helps to better interpret how accurate and how explanatory the model is.

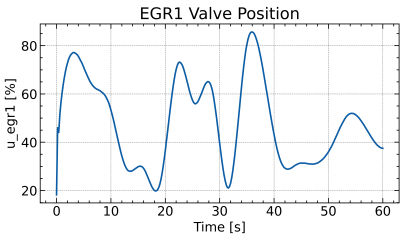
Finally , we plot the predictions for the state dynamics as well as the predicted values for the unknown parameters and we compare with the true ones for the evaluation data set. We also present the progression of the loss function for each one of the PINNs in each training phase of the model (the model was trained in 3 stages as we discussed in Chapter 3). While state variables differ significantly in scale we use L_2 relative error metric and also R^2 in this case as well. Using both ensures a fair comparison across models, highlighting both accuracy and explanatory power regardless of scale differences.

4.1 Model perfromance with clean data

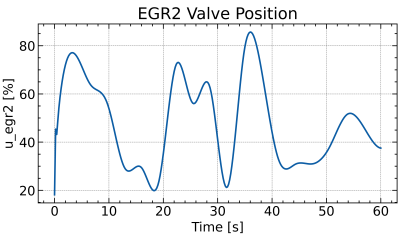
Generated Data



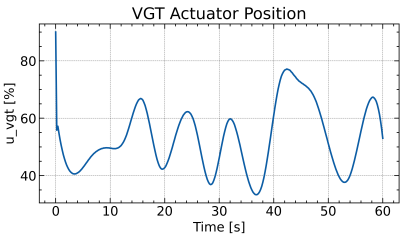
(a) $p_{em}(t)$ (b) $p_{im}(t)$ (c) ω_t



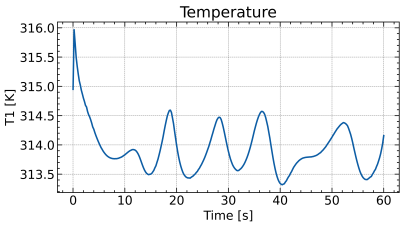
(d) u_{egr1}



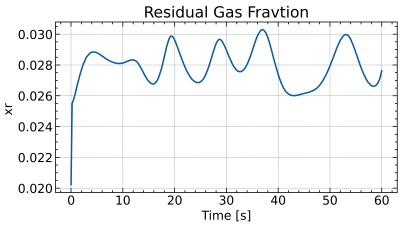
(e) u_{egr2}



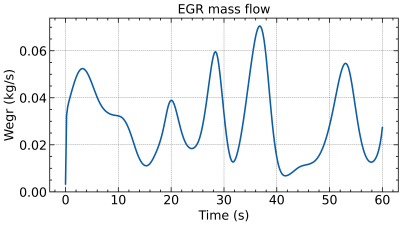
(f) $u_{vgt}(t)$



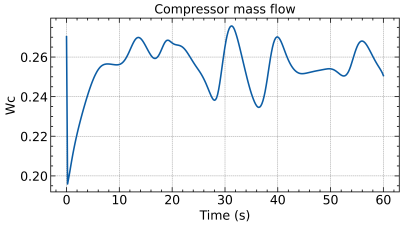
(g) $T_1(t)$



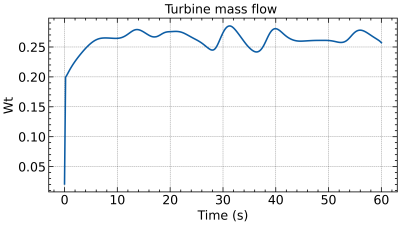
(h) $x_r(t)$



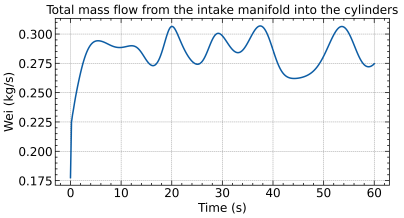
(i) $W_{egr}(t)$



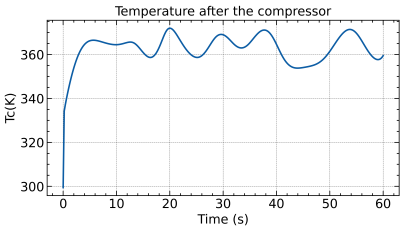
(j) $W_c(t)$



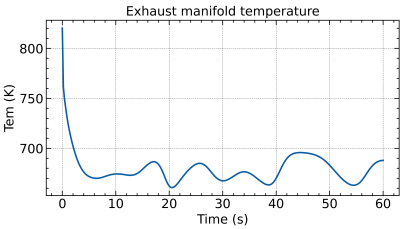
(k) $W_t(t)$



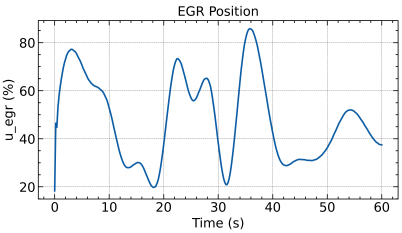
(l) $W_{ei}(t)$



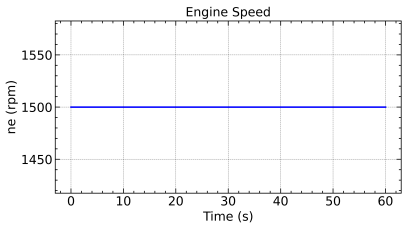
(m) $T_c(t)$



(n) $T_{em}(t)$



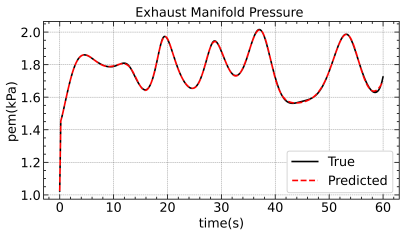
(o) $\tilde{u}_{egr}(t)$



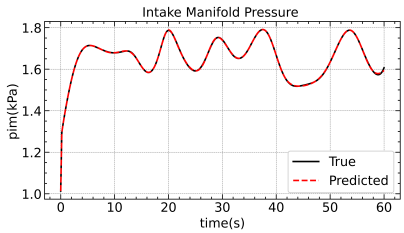
(p) $n_e(t)$

Figure 4.1: Clean data for state variables dynamics as well as dynamics for variables required for training of empirical formulae. Data was generated by the Simulink model in [15] while RK45 method was used for numerically solving the ODEs responsible for predicting the dynamics of $\tilde{u}_{egr1}$, $\tilde{u}_{egr2}$, $\tilde{u}_{vgt}$, ω_t variables. From the whole data generation process we managed to obtain 2200 samples for the time domain 0 to 60s

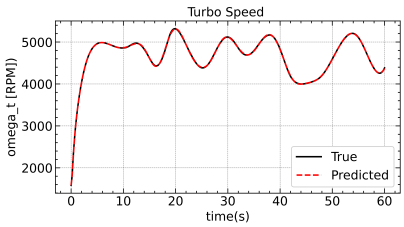
State Variables Dynamics



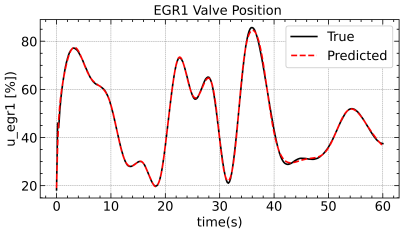
(a) $p_{em}(t)$



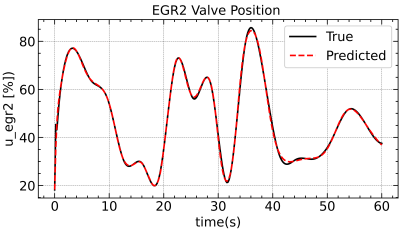
(b) $p_{im}(t)$



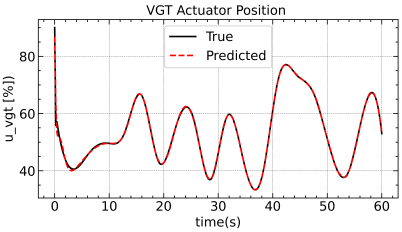
(c) ω_t



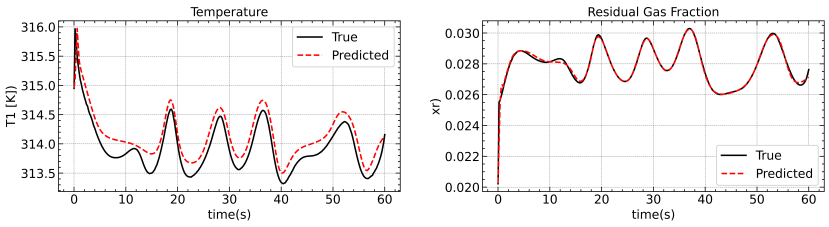
(d) u_{egr1}



(e) u_{egr2}



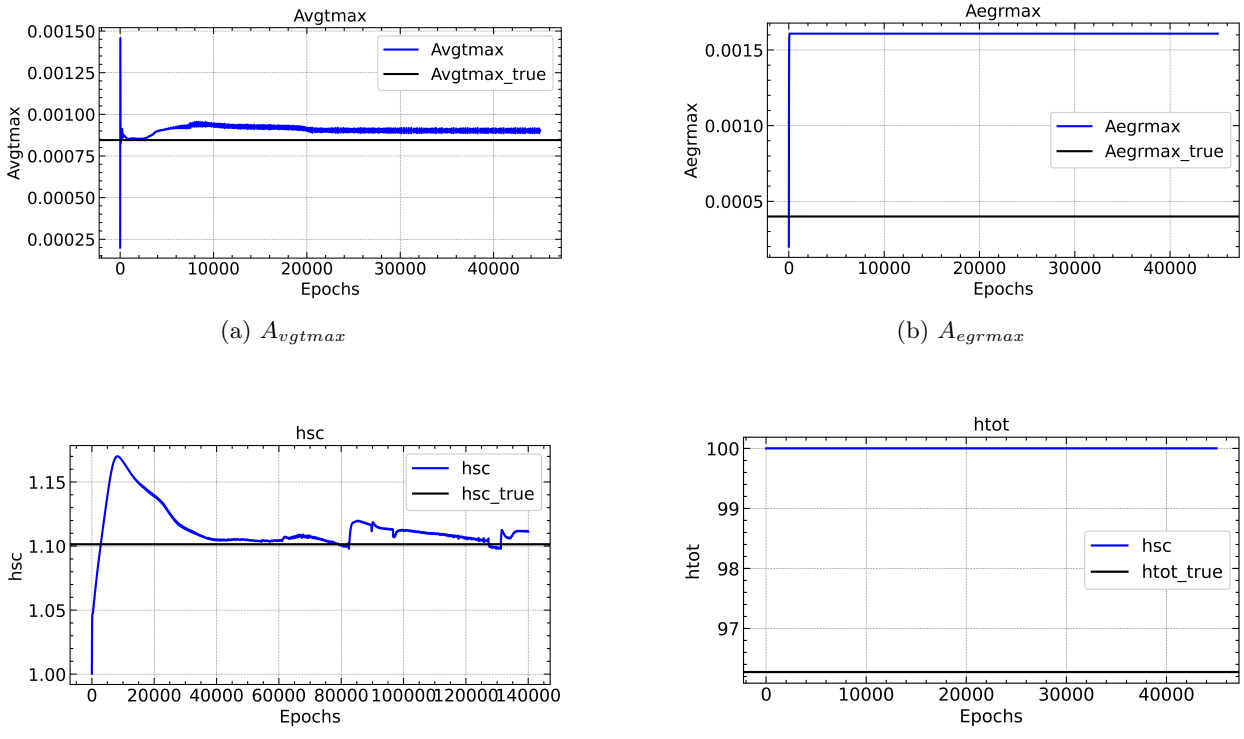
(f) $u_{vgt}(t)$



(g) $T_1(t)$ (h) $x_r(t)$

Figure 4.2: Performance of the model in predicting the dynamics of the state variables on a clean evaluation dataset. The training dataset consisted of 301 samples per variable, corresponding to target values at each time step of $\Delta t = 0.2\,\text{s}$ over a total time span of $T = 1\,\text{min}$. The testing (validation) dataset contained 60 samples with randomly distributed sparsity across the entire time domain. The figure illustrates the model's performance on unseen evaluation data, which was not included in the training process. Overall, the model accurately captures the dynamics of all state variables. However, compared to the other variables, it shows relatively greater difficulty in predicting the dynamics of the temperature variable T_1.

Unknown Parameters



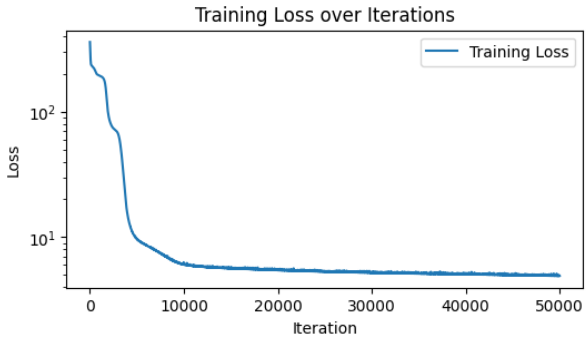
(a) A_{vgtmax} (b) A_{egrmax}

(c) h_{sc} (d) h_{tot}

Figure 4.3: Progressive predictions for unknown parameters' values during the training process with clean data. It is obvious that only A_{vgtmax} and h_{sc} parameters are approximated during the training process. While the trainable parameter A_{egrmax} is updated from the initial value given, but does not approximate its actual true value, h_{sc} is not updated at all. The values for parameters that are satisfactorily approximated are $A_{vgtmax} = 9 \times 10^{-4}$ and $h_{sc} = 1.112$.

PINNs Losses

(a) Training Loss for PINN $\mathcal{N}_6(t; x_6)$—responsible for predicting the dynamics of u_{vgt}—with clean data.



(b) Training Loss for PINN $\mathcal{N}_4(t; x_4)$—responsible for predicting the dynamics of u_{egr1}, u_{egr2}—with clean data.

Figure 4.4: Training Losses for the PINNs, responsible for predicting VGT actuator and EGR valve positions.

Figure 4.5: Loss functions of PINNS for predicting the dynamics of the variables p_{im}, p_{em}, ω_t, T_1, x_r. PINNS are trained with clean data. The loss of the PINN1 responisble for predicting the dynamics of p_{im} and p_{em} contribute significantly in the total loss comparing to the other PINN losses. The variables T_1 and x_r are approximated exclusively using the embedded physics in the model, as no data loss terms are included in the training process for the PINNs responsible for these variables. However, he physics loss terms for these variables do not seem to downgrade substantially in this training stage of the model which means that the downgrade of the loss terms for these two variables occurs from the minimization of initial condition loss. It is observed that all loss components—equation loss, initial condition loss, and data loss (where applicable)—decrease during training for the rest of the variables, indicating that the model does not rely solely on data but also effectively leverages the underlying physical laws of the system.

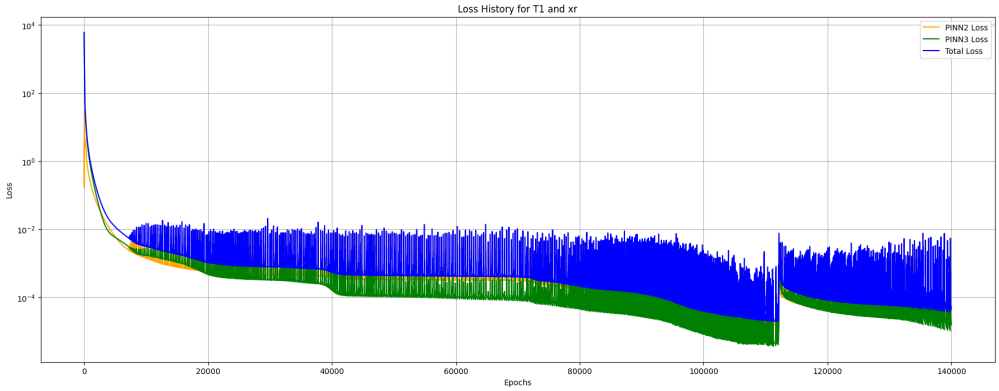
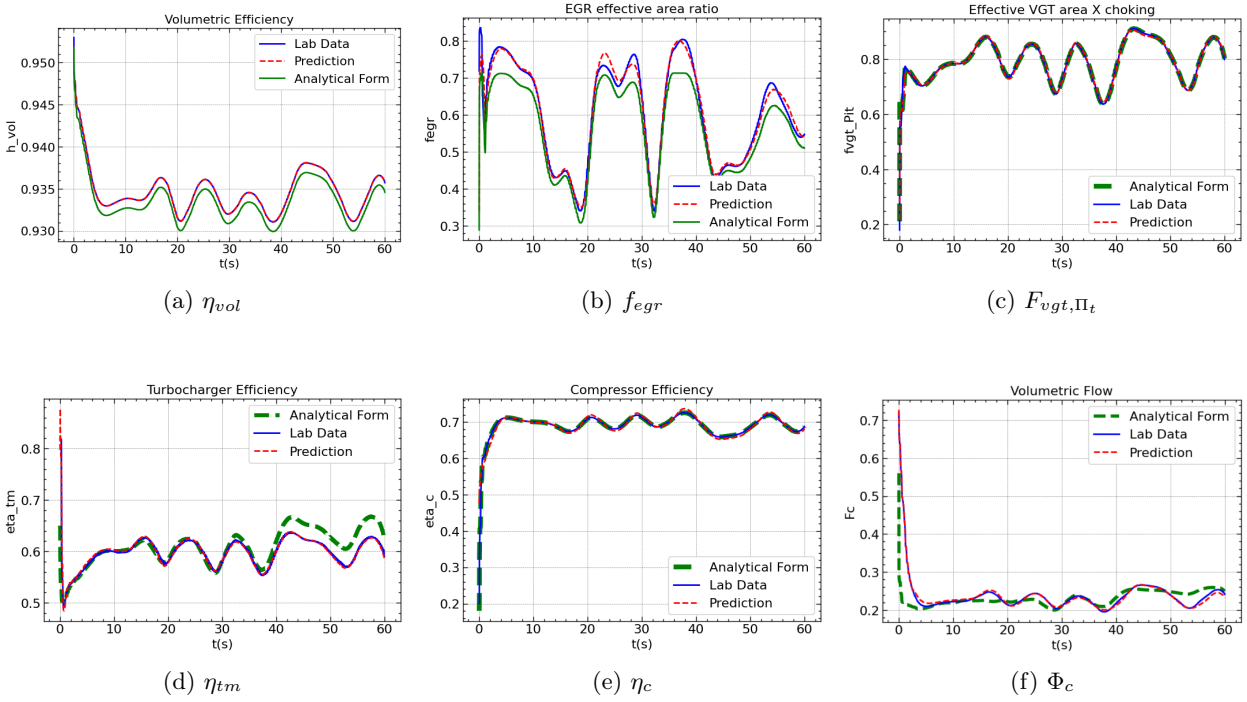


Figure 4.6: Loss functions of the PINNs used to predict the dynamics of the variables T_1 and x_r, after freezing the trainable parameters of the other PINNs that were trained using clean data for p_{im}, p_{em}, ω_t, and W_{egr}.

Empirical formulae



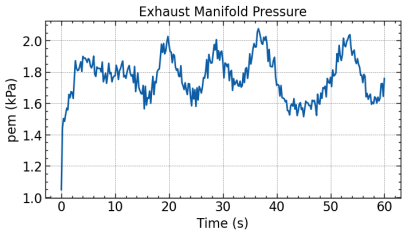
(a) η_{vol} (b) f_{egr} (c) F_{vgt,Π_t}

(d) η_{tm} (e) η_c (f) Φ_c

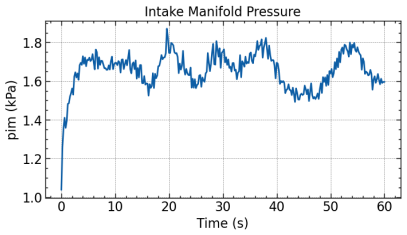
Figure 4.7: Empirical formulae on clean data. **Blue line:** Empirical formulae with clean data from simulink model. **Red line:**Pre-trained neural network prediction for empirical formulae. **Green line:** Variables are calculated analytically with functions described in Appendix B. The surrogates were trained with 1800 training samples while 400 samples we used for the testing(validation) dataset during training. The analytical forms of the surrogates helps us confirm that the surrogates qualitatively correspond to the engine's physical analysis. However, they do not contribute to the surrogates' training.

4.1.1 Performance in noisy conditions

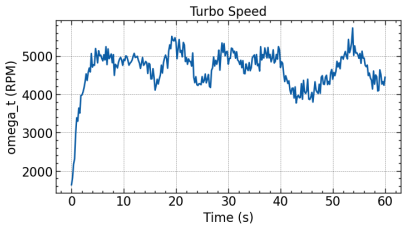
Generated Data



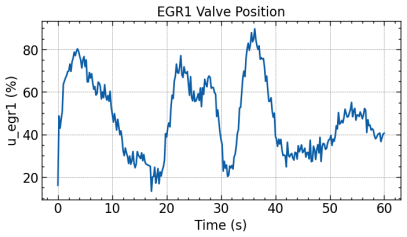
(a) $p_{em}(t)$



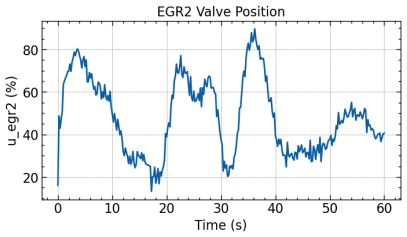
(b) $p_{im}(t)$



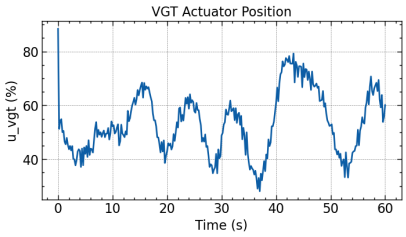
(c) ω_t



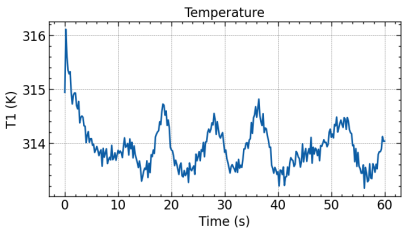
(d) u_{egr1}



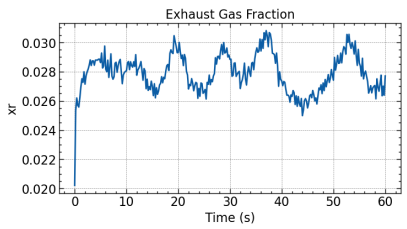
(e) u_{egr2}



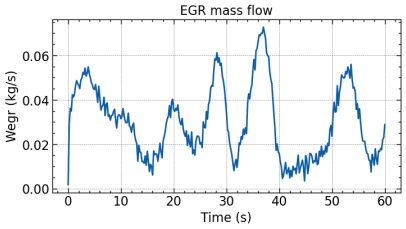
(f) $u_{vgt}(t)$



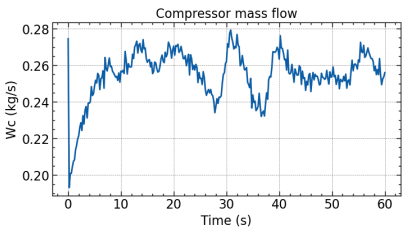
(g) $T_1(t)$



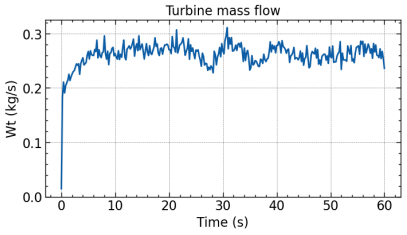
(h) $x_r(t)$



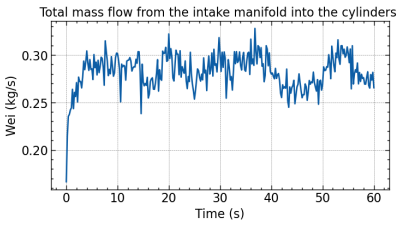
(i) $W_{egr}(t)$



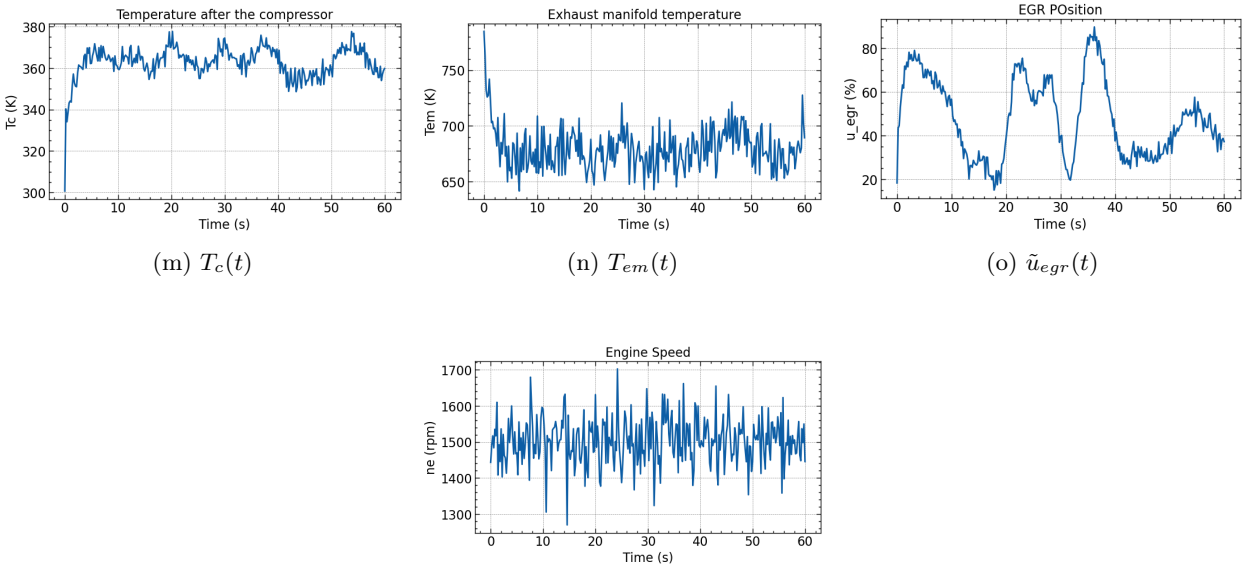
(j) $W_c(t)$



(k) $W_t(t)$



(l) $W_{ei}(t)$



(m) $T_c(t)$ (n) $T_{em}(t)$ (o) $\tilde{u}_{egr}(t)$

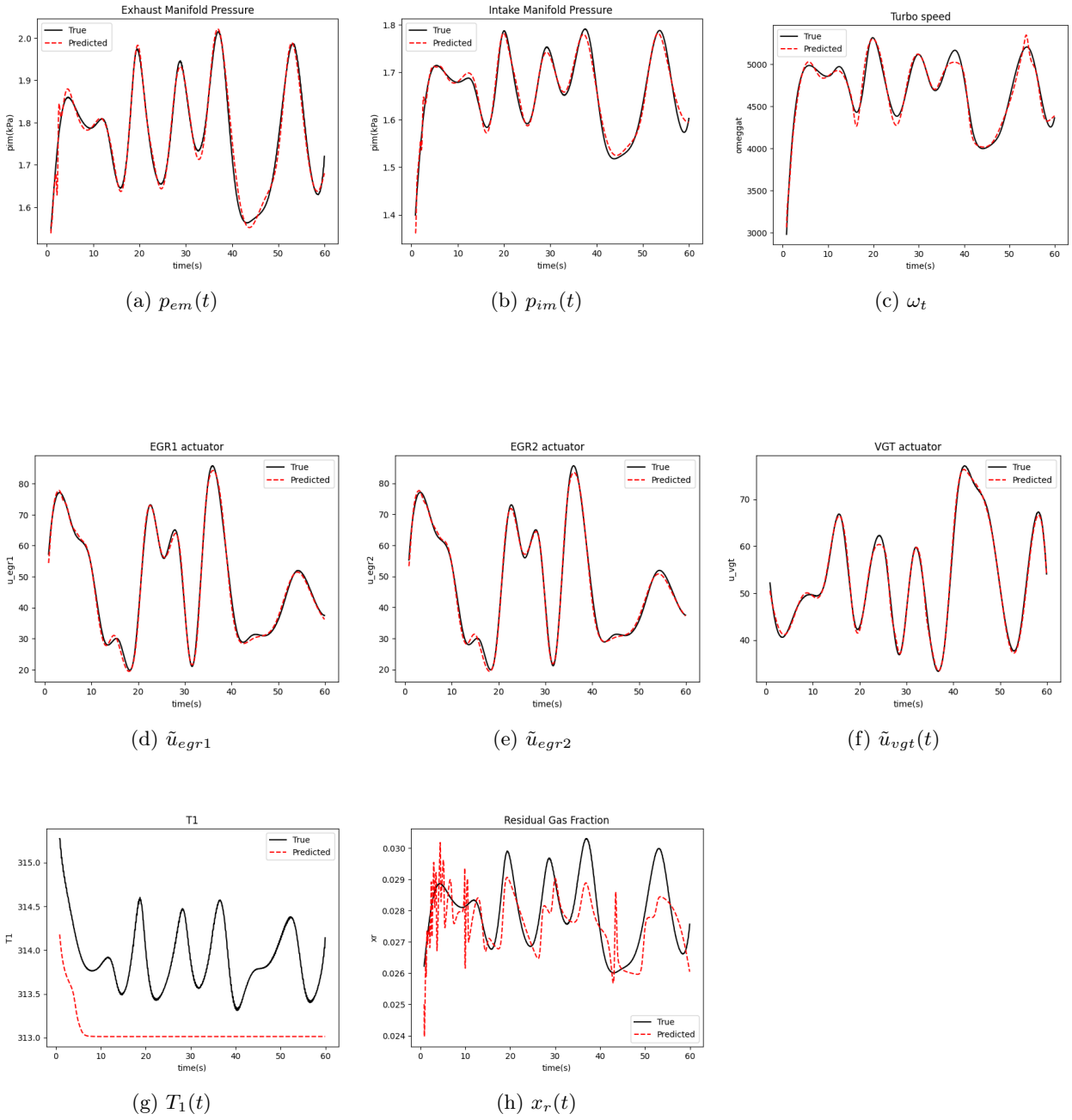
(p) $n_e(t)$

Figure 4.8: Noisy data for the dynamics of state variables, as well as for the variables used in the training of empirical formulae. The noisy data were generated by adding 4% white noise to the corresponding clean datasets of the variables.

State Variables Dynamics

Dataset	Noise level	p_{im}	p_{em}	ω_t	u_{egr1}	u_{egr2}	u_{vgt}	T_1	x_r
L2 Error									
Training	**0%**	4.04e-03	6.54e-03	1.56e-03	1.50e-02	1.65e-02	1.14e-02	4.43e-04	8.30e-03
Testing	0%	3.91e-03	6.93e-03	1.48e-03	1.30e-02	1.43e-02	4.79e-03	3.80e-04	6.23e-03
Evaluation	0%	4.06e-03	6.71e-03	1.43e-03	9.72e-03	1.13e-02	4.65e-03	3.49e-04	6.10e-03
Training	**1%**	2.57e-03	3.23e-03	3.27e-03	1.33e-02	1.38e-02	1.23e-02	4.52e-04	7.80e-03
Testing	1%	2.70e-03	3.63e-03	3.41e-03	1.27e-02	1.31e-02	8.60e-03	3.92e-04	5.63e-03
Evaluation	1%	2.44e-03	3.29e-03	3.15e-03	1.22e-02	1.28e-02	5.88e-03	3.83e-04	4.69e-03
Training	**2%**	3.62e-03	4.53e-03	5.71e-03	1.46e-02	1.57e-02	1.29e-02	1.51e-03	1.58e-02
Testing	2%	3.78e-03	4.92e-03	6.06e-03	1.37e-02	1.53e-02	8.79e-03	1.56e-03	1.53e-02
Evaluation	2%	3.49e-03	4.58e-03	5.61e-03	1.28e-02	1.35e-02	7.23e-03	1.48e-03	1.53e-02
Training	**3%**	7.25e-03	9.28e-03	8.35e-03	1.75e-02	1.80e-02	1.42e-02	2.38e-03	2.38e-02
Testing	3%	7.43e-03	9.04e-03	8.75e-03	1.74e-02	1.76e-02	1.16e-02	2.46e-03	2.47e-02
Evaluation	3%	7.06e-03	9.55e-03	8.16e-03	1.65e-02	1.67e-02	9.00e-03	2.35e-03	2.28e-02
Training	**4%**	6.15e-03	7.68e-03	1.17e-02	2.11e-02	2.11e-02	1.62e-02	2.75e-03	2.93e-02
Testing	4%	6.45e-03	7.73e-03	1.19e-02	2.18e-02	2.11e-02	1.43e-02	2.83e-03	3.33e-02
Evaluation	4%	5.94e-03	7.64e-03	1.15e-02	2.04e-02	2.05e-02	1.18e-02	2.73e-03	2.87e-02
R^2 **Score**									
Training	**0%**	0.995	0.992	1.000	0.998	0.998	0.997	0.891	0.965
Testing	0%	0.994	0.990	1.000	0.999	0.998	0.999	0.916	0.976
Evaluation	0%	0.992	0.990	1.000	0.999	0.999	0.999	0.907	0.976
Training	**1%**	0.998	0.998	0.999	0.998	0.998	0.997	0.886	0.969
Testing	1%	0.998	0.998	0.999	0.999	0.998	0.998	0.913	0.984
Evaluation	1%	0.997	0.998	0.998	0.999	0.999	0.999	0.894	0.986
Training	**2%**	0.996	0.996	0.997	0.998	0.998	0.996	-0.271	0.872
Testing	2%	0.996	0.996	0.997	0.998	0.998	0.998	-0.390	0.878
Evaluation	2%	0.994	0.995	0.995	0.999	0.998	0.999	-0.588	0.850
Training	**3%**	0.984	0.984	0.994	0.997	0.997	0.995	-2.14	0.710
Testing	3%	0.983	0.986	0.993	0.997	0.997	0.997	-2.43	0.684
Evaluation	3%	0.977	0.979	0.990	0.998	0.998	0.998	-2.99	0.668
Training	**4%**	0.988	0.989	0.988	0.996	0.996	0.994	-3.21	0.561
Testing	4%	0.987	0.989	0.987	0.996	0.996	0.995	-3.57	0.427
Evaluation	4%	0.984	0.987	0.980	0.996	0.996	0.997	-4.40	0.473

Table 4.1: Prediction performance of the PINNs during training, testing, and evaluation under varying noise levels. L2 errors and R^2 scores are reported for the state variables. The model successfully predicts the dynamics of the state variables when trained on a clean dataset. However, as noise levels increase, the model progressively struggles to predict the dynamics of the variables x_r and T_1. At the highest noise level, the model fails to predict the dynamics of T_1 and yields poor predictions for x_r. The remaining variables are predicted satisfactorily across all noise levels.



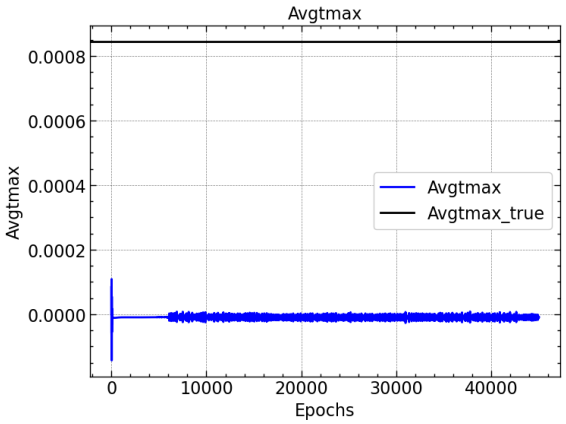
(a) $p_{em}(t)$ (b) $p_{im}(t)$ (c) ω_t

(d) $\tilde{u}_{egr1}$ (e) $\tilde{u}_{egr2}$ (f) $\tilde{u}_{vgt}(t)$

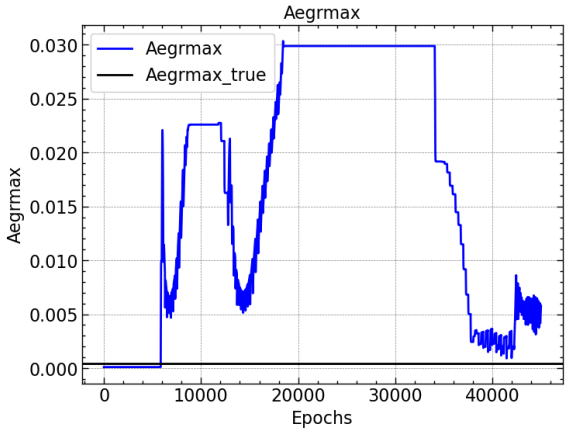
(g) $T_1(t)$ (h) $x_r(t)$

Figure 4.9: Performance of the model showing state variable dynamics over time while 4% white noise was applied on clean data signals for p_{im}, p_{em}, ω_t and W_{egr}. The training process follwed as well as the data sizes for training, testing and evaluation dataset are the same with the ones we discussed in the case were clean data were provided for training. The model seems to poorly on predicting variable x_r while it completely fails to predict variable $T1$. The rest of the variables seem to be satisfactorily predicted even under noisy conditions.

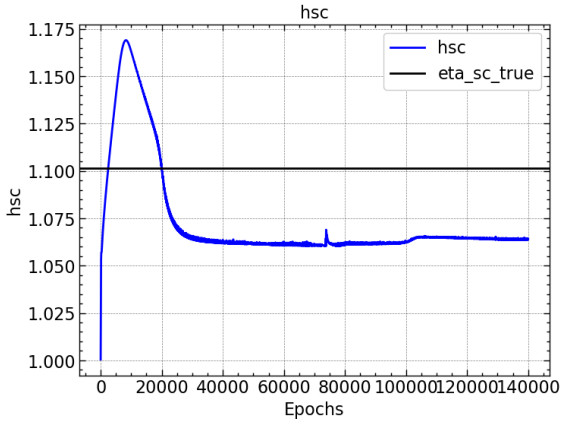
Unknown Parameters



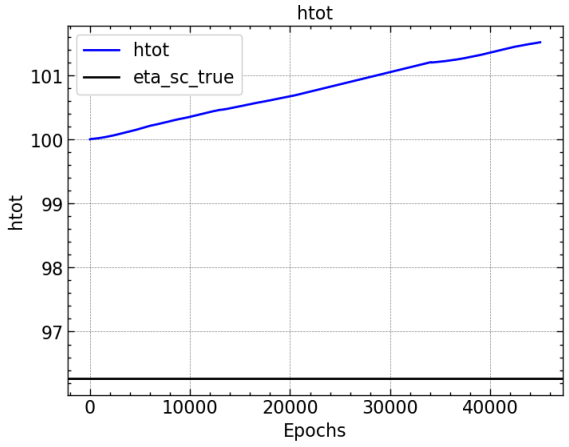
(a) A_{vgtmax}



(b) A_{egrmax}



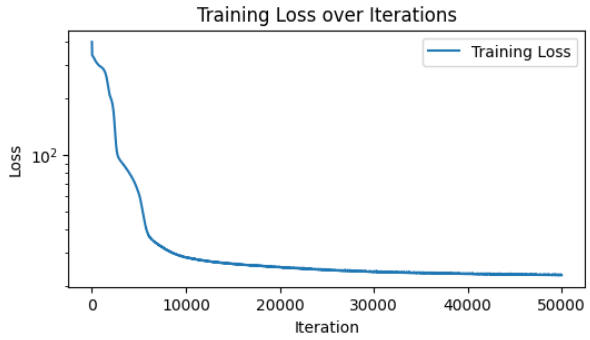
(c) h_{sc}



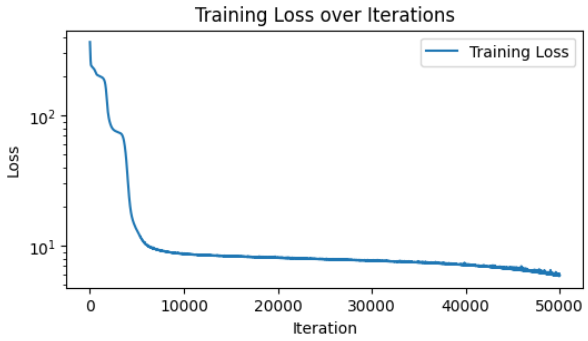
(d) h_{tot}

Figure 4.10: Progressive predictions for unknown parameters' values during the training process with noisy data. Comparing with the case where the model was trained with clean data, in this case we observe that all the trainable parameters are at least updated during the training process. Parameters A_{egrmax} and h_{sc} somehow approximate the actual true values while model completely fails to approximate parameters A_{vgtmax} and h_{tot}. More specifically, for the parameters that the model has managed to approximate we have the following values: $A_{egrmax} = 4.9 \times 10^{-3}$ and $h_{sc} = 1.065$

PINNs Losses



(a) Training Loss for PINN $\mathcal{N}_6(t; x_6)$—responsible for predicting the dynamics of u_{vgt}—with noisy data.



(b) Training Loss for PINN $\mathcal{N}_4(t; x_4)$—responsible for predicting the dynamics of u_{egr1}, u_{egr2}—with noisy data.

Figure 4.11: Training Losses for the PINNs, responsible for predicting VGT actuator and EGR valve positions for maximum noise applied in input data. Since the total loss for both PINNs includes only the equation and initial condition loss terms, minimizing these losses directly corresponds to successfully learning the underlying physics.

Figure 4.12: Loss functions of PINNS for predicting the dynamics of the variables p_{im}, p_{em}, ω_t, T_1, x_r. PINNS are trained with 4% white noise applied on clean data. The loss of the PINN1 responsisble again contributes significantly in the total loss comparing to the other PINN losses. Overall, the PINN losses values are higher than the corresponding ones for the case where the model was trained with clean data. In this case, where noise is applied , again PINN losses responsible for approximating x_r and T_1 arises from a downfall in initial condition rather than equation loss terms . As a result, again further training of these variables, while freezing the training parameters of the remaining PINNs, is important.

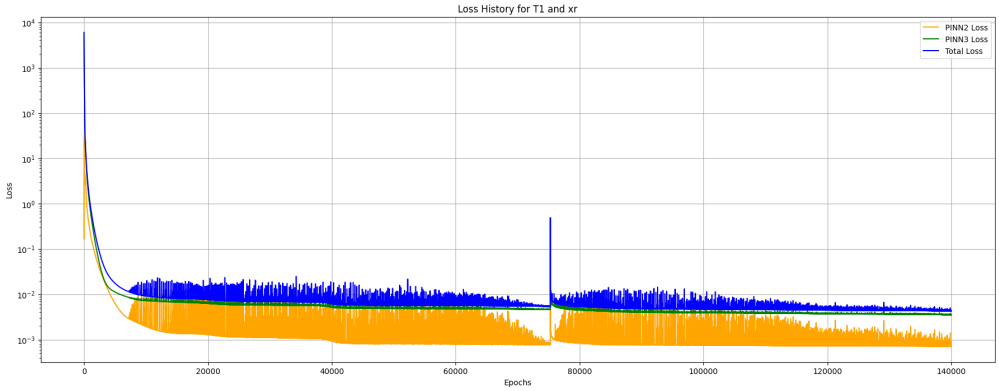
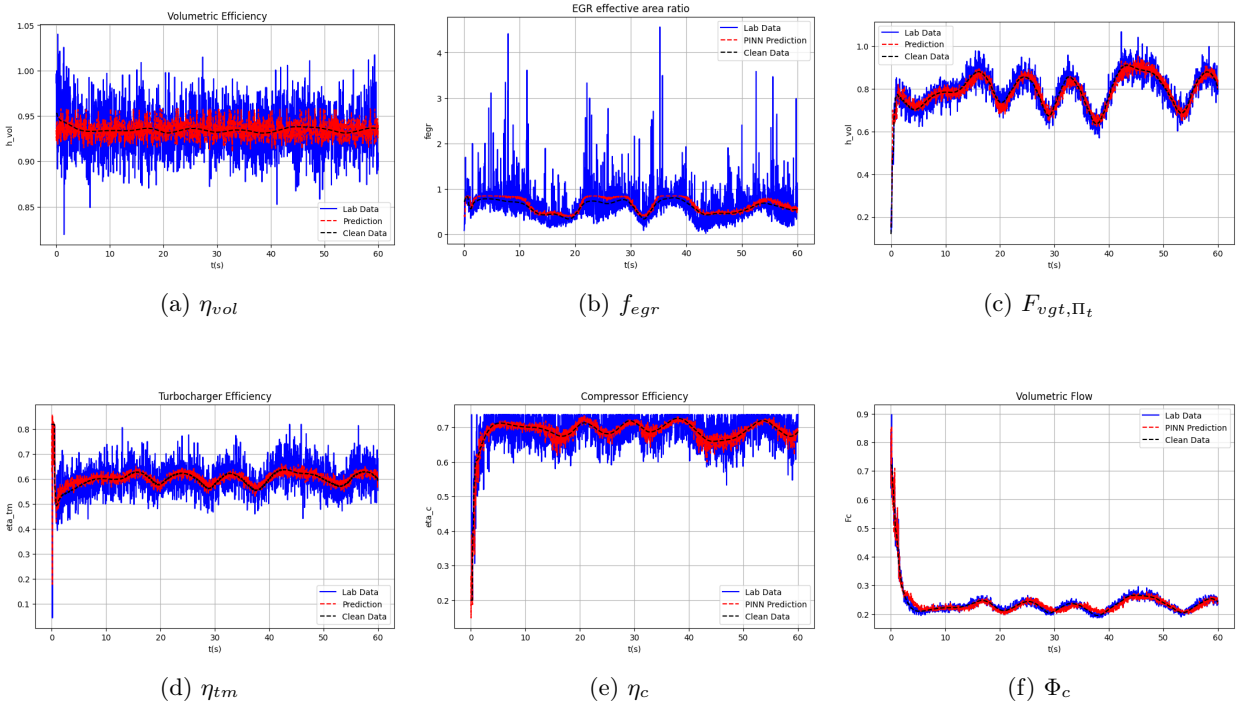


Figure 4.13: Loss functions of the PINNs used to predict the dynamics of the variables T_1 and x_r, after freezing the trainable parameters of the other PINNs that were trained using noisy data for p_{im}, p_{em}, ω_t, and W_{egr}. The loss terms for these variables apperar to be higher comparing with the case with the previous case. While PINN2(predicts the dynamics of x_r) and PINN3(predicts the dynamics of T_1) do not rely on data for training the poor performance possibly occurs from the slight decrease in performance of the PINNs responsible for predicting the dynamics of the rest state variables. In other words , it seems that slight errors in state variable predictions affect significantly these two variables. The downall of T_1 results from the minimization of the initial condition loss term exclusively while the loss terms(initial condition and equation loss terms) for x_r decrease both but not significantly comparing with model's training with clean data.

Empirical formulae



(a) η_{vol} (b) f_{egr} (c) F_{vgt,Π_t}

(d) η_{tm} (e) η_c (f) Φ_c

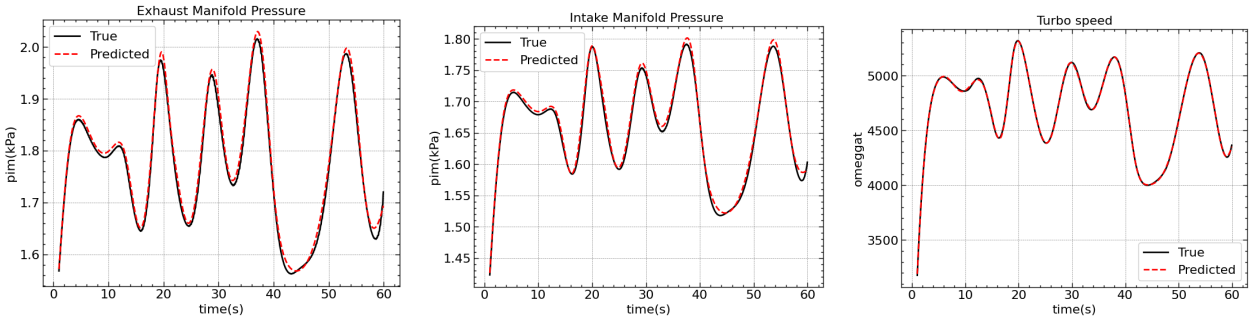
Figure 4.14: Surrogates predictions while trained under 4% white noise for the physical variables included in the empirical formulae. **Blue line:** Empirical formulae with noisy data from simulink model. **Red line:**Pre-trained neural network predictions for empirical formulae. **Black line:** Empirical Formulae for clean data. The training and testing data set sizes and the training strategy as well remain the same with the counterparts used in the case where clean data for empirical were provided

Dataset	Noise Level	η_{vol}	f_{egr}	F_{vgt}	η_c	η_{tm}	F_c
Mean Squared Error (MSE)							
Training	0.0%	1.67e-09	4.64e-04	2.14e-04	2.26e-04	1.85e-05	2.08e-05
Testing	0.0%	8.79e-10	2.52e-04	8.99e-05	4.01e-05	1.07e-05	2.06e-05
Training	1%	2.73e-07	1.15e-03	8.40e-05	1.63e-05	3.31e-05	4.03e-05
Testing	1%	2.77e-07	4.55e-04	8.27e-05	6.42e-06	3.43e-05	3.73e-05
Training	2%	6.75e-06	2.37e-03	2.81e-04	5.70e-05	1.24e-04	7.09e-05
Testing	2%	6.58e-06	1.63e-03	2.42e-04	2.70e-05	1.18e-04	6.43e-05
Training	3%	1.73e-05	3.25e-03	4.45e-04	1.53e-04	3.22e-04	8.53e-05
Testing	3%	1.83e-05	2.49e-03	4.12e-04	6.31e-05	3.09e-04	7.31e-05
Training	4%	4.26e-05	5.65e-03	6.13e-04	2.78e-04	6.07e-04	1.02e-04
Testing	4%	4.52e-05	4.87e-03	5.79e-04	1.56e-04	5.90e-04	8.65e-05
R^2 **Score**							
Training	0.0%	1.00e+00	9.74e-01	9.62e-01	8.21e-01	9.81e-01	9.92e-01
Testing	0.0%	1.00e+00	9.86e-01	9.80e-01	9.11e-01	9.84e-01	9.82e-01
Training	1%	9.69e-01	9.36e-01	9.88e-01	9.93e-01	9.71e-01	9.89e-01
Testing	1%	9.59e-01	9.74e-01	9.83e-01	9.90e-01	9.57e-01	9.77e-01
Training	2%	2.31e-01	8.69e-01	9.58e-01	9.75e-01	8.92e-01	9.80e-01
Testing	2%	3.25e-02	9.05e-01	9.50e-01	9.59e-01	8.51e-01	9.62e-01
Training	3%	-9.70e-01	8.20e-01	9.34e-01	9.33e-01	7.20e-01	9.77e-01
Testing	3%	-1.69e+00	8.55e-01	9.15e-01	9.04e-01	6.10e-01	9.59e-01
Training	4%	-3.85e+00	6.88e-01	9.09e-01	8.78e-01	4.71e-01	9.73e-01
Testing	4%	-5.65e+00	7.17e-01	8.80e-01	7.63e-01	2.55e-01	9.54e-01

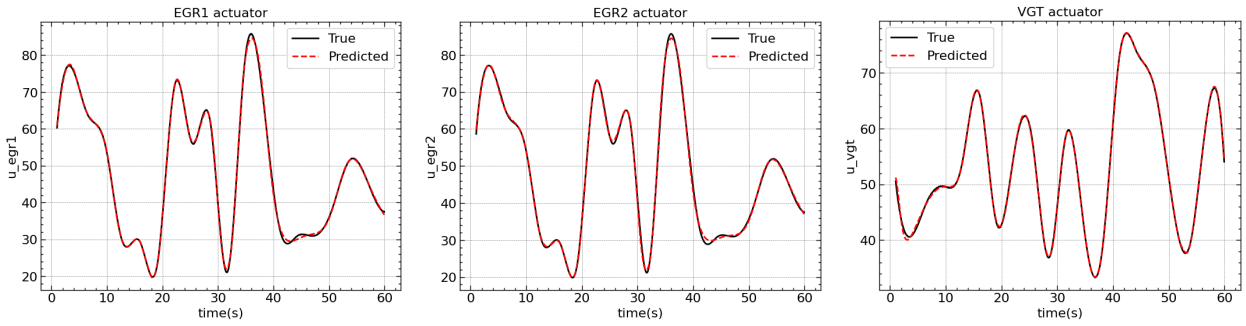
Table 4.2: Prediction performance of the Pre-trained NNs (Surrogates) during training and testing datasets. MSE errors and R^2 scores are reported for the empirical formulae. We compare the predictions of the surrogates with the actual empirical formulae calculations with clean data. Variables η_{tm} and especially $\eta_v ol$ are more sensitive to noise.

4.1.2 Performance in Sparse Data

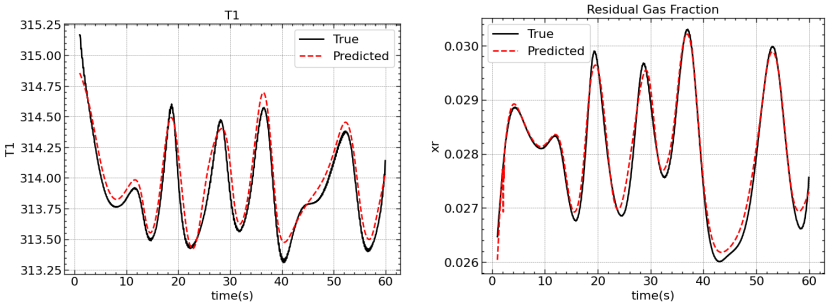
State Variable Dynamics



(a) $p_{em}(t)$ (b) $p_{im}(t)$ (c) ω_t



(d) $\tilde{u}_{egr1}$ (e) $\tilde{u}_{egr2}$ (f) $\tilde{u}_{vgt}(t)$



(g) $T_1(t)$ (h) $x_r(t)$

Figure 4.15: Performance of the model showing state variable dynamics on unseen data. In this case, the time space between training points is equal to $\Delta t = 2s$ while the model is trained with clean data of p_{em}, p_{im}, ω_t and W_{egr}. We observe that even be increasing by ten times the sparsity of the training data, the model manages to predict satisfactorily all state variables dynamics. However, it is clear that initial conditions for x_r and $T1$ are not satisfied.

Unknown parameters

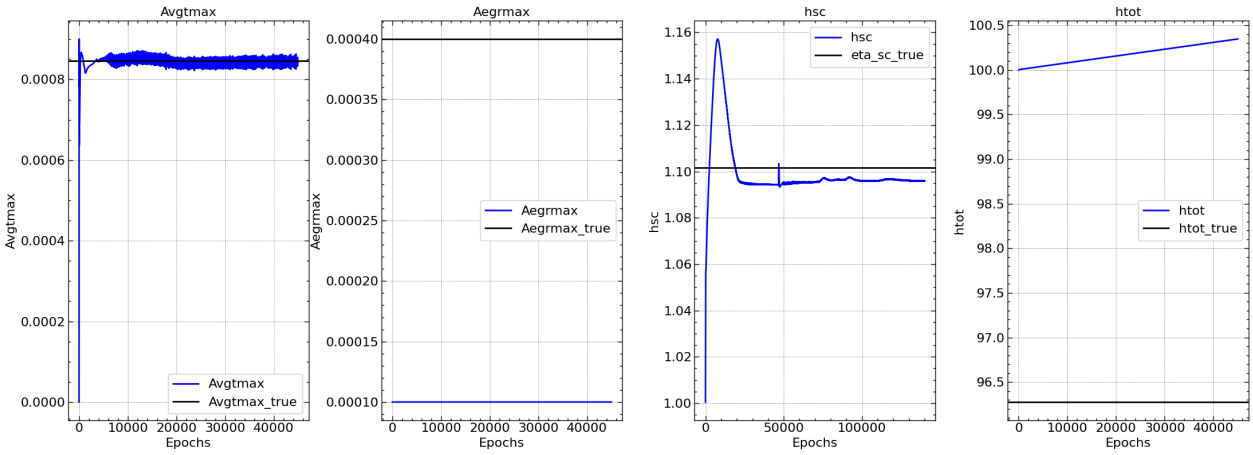
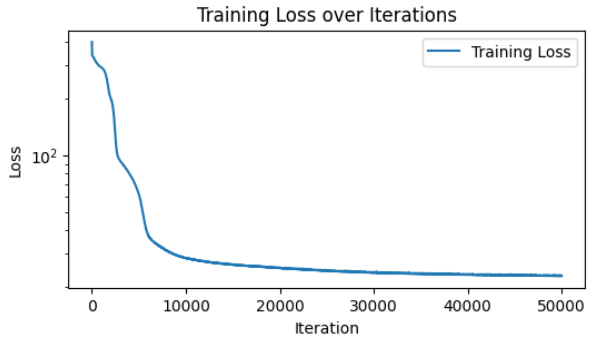
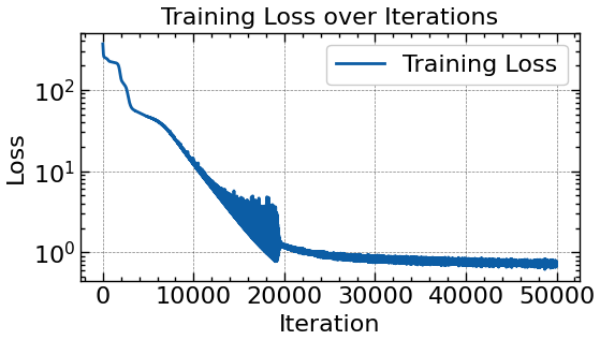


Figure 4.16: Progressive predictions for unknown parameters' values during the training process with sparse data. In this case the model has managed to approximate successfully trainable parameters A_{vgtmax} and h_{sc} while it has failed completely to predict true values for A_{egrmax} and h_{tot}.for the parameters that the model has managed to approximate we have the following values: $A_{vgtmax} = 4.44\ times10^{-4}$, $h_{sc} = 1.096$

Losses



(a) Training Loss for PINN $\mathcal{N}_6(t; x_6)$—responsible for predicting the dynamics of u_{vgt}—with sparse data.



(b) Training Loss for PINN $\mathcal{N}_4(t; x_4)$—responsible for predicting the dynamics of u_{egr1}, u_{egr2}—with spase data.

Figure 4.17: Training Losses for the PINNs, responsible for predicting VGT actuator and EGR valve positions. We observe that both PINN losses downgrade even when model is feeded with sparse clean data and in the same way where more dense data were given.

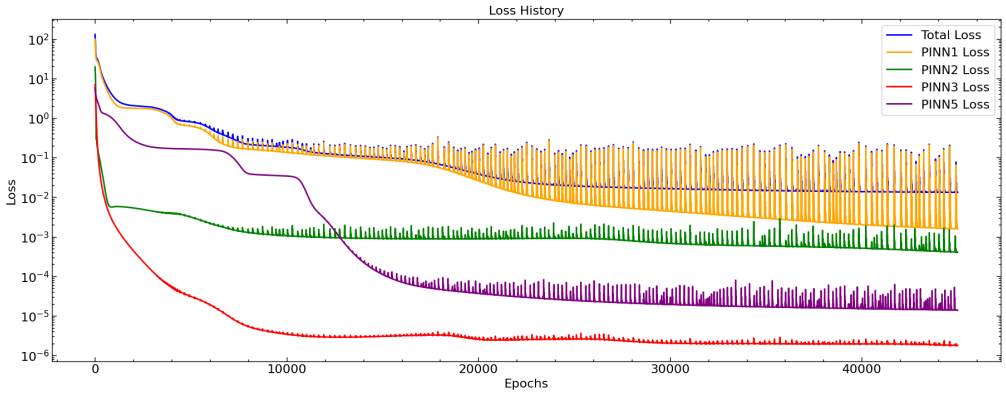


Figure 4.18: Loss functions of PINNS for predicting the dynamics of the variables $p_{im}, p_{em}, \omega_t, T_1, x_r$. PINNS are trained with time step $\Delta t = 2s$. The loss of the PINN1 responisble again contributes significantly in the total loss comparing to the other PINN losses. In this case, where noise is applied , again PINN losses responsible for approximating x_r and T_1 arises from a downfall in initial condition rather than equation loss terms . As a result, again further training of these variables, while freezing the training parameters of the remaining PINNs, is important

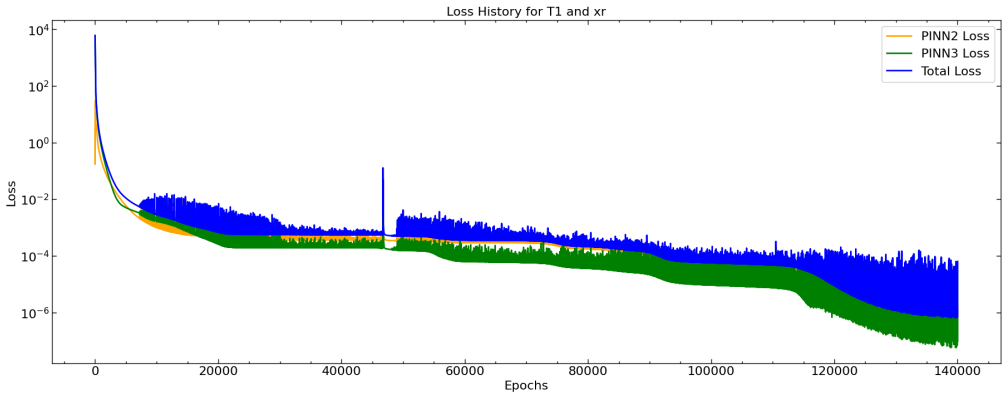


Figure 4.19: Loss functions of the PINNs used to predict the dynamics of the variables T_1 and x_r, after freezing the trainable parameters of the other PINNs that were trained using sparse data for p_{im}, p_{em}, ω_t, and W_{egr}. The loss terms for these variables appear to be even lower comparing with the case with the case where we have more dense and clean data. The downgrade of loss values for both values arises from the satisfaction of the differential equations but the model struggles to approximates initial conditions for both variables, so the corresponding loss terms do not witness dramatic minimization compared to the forementioned case.

Dataset	**Time Step**	p_{im}	p_{em}	ω_t	u_{egr1}	u_{egr2}	u_{vgt}	T_1	x_r
L2 Error									
Training	**0.2s**	2.36e-03	3.73e-03	1.56e-03	1.50e-02	1.65e-02	1.14e-02	7.36e-04	6.72e-03
Testing	random	2.36e-03	3.73e-03	1.56e-03	1.50e-02	1.65e-02	1.14e-02	7.36e-04	6.72e-03
Evaluation	random	2.26e-03	3.77e-03	1.44e-03	9.72e-03	1.13e-02	4.65e-03	7.13e-04	3.16e-03
Training	**0.5s**	1.40e-03	2.08e-03	1.09e-03	1.16e-02	2.55e-02	1.56e-02	3.88e-04	4.17e-03
Testing	random	1.42e-03	2.18e-03	2.49e-03	1.36e-02	1.96e-02	1.36e-02	3.95e-04	1.02e-02
Evaluation	random	1.65e-02	2.25e-02	3.68e-03	3.98e-02	3.69e-02	6.73e-02	5.29e-04	2.18e-02
Training	**1s**	5.96e-03	9.09e-03	9.26e-04	1.29e-02	2.01e-02	2.92e-02	4.44e-04	7.74e-03
Testing	random	2.94e-03	3.30e-03	7.31e-04	1.08e-02	1.60e-02	7.41e-03	3.92e-04	7.85e-03
Evaluation	random	2.40e-02	3.29e-02	5.87e-03	6.14e-02	3.28e-02	8.39e-02	6.74e-04	2.88e-02
Training	**2s**	1.59e-03	1.69e-03	1.09e-03	1.19e-02	1.42e-02	6.28e-02	3.71e-04	3.49e-03
Testing	random	1.62e-03	1.56e-03	5.69e-03	3.60e-02	3.70e-02	2.96e-02	5.54e-04	6.36e-03
Evaluation	random	3.54e-02	4.58e-02	1.62e-02	1.07e-01	9.32e-02	1.25e-01	8.08e-04	3.70e-02
R^2 **Score**									
Training	**0.2s**	0.998	0.997	1.000	0.998	0.998	0.997	0.698	0.977
Testing	random	0.998	0.997	1.000	0.998	0.998	0.997	0.698	0.977
Evaluation	random	0.998	0.997	1.000	0.999	0.999	0.999	0.614	0.994
Training	**0.5s**	1.000	0.999	1.000	0.999	0.995	0.995	0.911	0.993
Testing	random	0.999	0.999	1.000	0.998	0.995	0.996	0.942	0.947
Evaluation	random	0.914	0.900	0.999	0.986	0.988	0.889	0.871	0.722
Training	**1s**	0.993	0.989	1.000	0.999	0.997	0.982	0.881	0.979
Testing	random	0.994	0.997	1.000	0.998	0.996	0.999	0.808	0.949
Evaluation	random	0.831	0.791	0.998	0.966	0.990	0.829	0.815	0.522
Training	**2s**	1.000	1.000	1.000	0.999	0.998	0.925	0.914	0.997
Testing	random	0.999	0.999	0.995	0.990	0.989	0.972	0.675	0.976
Evaluation	random	0.653	0.604	0.992	0.892	0.917	0.600	0.762	0.222

Table 4.3: Prediction performance of the PINNs during training, testing, and evaluation under varying time step (sparsity) levels. L2 errors and R^2 scores are reported for the state variables. The model maintains high performance for most variables under dense sampling (0.2s and 0.5s). As time steps increase (sparser data), prediction accuracy for p_{im}, p_{em}, u_{vgt} x_r and T_1 deteriorates.

Chapter 5

Conclusions

The overall performance of the proposed model can be considered satisfactory based on the results obtained. The model demonstrates robust predictive capabilities in the presence of noisy and sparse data. When provided with clean data, the model successfully captures the dynamics of all state variables; however, only two out of the four unknown parameters are accurately approximated under these conditions. Among the state variables, the dynamics of the variables x_r and T_1 prove to be the most challenging to predict, exhibiting heightened sensitivity especially when data are noisy. For the maximum sparsity applied to the model we have satisfying results while the model struggles only to predict initial conditions for state variables x_r and T_1. These to variables appear to be the most sensitive also comparing to the others in noisy conditions while the model maintains its performance pretty well when trying to predict the rest state variables dynamics in such cases. The performance of the model, when trying to approximate the real values of unknown parameters appears to be problematic.

The failure of our model to predict the unknown parameters successfully even when clean data are provided arises from the implemented strategy of balancing all loss terms as explained in Chapter 3 in order each loss term to be minimized progressively in the same way and scale during the training process. However, the process of balancing these weights becomes particularly challenging in practical settings involving sparse or noisy observations. In such cases, the effective influence of each loss term during training may vary unpredictably across epochs. This instability in weighting can lead to an imbalanced optimization landscape, where certain losses dominate the training dynamics while others become marginalized. Consequently, the accurate inference of unknown parameters—often reliant on residual losses and their sensitivity to the parameter space—may be adversely affected. If the gradients associated with parameter-sensitive losses diminish too early or become overshadowed by high-weighted data terms, the model may converge to suboptimal parameter values. This highlights the critical role of robust and possibly adaptive loss balancing strategies in ensuring reliable parameter estimation, especially under realistic data conditions.

A significant limitation encountered during this work was the lack of access to high-performance computational resources. The entire model was executed on a basic CPU, which resulted in substantially prolonged computational times. Consequently, the frequency and extent of modifications to the Physics-Informed Neural Networks (PINNs) architectures, learning rates, and other hyperparameters were constrained, as each training iteration required a considerable amount of time. Access to more powerful computational infrastructure would likely expedite the training process, enabling a more thorough exploration of alternative modeling strategies and potential improvements.

Regarding the weighting strategy employed in the loss function, our approach involved manually assigning constant weights to each individual loss term, with the goal of ensuring equal contribution from each term to the total loss at the onset of training. A similar approach is reported in [8], albeit with some differences. Specifically, while their method applies constant weights only to selected loss terms, our approach applies manual weighting uniformly across all loss components. Furthermore, the work presented in [8] introduces dynamic weighting schemes, where weights are automatically adjusted during training based on an optimization algorithm. These dynamic weights appear to yield improved model performance compared to the manual weighting strategy adopted in the present study.In addition, the strategy of training the corresponding Physics-Informed Neural Networks (PINNs) in three distinct stages, rather than in a single unified

step, effectively reduced the overall complexity of the model. This staged training approach contributed to improved accuracy, particularly in predicting the dynamics of the variables x_r and T_1, which posed greater challenges. Although this methodology yielded satisfactory final results, it came at the expense of increased computational cost.

In this work, a PINN model has been developed to predict the state dynamics and estimate unknown parameters of a diesel engine, based on specific input functions corresponding to valve positions. The same problem is also solved in the work of [12] but with a purely driven-data model, without taking into account the governing physics of the system of the diesel engine. These kind of models are described as Deep Neural Operators or briefly DeepONets. A fundamental distinction between this approach and operator learning frameworks such as DeepONets lies in their respective mappings: DeepONets learn mappings from input functions to output functions, enabling them to generalize across a wide range of input scenarios by directly approximating the underlying operator. In contrast, PINNs typically map fixed input parameters or initial conditions to corresponding outputs by embedding the governing physical laws within the network architecture. Although DeepONets offer the advantage of generalized solutions capable of predicting system behavior across a wide range of inputs, they typically require large volumes of data for effective training. In contrast, PINNs generally demand significantly less data, as they leverage the known physical relationships to constrain the solution space. This characteristic not only reduces the computational cost associated with data acquisition and training but also renders PINNs particularly valuable in scenarios where data are scarce or expensive to obtain. While PINNs may not always provide the same level of generalization as DeepONets, their ability to efficiently incorporate physical knowledge makes them a powerful tool for modeling complex dynamical systems such as diesel engines.

For future work, the diesel engine model could be further validated and tested under a broader and more diverse set of extended environmental conditions, including variations in ambient temperature, pressure fluctuations, fuel composition, and humidity levels. Such comprehensive testing would provide deeper insights into the model's robustness and reliability, especially in real-world scenarios characterized by noisy, sparse, and uncertain data. This would also help to identify potential limitations and areas for improvement, thereby enhancing the model's applicability for practical engine monitoring and control across different operating environments.

Appendix A

Supplementary Mathematical Theory

Hadamard Product

The Hadamard product, also known as the element-wise product or Schur product, is a binary operation that takes two matrices or vectors of the same dimensions and produces another matrix or vector of the same size, where each element is the product of the corresponding elements from the original inputs.

Formally, let $\mathbf{A} = [a_{ij}]$ and $\mathbf{B} = [b_{ij}]$ be two matrices of the same dimension $m \times n$. The Hadamard product of $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A} \circ \mathbf{B}$ (or sometimes $\mathbf{A} \odot \mathbf{B}$) and is defined as:

$$\mathbf{A} \circ \mathbf{B} = [a_{ij} b_{ij}] \quad \text{for all} \quad i = 1, \ldots, m, \quad j = 1, \ldots, n.$$

Unlike standard matrix multiplication, which involves summations across rows and columns, the Hadamard product operates purely element-wise and does not mix elements from different positions.

Proof of Derivatives for the Hadamard Product and Summation

Let $H = w \odot x$ denote the Hadamard (element-wise) product of vectors $w, x \in \mathbb{R}^N$. That is, for each component $i = 1, \ldots, N$:

$$H_i = w_i x_i.$$

Partial Derivative of the Hadamard Product

We wish to show that:

$$\frac{\partial H}{\partial w} = \operatorname{diag}(x_1, x_2, \ldots, x_N).$$

Proof. By definition,

$$\frac{\partial H_i}{\partial w_j} = \frac{\partial}{\partial w_j}(w_i x_i) = \begin{cases} x_i, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, the Jacobian matrix $\frac{\partial H}{\partial w}$ is diagonal, with the i-th diagonal element equal to x_i:

$$\frac{\partial H}{\partial w} = \begin{bmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & x_N \end{bmatrix} = \operatorname{diag}(x_1, x_2, \ldots, x_N).$$

Partial Derivative of the Summation

Let

$$S = S(H) = \sum_{i=1}^{N} H_i.$$

We wish to show that:

$$\frac{\partial S}{\partial H} = 1^\top, \quad \text{where } 1^\top = [1,\, 1,\, \ldots,\, 1] \in \mathbb{R}^{1\times N}.$$

Proof. By definition,

$$\frac{\partial S}{\partial H_i} = 1 \quad \forall i.$$

Thus, the gradient is simply:

$$\frac{\partial S}{\partial H} = [1,\, 1,\, \ldots,\, 1] = 1^\top.$$

Description of the Dormand–Prince RK45 Method

The Runge–Kutta method of order 5(4), commonly referred to as RK45, is an explicit embedded Runge–Kutta scheme developed by Dormand and Prince (1980). It is widely used for solving ordinary differential equations numerically due to its efficient error estimation and adaptive step size control.

Mathematical Formulation

Consider an initial value problem (IVP) of the form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0.$$

The RK45 method advances the solution from t_n to $t_{n+1} = t_n + h$ using the following scheme:

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f\Big(t_n + c_2 h,\ y_n + h(a_{21}k_1)\Big),\\
k_3 &= f\Big(t_n + c_3 h,\ y_n + h(a_{31}k_1 + a_{32}k_2)\Big),\\
k_4 &= f\Big(t_n + c_4 h,\ y_n + h(a_{41}k_1 + a_{42}k_2 + a_{43}k_3)\Big),\\
k_5 &= f\Big(t_n + c_5 h,\ y_n + h(a_{51}k_1 + a_{52}k_2 + a_{53}k_3 + a_{54}k_4)\Big),\\
k_6 &= f\Big(t_n + c_6 h,\ y_n + h(a_{61}k_1 + a_{62}k_2 + a_{63}k_3 + a_{64}k_4 + a_{65}k_5)\Big),
\end{aligned}$$

The next value is then estimated by combining these stages:

$$y_{n+1} = y_n + h \sum_{i=1}^{6} b_i k_i.$$

The Dormand–Prince method also computes a second, lower-order estimate for error control:

$$\tilde{y}_{n+1} = y_n + h \sum_{i=1}^{6} \tilde{b}_i k_i.$$

The local truncation error estimate is:

$$E_{n+1} = y_{n+1} - \tilde{y}_{n+1}.$$

This error estimate is used to adapt the step size h dynamically to meet a specified error tolerance.

Dormand–Prince Coefficients

The coefficients for the Dormand–Prince RK45 method are:

$$c_2 = \frac{1}{5}, \quad c_3 = \frac{3}{10}, \quad c_4 = \frac{4}{5}, \quad c_5 = \frac{8}{9}, \quad c_6 = 1.$$

$$\begin{aligned}
a_{21} &= \frac{1}{5}, \\
a_{31} &= \frac{3}{40}, \quad a_{32} = \frac{9}{40}, \\
a_{41} &= \frac{44}{45}, \quad a_{42} = -\frac{56}{15}, \quad a_{43} = \frac{32}{9}, \\
a_{51} &= \frac{19372}{6561}, \quad a_{52} = -\frac{25360}{2187}, \quad a_{53} = \frac{64448}{6561}, \quad a_{54} = -\frac{212}{729}, \\
a_{61} &= \frac{9017}{3168}, \quad a_{62} = -\frac{355}{33}, \quad a_{63} = \frac{46732}{5247}, \quad a_{64} = \frac{49}{176}, \quad a_{65} = -\frac{5103}{18656}.
\end{aligned}$$

The weights for the fifth-order solution are:

$$b_1 = \frac{35}{384}, \quad b_2 = 0, \quad b_3 = \frac{500}{1113}, \quad b_4 = \frac{125}{192}, \quad b_5 = -\frac{2187}{6784}, \quad b_6 = \frac{11}{84}.$$

The weights for the embedded fourth-order solution are:

$$\begin{aligned}
\tilde{b}_1 &= \frac{5179}{57600}, \quad \tilde{b}_2 = 0, \quad \tilde{b}_3 = \frac{7571}{16695}, \\
\tilde{b}_4 &= \frac{393}{640}, \quad \tilde{b}_5 = -\frac{92097}{339200}, \quad \tilde{b}_6 = \frac{187}{2100}, \quad \tilde{b}_7 = \frac{1}{40}.
\end{aligned}$$

An extra stage is used for the error estimate, but its contribution to the fifth-order solution is zero.

Appendix B

Detailed Formulations of ODES for the state variables

In this section, we are going to discuss further the mathematical analysis of the equations 3.1 - 3.8 that describe the dynamics of the state variables , as well as the analytical expressions for physical quantities that were approximated by NNs with empirical formulas in 3.2.2. As we have already mentioned, these analytical expressions are engine-specific and describe the physics of the mean engine model in [14]. While the work in this study is based on the same model, these expressions are used for verification. In other words, while empirical formulas constitute a more general approach for a diesel engine, the results from both empirical and analytical approaches have to be similar. Values for constants included in this mathematical analysis are given in Table B.1 and Table C.1. A deeper analysis than the one included in this work can be found in [14].

The physical quantities we are going to describe, correspond to the physical process of the diesel engine we discussed about in 3.1.1.

Cylinder The diesel engine's cylinder is associated with 3 physical quantities: the total mass flow from the cylinder W_{ei}, the total mass flow out of the cylinder Weo and the fuel mass flow int the cylinder W_f. The expressions for these terms are the following:

$$W_{ei} = \frac{\eta_{vol} p_{im} n_e V d}{120 R_a T_{im}} \tag{B.1}$$

$$W_{eo} = W_f + W_{ei} \tag{B.2}$$

$$W_f = \frac{10^{-6}}{120} u_\delta n_e n_{cyl} \tag{B.3}$$

where V_d,n_e,n_{cyl} are the displaced volume, engine speed and the number of cylinders respectively.
The analytical form of the volumetric efficiency η_{vol} is given by:

$$\eta_{vol} = c_{vol1}\sqrt{p_{im}} + c_{vol2}\sqrt{n_e} + c_{vol3} \tag{B.4}$$

The pressure ratio x_p and the volume ratio x_v in the Seliger cycle after and before combustion are modeled as:

$$x_p = 1 + \frac{q_{in} x_{cv}}{c_{V_a} T_1 r_c^{\gamma_a - 1}} \tag{B.5}$$

$$x_v = 1 + \frac{q_{in}(1 - x_{cv})}{c_{pa}[(q_{in} x_{cv}/c_{V_a}) + T_1 r_c^{\gamma_a - 1}]} \tag{B.6}$$

where x_{cv} is the fuel consumed during constant-volume combustion , c_{V_a} is the heat capacity at constant volume air, c_{pa} is the heat capacity at constant pressure of air, r_c is the compression ratio , γ_a is the heat capacity ratio of air , T1 is the temperature when the inlet valve closes after the intake stroke and mixing and the energy q_{in} of the charge is modeled as :

$$q_{in} = \frac{W_f q_{HV}}{W_{ei} + W_f}(1 - x_r) \tag{B.7}$$

where q_{HV} is a constants for the fuel's heating and x_r is the residual gas fraction.

The existence of heat loss in the exhaust pipes between the cylinder and the exhaust manifold results in difference between temperature in cylinder's exit T_e and exhaust manifold temperature T_{em}. These two types of temperatures are given respectively by the following mathematical expressions:

$$T_e = \eta_{sc}\Pi_c^{1-1/\gamma_a} r_c^{1-\gamma_a} x_p^{1/\gamma_a - 1}\left[q_{in}\left(\frac{1 - x_{cv}}{c_{pa}} + \frac{x_{cv}}{c_{V_a}}\right) + T_1 r_c^{\gamma_a - 1}\right] \tag{B.8}$$

where η_{sc} is the compensation factor for non-ideal cycles and Π_e is the pressure ratio over the cylinder,

$$\Pi_e = \frac{p_{em}}{p_{im}} \tag{B.9}$$

$$T_{em} = T_{amb} + (T_e - T_{amb})exp\left(\frac{-h_{tot}\pi d_{pipe} l_{pipe} n_{pipe}}{W_{eo} c_{pe}}\right) \tag{B.10}$$

where h_{tot} is the total heat tranfer coefficient of the exhaust pipes, T_{amb} is the ambient temperature and d_{pipe}, l_{pipe}, n_{pipe} are the pipe diameter, pipe length and the number of pipes, respectively.

EGR Valve system The mass flow through the EGR valve is modeled as :

$$W_{egr} = \frac{A_{egr} p_{im} \Psi_{egr}}{\sqrt{T_{em} R_e}} \tag{B.11}$$

where Ψ_{egr} is a parabolic function given by:

$$\Psi_{egr} = 1 - \left(\frac{1 - \Pi_{egr}}{1 - \Pi_{egropt}} - 1\right)^2 \tag{B.12}$$

$$\Pi_{egr} = \begin{cases} \Pi_{egropt} & \text{if } \dfrac{p_{im}}{p_{em}} < \Pi_{egropt} \\ \dfrac{p_{im}}{p_{em}} & \text{if } \Pi_{egropt} \leq \dfrac{p_{im}}{p_{em}} \leq 1 \\ 1 & \text{if } 1 < \dfrac{p_{im}}{p_{em}} \end{cases} \tag{B.13}$$

Also , the effective area of the EGR valve is modeled as :

$$A_{egr} = A_{egrmax} f_{egr} \tag{B.14}$$

where maximum effective area of the EGR valve A_{egrmax} and pressure ratio Π_{egropt} are constants. The function f_{egr} is modeled as :

$$f(\tilde{u}_{egr}) = \begin{cases} c_{egr1}\tilde{u}_{egr}^2 + c_{egr2}\tilde{u}_{egr} + c_{egr3} & \text{if } \tilde{u}_{egr} \leq -\dfrac{c_{egr2}}{2c_{egr1}} \\ c_{egr3} - \dfrac{c_{egr2}^2}{4c_{egr1}} & \text{if } \tilde{u}_{egr} \geq -\frac{c_{egr2}}{2c_{egr1}} \end{cases} \tag{B.15}$$

The above calculations are valid only if we assume that $p_{em} < p_{im}$

Turbocharger The dynamics of engine speed ω_t can be obtained by solving the equation 3.3 where J_t is the interia, P_t is the power delivered by the turbine, P_c is the power required to drive the compressor and η_m is the mechanical efficiency of the turbocharger, which can be calculated with the empirical formulae in Table 3.4.

The turbine mass flow W_t is modeled as :

$$W_t = \frac{A_{vgtmax} p_{em} f_{\Pi_t}(\Pi_t) f_{vgt}(\tilde{u}_{vgt})}{\sqrt{T_{em} R_e}} \tag{B.16}$$

the analytical expressions for functions f_{Π_t} and effective area of VGT valve f_{vgt} are the following:

$$f_{\Pi_t}(\Pi_t) = \sqrt{1 - \Pi_t^{K_t}} \tag{B.17}$$

$$f_{vgt}(\tilde{u}_{vgt}) = c_{f_1} + c_{f_1}\sqrt{max(0, 1 - \left(\frac{\tilde{u}_{vgt} - c_{vgt2}}{c_{vgt1}}\right)} \tag{B.18}$$

where the pressure ratio Π_t is given by ;

$$\Pi_t = \frac{p_{amb}}{p_{em}} \tag{B.19}$$

The product of the power delivered by the turbine P_t, and the mechanical efficiency of the turbocharger η_m in equation 3.3 is given by :

$$P_t \eta_m = \eta_{tm} W_t c_{pe} T_{em} \left(1 - \Pi_t^{1-1/\gamma_e}\right) \tag{B.20}$$

where turbine mechanical efficiency η_{tm} is calculated analytically as it follows:

$$\eta_{tm} = \eta_{tm,max} - c_m (BSR - BSR_{opt})^2 \tag{B.21}$$

$$BSR = \frac{R_t \omega_t}{\sqrt{2 c_{pe} T_{em} (1 - \Pi_t^{1-1/\gamma_e})}} \tag{B.22}$$

$$c_m = c_{m1}[max(0, \omega_t - c_{m2})]^{c_{m3}} \tag{B.23}$$

where BSR is the blade speed ratio, R_t is the turbine blade radius and c_{m1}, c_{m2}, c_{m3} are constants.

Compressor The two physical quantities we are interested in, and are essential for describing the physics of the compressor are the compressor power P_c and compressor mass flow W_c. The mathematical expressions for these two physical quantities are the following:

$$P_c = \frac{T_{amb}\left(\Pi_c^{1-1/\gamma_a} - 1\right)}{T_c - T_{amb}} \tag{B.24}$$

$$W_c = \frac{p_{amb}\pi R_c^3 \omega_t}{R_a Tamb}\Phi_c \tag{B.25}$$

where T_c is the temperature after the compressor. The pressure ratio Π_c is given by :

$$\Pi_c = \frac{p_{im}}{p_{amb}} \tag{B.26}$$

The volumetric flow coefficient Φ_c is a function, the output of which can be calculated analytically with the following mathematical expressions:

$$\Phi_c = \sqrt{max\left(0, \frac{1 - c_{\psi 1}(\Psi_c - c_{\Psi 2})^2}{c_{\Phi 1}}\right)} + c_{\Phi_2} \tag{B.27}$$

$$\Psi_c = \frac{2c_{pa}T_{amb}\left(\Pi_c^{1-1/\gamma_a} - 1\right)}{R_c^2\omega_t^2} \tag{B.28}$$

where Ψ_c is the energy transfer and $c_{\psi 1}$, $c_{\Psi 2}$, $c_{\Phi 1}$, $c_{\Phi 2}$ are constants.

Finally , compressor efficiency η_c is modelled as an ellipse, which depends on the pressure ratio (Π_c) and compressor mass flow (W_c),

$$\eta_c = \eta_{c_{\max}} - \mathcal{X}^T Q_c \mathcal{X} \tag{B.29}$$

where $\mathcal{X}$ is a vector and given as,

$$\mathcal{X} = \begin{bmatrix} W_c - W_{c_{\text{opt}}} \\ \pi_c - \pi_{c_{\text{opt}}} \end{bmatrix} \tag{B.30}$$

where $W_{c_{\text{opt}}}$ and $\pi_{c_{\text{opt}}}$ are the optimum values of W_c and π_c respectively. π_c is a nonlinear transformation of Π_c as

$$\pi_c = (\Pi_c - 1)^{c_\pi} \tag{B.31}$$

and Q_c is a semi-definite matrix

$$Q_c = \begin{bmatrix} a_1 & a_3 \\ a_3 & a_2 \end{bmatrix}. \tag{B.32}$$

Constant values included in the previous mathematical analysis are presented in Table B.1 Table C.1

	Symbol	Value
1	η_{cmax}	0.7364
2	c_{egr1}	-1.1104
3	c_{egr2}	0.0178
4	c_{egr3}	0
5	$cm1$	1.3563
6	$cm2$	$2.7692e+03$
7	$cm3$	0.01
8	BSR_{opt}	0.9755
9	$\eta_{tm,max}$	0.8180
10	$c_{\Psi 2}$	0
11	$c_{\Phi 2}$	0
12	c_{vgt1}	126.8719
13	c_{vgt2}	117.1447
14	K_t	2.8902
15	π_{copt}	1.0455
16	W_{copt}	0.2753
17	a_1	3.0919
18	a_2	2.1479
19	a_3	-2.4823
20	c_π	0.2708

Table B.1: Constants included in mathematical forms of functions included in the empirical formulae

Appendix C

Tables of Physical Quantities

	Description	Symbol	Value
1	Ideal gas constant of air	R_a	287
2	Intake manifold temperature	T_{im}	300.6186
3	Intake manifold volume	V_{im}	0.022
4	Ideal gas constant of exhaust gas	R_e	286
5	Exhaust manifold volume	V_{em}	0.02
6	Displaced volume of the cylinder	V_d	0.0127
7	Number of cylinder	n_{cyl}	6
8	Specific heat capacity ratio of air	γ_a	1.3964
9	Specific heat capacity at constant pressure of air	c_{pa}	1011
10	Specific heat capacity at constant volume air	c_{va}	724
11	Compression ratio	r_c	17
12	Fuel consumed during constant-volume combustion	x_{cv}	2.3372×10^{-14}
13	Heating value of fuel	q_{HV}	429×10^5
14	Diameter of exhaust pipe	d_{pipe}	0.1
15	Length of exhaust pipe	l_{pipe}	1
16	Number of exhaust pipe	n_{pipe}	2
17	Specific heat capacity at constant pressure of exhaust gas	c_{pe}	1332
18	Time constant 1 for EGR	τ_{egr1}	0.05
19	Time constant 2 for EGR	τ_{egr2}	0.13
20	Time delay constant for EGR	τ_{degr}	0.065
21	Constant for EGR overshoot	K_{egr}	1.8
22	Optimal value of pressure ratio of EGR	Π_{egropt}	0.65
23	Inertial of turbocharger	J_t	2×10^{-4}
24	Time constant for VGT	τ_{vgt}	0.025
25	Time delay constant for VGT	τ_{dvgt}	0.04
26	Specific heat capacity at constant pressure of exhaust	c_{pe}	1332
27	Specific heat capacity ratio of exhaust gas	γ_e	1.2734
28	Turbine blade radius	R_t	0.04
29	Compressor blade radius	R_c	0.04

Table C.1: List of physical constants used in the state variables' differential equations

Appendix D

Code Implementation Details

The computational work for the model was done in Python, with the core heavy lifting handled by PyTorch. Developed by the AI Research team at Facebook, PyTorch is an open-source library that has quickly become the go-to toolkit for most deep-learning projects. Thanks to its dynamic computational graph, researchers can change the network structure on the fly, making debugging easier and experimentation far more intuitive. Moreover, PyTorch takes care of automatic differentiation, so getting the gradients needed by standard optimizers is both simple and reliable. Beyond these features, the library ships with a rich collection of pre-built layers, data-loaders, and math utilities that streamline the process of assembling complex models. Because it is well documented, actively maintained, and backed by a large community, PyTorch is now favored in fields as varied as physics-informed modeling, computer vision, natural language processing, and reinforcement learning. While this study relied solely on a CPU for training-a limitation of available resources-PyTorchs seamless integration with high-end GPUs means the model can be scaled up easily whenever stronger hardware becomes accessible.

Bibliography

[1] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[2] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18(153):1–43, 2018.

[3] John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[4] GeeksforGeeks. Adam optimizer - deep learning. `https://www.geeksforgeeks.org/deep-learning/adam-optimizer/`.

[5] John B Heywood. Combustion engine fundamentals. *1ª Edição. Estados Unidos*, 25:1117–1128, 1988.

[6] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[7] Varun Kumar, Somdatta Goswami, Daniel Smith, and George Em Karniadakis. Real-time prediction of gas flow dynamics in diesel engines using a deep neural operator framework. *Applied Intelligence*, 54(1):14–34, 2024.

[8] Kamaljyoti Nath, Xuhui Meng, Daniel J Smith, and George Em Karniadakis. Physics-informed neural networks for predicting gas flow dynamics and unknown parameters in diesel engines. *Scientific Reports*, 13(1):13683, 2023.

[9] Michael A. Nielsen. Chapter 2: How the backpropagation algorithm works. `http://neuralnetworksanddeeplearning.com/chap2.html`, 2015. Accessed: 2025-06-25.

[10] Terence Parr and Jeremy Howard. The matrix calculus you need for deep learning. *arXiv preprint arXiv:1802.01528*, 2018.

[11] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[12] Nir Shlezinger, Jay Whang, Yonina C Eldar, and Alexandros G Dimakis. Model-based deep learning. *Proceedings of the IEEE*, 111(5):465–499, 2023.

[13] George Stamou. Machine learning seminar notes, 2023. National Technical University of Athens (NTUA), Academic Year 2023–2024.

[14] Johan Wahlström and Lars Eriksson. Modelling diesel engines with a variable-geometry turbocharger and exhaust gas recirculation by optimization of model parameters for capturing non-linear system dynamics. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 225(7):960–986, 2011.

[15] Johan Wahlström and Lars Eriksson. Software packages for vehicular systems. `http://www.fs.isy.liu.se/Software`. Accessed: 2025-04-01.