**FACULTY OF SCIENCE AND TECHNOLOGY**

# MASTER THESIS

Study programme: Structural and Mechanical Engineering

The spring semester, 2022
Open

Author: Tina Dinh

Course coordinator: Knut Erik Teigen Giljarhus

Supervisors: Homam Nikpey Somehsaraei
Knut Erik Teigen Giljarhus

Thesis title: Use of ANN for monitoring application of a distributed energy generation system based on mGT

Credits (ECTS): 30

Keywords: micro gas turbine, artificial neural networks, autoencoder, predicative maintenance, condition monitoring, denoising autoencoder

Pages: 92

+ appendix: 70

Stavanger, 14/06/22

# *Abstract*

In today's dynamic energy landscape, renewable energy sources are steadily increasing their share of electricity on the grid. The world population recognises the benefits of going carbon neutral and is now willing to invest heavily in this cause. The biggest drawback with renewables like wind and solar is their intermittent nature, thus not being able to meet the demand timely. Retrofitting state-of-the-art machinery is a viable solution for satisfying the increasing need for electricity. Low-emission technology complementary to renewable energy production should be invested in and researched. The goal of this thesis is dedicated to precisely this, studying the potential of innovative solutions for future energy systems.

Research has been conducted at the Vrije Universiteit in Brussels on transforming a micro gas turbine into a micro humidified air turbine. The results have shown numerous benefits, including reduced levels of $NO_x$ and increased electrical efficiency. However, there are still areas that need improvement, and in cooperation with the University of Stavanger, a task has been set to develop data-driven models adapted for condition monitoring. These models are built using sensor measurements, which will be used to predict failures and contribute to reliable operation.

In this work, autoencoder models have successfully been trained and evaluated in detail. The task of denoising sensor measurements has produced satisfying results and has significantly enhanced the data quality. These results will be used as a preprocessing step to improve the performance of the multi-layered perceptron developed in association with this project. The second task was to develop an autoencoder model that should be able to give early alerts based on normal- and faulty operational data. A suitable baseline model has been identified for this purpose. However, a residual calculation has not been performed due to the lack of time. The development and analysis of such a model are suggested for future work.

# *Acknowledgements*

I want to dedicate a special appreciation to Homam Nikpey Somehsaraei, my supervisor for this thesis. First and foremost, for being understanding and managing to adapt to all of my situations the past year. It has been challenging at times, but the results are gratifying. Second, for letting me participate in this project, this is precisely how I imagined my final thesis would turn out. It has been my great pleasure to work with you, Homam. I have genuinely enjoyed working on this topic, and I hope it reflects in this work.

I also want to address a huge thanks to the Brussels team for their amazing work on this project. Your efforts has laid the foundation for exciting research, and I wish you good luck in the continuation.

Also, thank you, Ingrid and Karl, my good friends, for convincing me to convert to Latex, or else this thesis would never be finished in time.

Tina Dinh

Munich, June 2022

# *Contents*

**Abstract** iii

**Acknowledgements** v

**Contents** vii

**List of Tables** xi

**List of Figures** xvi

**List of Symbols** xvii

**1 Introduction** 1

1.1 Motivation . . . . . 1

1.2 Objective . . . . . 2

1.3 Limitations . . . . . 3

1.4 Methodology . . . . . 4

1.5 Thesis outline . . . . . 5

**2 Technology** 7

2.1 Micro humid air turbine cycle . . . . . 7
2.2 System description: mHAT setup . . . . . 8
2.3 Monitoring with data-driven methods . . . . . 12
2.4 A brief introduction to machine learning . . . . . 13
2.4.1 Artificial neural networks . . . . . 14
2.4.2 Autoencoders . . . . . 17

**3 Development of autoencoder model 21**
3.1 Preprocessing . . . . . 22
3.1.1 Train Test Split . . . . . 23
3.1.2 Feature scaling . . . . . 24
3.2 Training . . . . . 24
3.2.1 Input and output parameters . . . . . 24
3.2.2 Selection of training and validation data . . . . . 25
3.2.3 Define Keras model . . . . . 27
3.2.4 Fine-tuning of Hyperparameters . . . . . 29
3.3 Testing . . . . . 29

**4 Results 31**
4.1 Reconstruction loss . . . . . 31
4.2 Best network . . . . . 36
4.2.1 Parameters with high levels of noise . . . . . 37
4.2.2 Parameters with low levels noise . . . . . 40
4.2.3 Parameters with zero noise levels . . . . . 44
4.3 Best parameters . . . . . 46
4.4 Fine-tuned model . . . . . 54

4.5 Changed number of inputs . . . . . 57

**5 Conclusion** **63**

**6 Future work** **65**

**References** **67**

**A Unprocessed features** **73**

**B Results from reconstruction loss calculations** **81**

**C Jupyter Notebook code** **85**

# *List of Tables*

3.1 The table compares the category proportions for feature $T_1$ generated using stratified sampling and random sampling . . . . . 23

3.2 The distribution of samples in training-, validation- and test set . . 26

4.1 The ranking table in order to identify the best models. The indication of a cross (x) is equivalent to a point, thus represents an acceptable result. . . . . 35

4.2 The configuration of for the best network before hyperparameter fine-tuning. . . . . 36

B.1 Overview over MSE loss with varying number of neuron configurations . . . . . 82

B.2 Overview over MAE loss with varying number of neuron configurations . . . . . 83

# *List of Figures*

2.1 Schematic of the modified mGT cycle from VUB testing rig [1]. When a saturation tower is added between the compressor outlet and recuperator inlet, the water saturates compressed air and the cycle runs in mHAT mode. . . . . 9

2.2 The relation between artificial intelligence, machine learning and deep learning. . . . . 14

2.3 An image from *Towards Data Science* [2] showing a single brain biological and single artificial neuron. A neuron is a connecting point for input signals which are transformed and transmitted as outputs . . . . . 15

2.4 Stacked autoencoder architecture which is made of an encoder-, bottleneck- and decoder part. This network consists of one input and output layer and three hidden layers. . . . . 18

3.1 All input parameters have been plotted with respect to the same time index. This visualisation contains all data measurements with the exception of removed outliers. . . . . 25

3.2 An image from *Medium* [3] showing different ways to fit a model. The goal of the training is to develop a balanced model that is neither underfitting nor overfitting the data . . . . . 26

4.1 Mean Absolute Error (MAE) as reconstruction loss function for different model node configurations . . . . . 32

4.2 Mean Squared Error (MSE) as reconstruction loss function for different model node configurations . . . . . 33

4.3 Example of two networks with vast different denosing capabilities. The reconstruction loss is well reflected through the visualisation. 34

4.4 Best network prediction for $\Delta P$ . . . . . 38

4.5 Best network prediction for $P_{gen}$ . . . . . 38

4.6 Best network prediction for TOT . . . . . 39

4.7 Best network prediction for $T_1$ . . . . . 39

4.8 Best network prediction for CCIT . . . . . 40

4.9 Best network prediction for N . . . . . 41

4.10 Best network prediction for COT . . . . . 41

4.11 Best network prediction for COP . . . . . 42

4.12 Best network prediction for $EIT_1$ . . . . . 42

4.13 Best network prediction for $EIT_2$ . . . . . 43

4.14 Best network prediction for $RIT_a$ . . . . . 43

4.15 Best network prediction for $P_{amb}$ . . . . . 44

4.16 Best network prediction for $V_{wSat}$ . . . . . 45

4.17 Best network prediction for $T_{wSatIn}$ . . . . . 45

4.18 Best network prediction for $T_{wIn}$ . . . . . 46

4.19 Network 2-10 has produced the best results for $P_{gen}$. . . . . 47

4.20 Network 3-10 has produced the best results for $T_1$, COT and $RIT_a$. 48

4.21 Network 4-10 has produced the best results for CCIT and COP . . 49

4.22 Network 5-10 has produced the best results for N, $EIT_1$, $EIT_2$ and $T_{wSatIn}$. . . . . 50

4.23 Network 1-6 has produced the best results for $P_{amb}$. . . . . 51

4.24 Network 2-8 has produced the best results for $\Delta$P. Supplementary plots are presented with different neurons in the bottleneck- and mapping giving an insight of the tuning process. . . . . 52

4.25 Network 4-6 has produced the best results for TOT . . . . . 52

4.26 Network 5-8 has produced the best results for $V_{wSat}$ and $T_{wIn}$ . . . 53

4.27 $P_{gen}$ depicted to compare the visual effect of changing the patience from 50 to 200 . . . . . 54

4.28 $P_{gen}$ depicted to compare the visual effect of changing the learning rate between 0,1 and 1. . . . . 55

4.29 Depictions of the difference when changing between tanh and ReLU. 56

4.30 CCIT when using different number of input features for modelling 57

4.31 COP when using different number of input features for modelling 58

4.32 COT when using different number of features for modelling . . . 58

4.33 deltaP when using different number of input features for modelling 58

4.34 $EIT_1$ when using different number of input features for modelling 59

4.35 $EIT_2$ when using different number of input features for modelling 59

4.36 N when using different number of input features for modelling . . 59

4.37 $P_{gen}$ when using different number of input features for modelling . 60

4.38 $RIT_a$ when using different number of input features for modelling 60

4.39 $T_1$ when using different number of input features for modelling . . 60

4.40 TOT when using different number of input features for modelling 61

A.1 $T_1$ before denosing . . . . . 73

A.2 $\Delta$P before denosing . . . . . 74

A.3 $P_{amb}$ before denosing . . . . . 74

A.4 $V_{wsat}$ before denosing . . . . . 75

A.5 $T_{wsatin}$ before denosing . . . . . 75

A.6 $T_{wIn}$ before denosing . . . . . 76

A.7 P before denosing . . . . . 76
A.8 N before denosing . . . . . 77
A.9 TOT before denosing . . . . . 77
A.10 COT before denosing . . . . . 78
A.11 COP before denosing . . . . . 78
A.12 RITa before denosing . . . . . 79
A.13 CCIT1 before denosing . . . . . 79
A.14 EIT1 before denosing . . . . . 80
A.15 EIT2 before denosing . . . . . 80

# *List of Symbols*

**Abbreviations**

| | |
|---|---|
| AE | Autoencoder |
| ANN | Artificial Neural Network |
| API | Application Programming Interface |
| CHP | Combined Heat and Power |
| CM | Condition Monitoring |
| CPU | Central Processing Unit |
| CV | Cross validation |
| DAE | Denoising autoencoder |
| DL | Deep Learning |
| EvGT | Evaporative Gas Turbine |
| EU | European Union |
| GD | Gradient descent |
| GHG | Greenhouse gas |
| GPU | Graphics Processing Unit |
| HAT | Humid Air Turbine |
| ICE | Internal Combustion Engine |

MAE Mean Absolute Error

mGT Micro Gas Turbine

mHAT micro Humidified Air Turbine

ML Machine Learning

MLP Multi Layered Perceptron

MSE Mean Squared Error

O&M Operation and Maintenance

RAM Random Access Memory

ReLU Rectified Linear Unit

SVM Support Vector Machine

Tanh Tangens hyperbolicus

UiS University of Stavanger

VUB Vrije Universiteit Brussel

WAC Water Atomizing inlet ait Cooling

COP26 2021 United Nations Climate Change Conference

**Network parameters**

$\sigma$ Activation function

v Bias

$\Re$ Dimension of network layer

z Input from previous layer

$f_{\text{NN}}$ Nesten output function

M Number of samples

$\theta$ Model parameter vector

W Weight

$x$ Input

| | | |
|---|---|---|
| $\hat{x}$ | Prediction of input | |
| $y$ | Network output function | |

**Modelling parameters**

| | | |
|---|---|---|
| CCIT | Combustion Chamber Inlet Temperature | $^{\circ}C$ |
| COT | Compressor Outlet Pressure | $bar$ |
| COP | Compressor Outlet Temperature | $^{\circ}C$ |
| $\Delta$P | Pressure difference over fine filter | $mbar$ |
| $EIT_1$ | Economiser Inlet Temperature 1 | $^{\circ}C$ |
| $EIT_2$ | Economiser Inlet Temperature 2 | $^{\circ}C$ |
| N | Engine rotational speed | $rpm$ |
| $P_{gen}$ | Generated power | $kW$ |
| $P_{amb}$ | Ambient pressure | $bar$ |
| $RIT_a$ | Recuperated Air Inlet Temperature | $^{\circ}C$ |
| TOT | Turbine Outlet Temperature | $^{\circ}C$ |
| $T_1$ | Compressor Inlet Temperature | $^{\circ}C$ |
| $T_{wIn}$ | Water temperature entering system | $^{\circ}C$ |
| $T_{wSatIn}$ | Water temperature entering saturation tower | $^{\circ}C$ |
| $V_{wSat}$ | Injection flow rate of water in saturation tower | $m^3/h$ |

# Chapter I

## *Introduction*

## 1.1 Motivation

One of the primary outcomes of the 2021 United Nations Climate Change Conference (COP26) held in Glasgow was to secure net zero emissions by 2050 and keep a maximum of 1.5°C of warming within reach. In order to achieve this goal, two of the target areas are encouraging investments in renewables and accelerating the phase-out of coal [4]. Additionally, in light of the current European political situation, i.e., Russia's war in Ukraine, and natural gas from Russia making up 40% of European Unions (EU) gas imports, the European Commission has proposed to make Europe independent from Russian fossil fuels before 2030 [5]. Consequently, this is the chance for Europe to accelerate the transition to renewable energy.

How energy is generated, distributed, and consumed is rapidly changing. Historically the power production systems in Europe have been built to accommodate central power plants, nuclear plants and hydropower stations. However, the energy landscape is under transition driven by digitalisation, decarbonisation and decentralisation trends. Decentralised power generation is becoming an essential part of the energy transition because of the increasing amount of intermittent renewable energy on the grid, the expensive nature of energy storage, and the need for grid resilience.

The drivers for decentralisation are not only to reduce pollutant emissions and increase the share of renewables but also to improve energy efficiency, increase power production capacity, and support the grid. Micro Gas Turbines (mGTs) could be an interesting option in small-size energy generation as the most competitive alternative to the Internal Combustion Engine (ICE). Benefits like significantly lower $CO_2$ and $NO_x$ emissions, reduced Operation and Maintenance (O&M) costs and lower noise and vibration levels makes the mGT an attractive option [6, 7].

mGTs can be used in many applications such as continuous power generation, premium power, peak shaving, emergency standby, remote power, mechanical drive, wastes and bio-fuels [8]. However, the most promising application lies in Combined Heat and Power (CHP) production, with the capability of delivering a power output ranging from 30 $kW_{el}$ to 200 $kW_{el}$ and 450 $kW_{th}$. Additionally, mGTs offer operation and fuel flexibility, low maintenance costs, and short ramp-up times [9]. In 2019, estimates suggest that CHP plants generated 11.7% (348TWh) of EU electricity demand. Approximately 50% of this came from plants with less than 10 MWe capacity [10, 11].

Despite the benefits, the mGT has not been able to penetrate the CHP market as the ICE is still preferred mainly due to its higher electrical efficiency and lower investment costs [8]. The economic performance of the mGT stumbles when there is no heat demand, and consequently, the installation needs to be shut down. Therefore, for the mGT to be profitable, improved electric efficiency and flexibility are needed. Research on humidification of mGTs has shown great potential to increase electrical efficiency by decoupling the heat and electricity production [12, 13]. By humidifying the cycle, the heat in the exhaust gases can be utilised to warm up water to be re-injected into the cycle during periods with low heat demand [14].

Moreover, for the humidified mGT to become a viable alternative, operation and maintenance need to have a reliable monitoring system with fast processing, real-time surveillance and the possibility for ad hoc installations for existing systems. A maintenance program should be made so that end-users can monitor and do the essential measures. This thesis will use Artificial Neural Networks (ANN) to develop a data-driven model for monitoring a mGT. The final model can be used for noise reduction, feature detection and performance monitoring applications. This work is based on experimental data obtained from a humidified mGT test rig in Belgium at the Vrije Universiteit Brussel (VUB).

## 1.2 Objective

The main objective of this thesis is to investigate the potential benefits of running innovative distributed energy technologies. This will be accomplished by studying the possibilities for operational flexibility of a micro Humidified Air Turbine (mHAT) cycle, which has been identified to complement the future energy production systems. This thesis is a part of an ongoing University of Stavanger (UiS) funded research project for improving the performance of small-scale energy conversion technologies to curb future emissions. A detailed description of the objectives is as follows:

- To analyse sensor measurements from a humidified mGT test rig at VUB and

identify influential parameters. This work has given the basis for modelling and optimising a data-driven model.

- To use Python and Keras to train and validate a data-driven model using autoencoders. The model is developed based on experimental gas turbine data from a testing rig in Brussels and is beneficial for validating results on an actual application. This will contribute to improving the reliability of the technology in question. The model is expected to provide accurate and reliable alerts in real-time applications. The aim is to use the model in condition monitoring or predictive monitoring.
- To develop a denoising autoencoder that can be used as a preprocessing step or a pre-trained layer for an ANN model. This thesis is based on and is complementary to a project involving the same technology and methods. A layered feed-forward neural network has been used to analyse the system, but some results were unsatisfactory due to noisy data inputs. Thus, a part of this thesis is dedicated to improving this data with an autoencoder model.
- To develop a baseline autoencoder model capable of detecting failure using data from normal operation. Failure detection can be done by comparing the residuals between the two operational modes and is the basis for a predictive monitoring model.
- To provide a tested and validated model that can generalise well and potentially be used to evaluate and improve system performance of other monitoring applications.

## 1.3 Limitations

The key focus of the thesis is developing an autoencoder model, wherein most of the limitations lie. The main goal has been to identify the model that can give an acceptable output compared with the initial inputs. Due to the features' different nature and denoising requirements, no specific criteria have been set to define what an "acceptable output" looks like.

The hyperparameter space of the networks has been explored and compared as best possible. However, some hyperparameter combinations are still left to be evaluated due to the time limit, leading to improved network performance.

Furthermore, a network suitable for failure prediction has been identified. It is ready to be analysed but has not been tested against turbine failure data to measure sensitivity due to the restricted time frame. These limitations are further discussed, and suggestions for future work are described in chapter 6.

## 1.4 Methodology

A literature study was conducted on turbomachinery and mHATs, followed by a study on the fundamentals of ANNs and how to construct them. It was conducted in parallel with material and coding exercises in Python and Keras. The work for this thesis can be divided up as follows:

- Extensive literature has been reviewed on turbomachinery essentials, centrifugal compressors and pumps and radial turbines better to understand the components and structural influences on gas turbines. Additionally, a literary review of the current distributed energy generation technologies and progress within the innovative gas turbine field has been conducted to gain insight into the advancements and potential of the humid air turbine cycle.

- Knowledge of machine learning techniques and programming has been developed and has laid the basis for the modelling process. Study topics include machine learning fundamentals, examples of building learning projects, training models, an introduction to ANN using Keras, training of deep neural networks and representation learning using autoencoders. This part has been essential to understanding the model training process and the influences of hyperparameters and tuning. Programming fundamentals are based on exercises and examples from "Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems" [15, 16].

- The sensor measurements from the mHAT set up at VUB have laid the basis for developing a monitoring program. The data has been preprocessed and normalised, then divided into validation, training and test sets suitable for modelling. A preliminary model was developed using all the input features available and was used as the primary model to develop the base code structure. Several models with different input feature parameters were tested to investigate their significant impact on the final denoising capability, as the impact of parameters with no noise was unknown. When picking a denoising model, the goal is not to find the model that yields the lowest error but the best denoising capability; it has been chosen visually. The results of all the training runs have been thoroughly documented.

- The results of denoised inputs have been assessed and ranked to find the final model configuration. The choice has been based on the model with the best ability to generally denoise all the input parameters. At the same time, the best individual denoised inputs have also been identified, as some models

would give satisfying results for a few inputs but far less inferior for the rest of the results.

- Finally, the best model was optimised by manually tweaking a set of hyperparameters. This was done instead of using an automated grid search because the lowest errors do not give this type of analysis the best results.

## 1.5 Thesis outline

Chapter 1 gives an overview of the thesis and relates the problem to the current global energy situation, a proposal to curb greenhouse gas (GHG) emissions, an outline of the objectives and limitations, and a description of the methodology. Chapter 2 provides an overview of the mHAT technology and advances within this field. It gives a detailed description of the test rig at VUB, sensors and data collection process and the motivation for data-driven methods. This chapter also presents fundamental concepts within machine learning and neural network modelling—the model's structure and development are described in Chapter 3. Chapter 4 presents the results of the data-driven models, together with a discussion og the different models and hyperparameter choices and their impacts. Chapter 5 concludes the work introduced in this thesis and the final chapter 6 gives suggestions for improvements and future work.

# Chapter II

## *Technology*

---

This chapter gives the fundamental concepts of the mHAT cycle and the advancements within this field. It describes the test rig set up at VUB and the importance of monitoring this system. This chapter also introduces monitoring with data-driven methods and basic machine learning concepts such as artificial neural networks and autoencoders.

---

## 2.1 Micro humid air turbine cycle

In order to increase the economic performance of the gas turbine, compared to the ICE, the heat-driven component and the electrical efficiency need to be improved [8]. The conversion of the mGT into a mHAT still allows for heat production, making the cycle the perfect candidate for flexible heat production from a mGT [1, 17]. Water or steam injection allows for decoupling heat and electricity production when heating is not required. When heat demand is high, the unit can run in cogeneration mode following the traditional gas turbine configuration; when the heat demand decreases, water injection enables re-usage of the heat in the exhaust gases to increase the electrical efficiency of the engine [9].

A humidified gas turbine uses air-water mixtures as working fluid and promises high electrical efficiencies, high specific power outputs, reduced specific investment costs, reduction of $NO_x$ in the combustor, reduced power output degradation and improved part-load performance compared with simple cycles [18].

There are several options for humidifying a gas turbine cycle: steam injection, water injection and evaporative cycles with humidification towers. The main idea

is the same: injected water or steam increases the mass flow rate through the turbine and thus increases the specific power output because much less work is required to increase the pressure of a liquid than a gas. Additionally, the cycle efficiency is raised when the gas turbine exhaust preheats water or generates steam for injection or preheating the combustion air in a recuperator.

In 2005, Jonsson and Yan published "Humidified gas turbines—a review of proposed and implemented cycles", a literature review of the most significant humidified gas turbine research and development published at the time [18]. The paper points out that the Humid Air Turbine (HAT) cycle offered the highest potential efficiency increase compared to the classic Brayton cycle. The HAT cycle was first patented by Rao in 1989 [19] and involves water injection in a humidification tower with a recirculation water loop. The interest in this type of evaporative gas turbine cycle increased in the 1980s, with research programs such as the HAT project in the US, followed by the Evaporative Gas Turbine (EvGT) project in Sweden. The Swedish research program was initiated to demonstrate the EvGT in a pilot plant, investigate the humidification process and propose future plant designs. Many configurations and fuels have been tested, especially tackling the flow mismatch between the compressor and turbine. Various cycle layouts and modifications have been researched and suggested, but there is still only one EvGT in operation today; the pilot plant is in Lund, Sweden.

Research by De Paepe identified the mHAT cycle as the perfect candidate for waste heat recovery through humidification with limited necessary cycle modifications [20]. Montero Carrero et al. have analysed the economic advantages of converting into a mHAT for domestic operations [21]. Simulations to investigate the thermodynamic efficiency have been assessed by Parente et al. [22], alongside Wang and Xiao, who studied the thermodynamic effects of humidification on mGT components [23]. Experimental tests have been performed by coupling a mHAT with a Water Atomizing inlet air Cooling (WAC) line. Nakano et al. could show a 3% efficiency increase through water injection experiments [24] while Dodo et al. could exhibit a 32% electrical efficiency and lowered $NO_x$ levels. A recent study by Wei and Zang demonstrated an increase in the power output when investigating the off-design behaviour of a small-sized HAT [25]. Moreover, the first experiments at VUB by De Paepe et al. could document an electrical efficiency increase of 1.2% and 2.4% [26].

## 2.2 System description: mHAT setup

This thesis is based on a modified Turbec T100 mGT installed at VUB. In cogeneration mode, otherwise dry operation mode, the T100 produces a power output of 100$kW_e$ and 166 $kW_{th}$ at a rotational speed of 70 000 rpm and total energy

efficiency of 80%, with an electrical efficiency of 30%. It follows the recuperated Brayton cycle as indicated by the black components on Figure 2.1. Additionally, an innovative spray saturation tower has been developed and integrated to reduce pressure and efficiency loss for the VUB testing rig [27].

Air is compressed in a radial compressor and humidified by spraying hot water in the saturation tower. The water vapour content increases air enthalpy and air mass flow as the air advances through the saturator. At the same time, heat is extracted from the circulating water below boiling temperature. Afterwards, the saturated air is preheated by the exhaust gases in the recuperator before entering the combustion chamber, where the air is burned with natural gas increasing the maximum turbine inlet temperature of 886°C. The hot gases are expanded over the turbine and deliver mechanical power to drive the compressor and the high-speed generator for electric power production. After passing through the recuperator, the exhaust gas heat is used to heat the water in the economiser. The hot water is then routed towards the saturation tower, which is sprayed over the air from the compressor. However, only 2% of the sprayed water evaporated in the saturator, and the rest will be pumped back to the water heater.

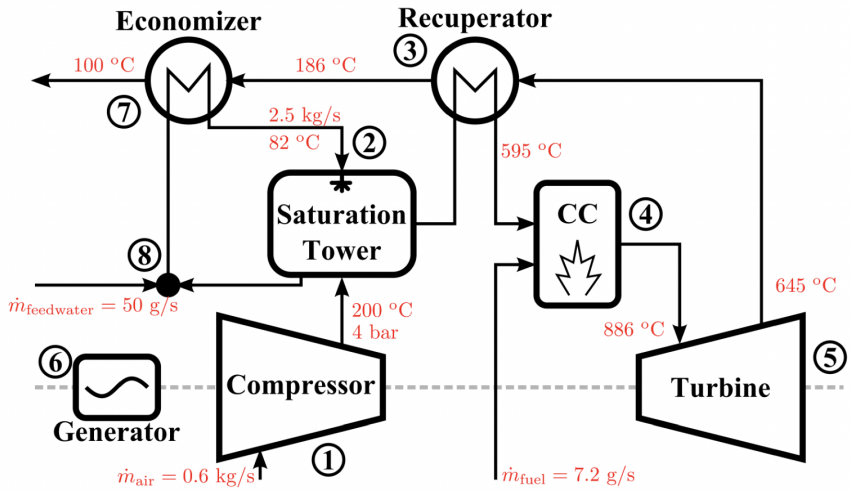



**Figure 2.1:** Schematic of the modified mGT cycle from VUB testing rig [1]. When a saturation tower is added between the compressor outlet and recuperator inlet, the water saturates compressed air and the cycle runs in mHAT mode.

The test rig is equipped with sensors to measure the effect of the water injection. The following sensors have been installed at the mGT unit:

- Temperature sensors to monitor water and compressed air temperature
- Thermocouples to measure turbine outlet-, compressor inlet- and recuperator outlet temperature
- A differential pressure sensor to measure pressure loss over the saturation tower
- Two rotameters, one to measure the amount of injected feedwater rate and the other measures the exact amount of injected water into the saturation tower.
- A water flow rate meter to measure the circulating water mass flow rate
- A flow meter that monitors the injected natural gas flow rate
- A vortex flow meter to measure compressor mass flow rate
- Rotational speed sensors and electrical power measurers have been installed to identify the impact of humidification on the total cycle performance

The Turbec T100 has a control system composed of two primary operational modes, which allows for an efficient nominal and part-load operation:

- Turbine Outlet Temperature (TOT) control - the TOT is kept at a constant of 645°C. The thermal input of the combust chamber can be altered by adjusting the valve opening time of the fuel valves in the combustion chamber and thus the injected rate of natural gas.
- Power output control - the engine delivers a constant power output by varying the rotational speed of the shaft.

The TOT has been kept constant for the experimental test runs with a slight change in the power output. This configuration has allowed for operation at a constant rotational speed. Tests are typically performed by starting the engine using a specific start-up procedure and include water injection before the engine starts, which allows for improved flame stability. After reaching a steady state, the engine typically runs for at least 30 minutes at a predefined power output or rotational speed with constant and stable water injection before switching to a new setpoint. All cycle parameters documented during these tests are taken with a sampling ratio

of 10Hz. Detailed descriptions of the mGT testing rig and installed measurement equipment have been thoroughly documented in [28, 29].

Nevertheless, some challenges must be addressed when converting to a mHAT cycle. The introduction of the saturation tower and additional piping to the system changes the operating conditions for the mGT. The additional components lead to a pressure loss, and a pressure mismatch arises between the compressor outlet and the turbine inlet. Compressor surge is a result of pressure loss over the humidification unit, piping network and reduced surge margin [28]. Moreover, adding water to the compressed air increases the mass flow entering the turbine, and since the turbine limits the total mass flow rate, the compressor mass flow rate is reduced. This results in pushing the compressor operating point closer to the surge margin. The injection rate must be restricted to keep the pressure ratio below the surge line, which is the limit of stable operation for the compressor. Compressor blades tend to stall in the surge region, which reduces the flow between the blades and can lead to flow separation. Some working fluid may rotate opposite the rotor, resulting in vibrations that may damage the compressor [30]. The cycle is equipped with a series of valves to limit these activities as much as possible including:

- A bleed valve to increase the compressor surge margin
- A blow-off valve protects the compressor from a shutdown surge.
- Bypass valves enable dry mode operation by bypassing the saturation tower.

Moreover, the humidification of the working fluid negatively impacts combustion efficiency and flame stability. The increase in water content alters the specific capacity of the working fluid, causing a reduction in the combustion chamber temperature and reaction rate, which leads to lower efficiencies and rise in CO emissions. The combustion chamber maintains the flame during wet operation while keeping a high combustion efficiency (>99%). The problems are mainly observed during the start-up phase, load changes or oversaturation. For example, during start-up, due to the sudden injection of water, the composition of the combustor inlet air changes rapidly. The existing control system cannot anticipate this change, resulting in a flameout. When a flameout occurs, the pressure ratio and rotational speed decrease, leading to a compressor surge. The bleed valves must stay open during wet start-up to keep the compressor operating point away from surging. Opening the blow-off valve during engine shutdown also keeps the compressor away from compressor surge.

The underlying factors leading to compressor surge are not fully understood, making it hard to predict. The existing Turbec T100 mGT controller is not able to detect

surges. When a drop in rotational speed and produced power was detected in past experiments, it would always be alerted as a flameout, as there is no actual flame detector installed in the combustor. Compressor surge should be avoided at all costs since, ultimately, it can damage and lead to destroying the compressor. Therefore, flame-out needs to be detected quickly and ensure that the blow-off valve can be opened before compressor surge. Today there are no available control options for flame-out detection or surge prevention. Previously, alternative measurements like TOT, pressure and rotational speed drop have been used to predict flame-out but have proven unsuccessful as surge occurs before any drop has been observed.

In a cooperation project with UiS, it is expected that by using data-driven methods, potential flame-outs can be predicted with better accuracy and fast enough to protect the compressor and avoid surges and unpredictable engine shutdown.

## 2.3 Monitoring with data-driven methods

Condition Monitoring (CM), otherwise predictive maintenance, has changed over the past decades. Traditionally CM has been based around vibration analysis, but modern-day sensors and software can provide reliable alerts in real-time whenever a change is detected. This makes CM one of the most innovative solutions for anticipating failures in machinery and is therefore widely used in the industrial sector. CM systems are crucial for understanding the behaviour of machinery, especially rotating machinery [31].

The advantages of CM implementation are many; early detection of damage allows for better maintenance planning and preventive actions to prevent further failure and unplanned downtime. The CM system can improve operation safety, increase plant availability through efficient operation and consistent quality, and reduce costs. Early detection, while the damage is still slight, can provide meaningful insight into machine design improvement and development.

CM can be performed using different methods such as physics-, model-, or data-driven methods [32]. Regardless of the approach, a monitoring system should be able to distinguish normal variation apart from variation caused by failure, degradation or sensor faults. The decision between a mathematical and a data-driven modelling approach depends on what kind of information is available for model development. A popular model-based approach is the Kalman filters. Whilst this technique for fault detection has advantages such as on-board and real-time implementation, their health monitoring reliability decreases with increasing system nonlinearity, complexity and model uncertainties [33]. Developing an accurate mathematical model that considers modelling errors and uncertainties can be complicated because sources of uncertainty are not easily quantifiable.

On the other hand, data-driven methods work on historical data captured within a time-varying interval, an excess resource not used to its full potential. Data-driven methods are essential in modern monitoring systems, especially for large-scale industry applications, since they do not require many computations. Hence, they are compatible with the real-time constraints of dynamic complex systems [34].

Data-driven approaches can be distinguished between supervised and unsupervised learning. In supervised learning, it is necessary to define the classes and label the training data before the training procedure. Meanwhile, unsupervised learning consists of a feature extraction step that maps the high-dimensional vectors to feature space to find specific projecting vectors with low-dimension. Supervised learning methods use historical data to construct a learning model for fault detection and diagnosis of new data. The most popular methods are Bayesian Networks, and ANNs [35]. ANN is the most common data-driven method for gas turbine modelling, monitoring and diagnosis applications. It is commonly utilized in fault detection and isolation and for developing condition-based diagnosis models [36].

In the last decade, more studies can be found using ANNs for CM and diagnostics for rotating machinery, which has repeatedly proven to outpace existing methods. These results come from neural networks' abilities to find non-linear relationships between input and output data. Asgari et al. developed system identification models for gas turbines using ANN techniques, and the final model was able to predict performance with high accuracy [37]. Barad et al. trained a health-monitoring model capable of giving robust and early warnings using mechanical parameters. The ANN techniques used were feed-forward neural networks, and back propagation [38]. Sampath and Singh presented a nested neural network function that could be used as a pre-processor or filter to reduce the number of network fault classes. The model gave results with improved accuracy, reliability and consistency [39]. A study by Yoon et al. evaluated the implanted deterioration data from a mGT. The data was used to train an ANN to evaluate component failure and network predictability. [40]. Needless to say, the development of data-driven methods can lead to more accurate and computationally effective engine assessment systems.

## 2.4 A brief introduction to machine learning

Artificial Intelligence (AI) is a broad term for machines emulating human intelligence. The goal of AI is simple; to create autonomous machines able to replicate human behaviour. Machine Learning (ML), a subset of AI, implements algorithms to replicate human behaviour and learns by finding underlying structures within data provided by **humans**. Learning methods, also known as training methods, can be categorized as *supervised*, *unsupervised* or *reinforcement*. These ML tech-

niques have been applied successfully for numerous applications, including pattern recognition, computer vision, spacecraft engineering, finance, entertainment, computational biology, biomedical and medical applications [41]. ML has inspired the development of a network structure based on the human brain known as artificial neural networks. These networks attempt to replicate human behaviour by identifying the interrelation between input- and output variables. When information is passed through the network, the model finds methods to learn useful patterns within the data efficiently. ANNs, in turn, can be divided into shallow and deep networks. Shallow networks are accustomed to linear behaviour, while deep networks are multi-layered and are suitable for more complex and non-linear problems.

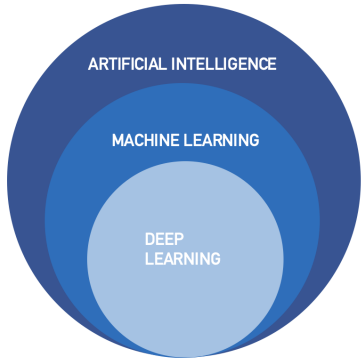



**Figure 2.2:** The relation between artificial intelligence, machine learning and deep learning.

### 2.4.1 Artificial neural networks

The first ANN architecture was introduced in "A Logical Calculus of Ideas Immanent in Nervous Activity" by Warren McCulloch and Walter Pitts [42]. The interest in neural networks has fluctuated since this paper was published in 1943. However, in the past two decades, the interest in this field has adequately matured to become one of the most popular research areas. The rise in popularity is due to breakthroughs like Deep Learning (DL), Support Vector Machines (SVM) and access to more data and better computational power. These steps are now paving the way for advancement in fields such as visual object recognition, speech processing, natural language understanding, neuroscience and medicine.

The ANN architecture is made to mimic the biological behaviour of the brain, with interconnected processing units called artificial neurons. These artificial neurons

have five components: an input, a weight, a linear transfer function, an activation function and a bias, as illustrated on Figure 2.3. Multiple neurons stacked together in a row constitute a layer, and multiple layers connected make a multi-layer neural network, or Multi Layered Perceptron (MLP). The ANN can be described as a statistical data modelling tool capable of structuring the non-linear functional relationship between a set of input and output parameters during training.

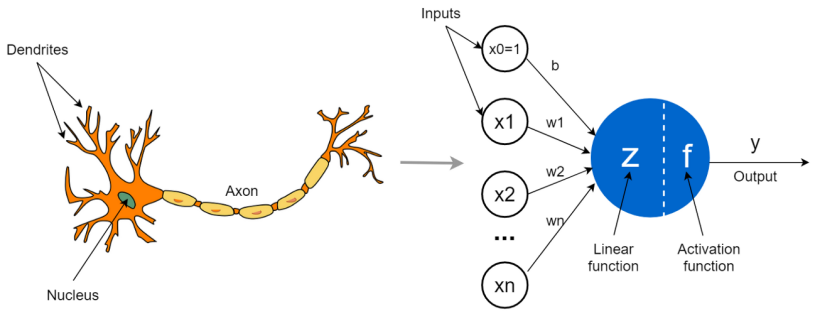



**Figure 2.3:** An image from *Towards Data Science* [2] showing a single brain biological and single artificial neuron. A neuron is a connecting point for input signals which are transformed and transmitted as outputs

When data is fed into a neural network, each neuron in one layer passes information via a transfer function to the connected neurons in the proceeding layer. The feedforward neural network is a network where information is passed from input to output via neurons. This network can provide a predicted output based on a set of inputs. The output function can be described as follows:

$$y = f_{\mathrm{NN}}(\mathbf{x}) \tag{2.1}$$

The $f_{\mathrm{NN}}$ function is a nested function due to the network layers. Equation 2.2 is an example of a 3-layer neural network that returns a scalar.

$$y = f_{\mathrm{NN}}(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))) \tag{2.2}$$

$f^{(2)}$ and $f^{(3)}$ are vectors functions that can be defined as follows

$$y = f^{(l)}(z) = \sigma^{(l)}(\mathbf{W}^{(l)}\mathbf{z} + \mathbf{v}^{(l)}) \tag{2.3}$$

Where *l* is the layer index equal to the number of hidden layers in the architecture. The activation function is denoted by $\sigma$ and is set according to the problem that will be solved. In a neural network, the activation function defines how the weighted sum of the inputs is transformed into a non-linear output. Popular activation function are logistic functions like *Sigmoid* (Equation 2.4), *Tangens hyperbolicus* (Tanh, Equation 2.5) and *Rectified Linear Unit* (ReLU, Equation 2.6) The interconnections between neurons are known as weights and are represented in the ***W*** matrix, together with the bias vector **v**, they constitute the adjustable hyperparameters of the network which are changed according to a cost function. These weights determine the importance of a given variable and are tuned during the training process to store knowledge of the trained data. Lastly, **z** represents the inputs from the previous layer.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.5}$$

$$Relu(z) = max(0, z) \tag{2.6}$$

The cost function is used to evaluate the accuracy of the trained model and will indicate how the model is performing. Going backwards through the network nodes, the model adjusts its weights and biases to reach convergence, or the local minimum. This process is called backpropagation, and with each training example, the parameters adjust to converge to a minimum gradually. The cost function is directly related to the activation function of the output layer, and for most problems, the goal is to minimise the cost function.

However, the performance of the ANNs is highly dependent on the underlying data. Consistency, completeness, and data accuracy play a significant part in training a DL model. Noisy and unclean data can make it harder for the system to detect underlying patterns, giving inaccurate or faulty predictions. A significant part of ML involves preparing data for analysis or cleaning. This process includes removing outliers, duplicate or irrelevant data, filtering missing values, and removing unwanted features. Methods for learning features from low-quality data have been presented by Vincent et al. [43], a denoising autoencoder model that learns

features and representations for noisy data. In these architectures, the model trains parameters by reconstructing the original input from the corrupted output.

### 2.4.2 Autoencoders

An autoencoder (AE) is an unsupervised type of ANN trained to copy its input from its output. By constraining the network, it is possible to learn the most useful representations from the input data efficiently. AEs are used to find structures within data by compressing representations of input data to a latent space. The latent space has a lower dimensionality than the input space, and the model must learn how to reconstruct the data from the reduced space. The typical AE architecture consists of three parts:

- *encoder* compresses the high-dimensional data into the low-dimensional latent space.
- *bottleneck*, the latent space that contains the compressed representation of the input data.
- *decoder*, decompresses the knowledge representation and reconstruct the data back to its encoded form.

A simple autoencoder architecture illustrated in Figure 2.4 and has the recognisable hourglass shape with a bottleneck layer. The input layer consists of six input nodes, each representing a data feature for this layout. The inputs are compressed to a lower dimension mapping layer. The outputs of the mapping layer are then further compressed into the bottleneck layer, which is the most crucial hyperparameter of the network. The bottleneck restricts the information flow from the encoder to the decoder, thus only allowing the most vital information to pass. The bottleneck layer is followed by the decoder that includes a (de)mapping layer and the output layer, which comprises the same number of feature nodes as in the input. The goal for the output $\hat{x}$ to be as similar to the input $x$ as possible.

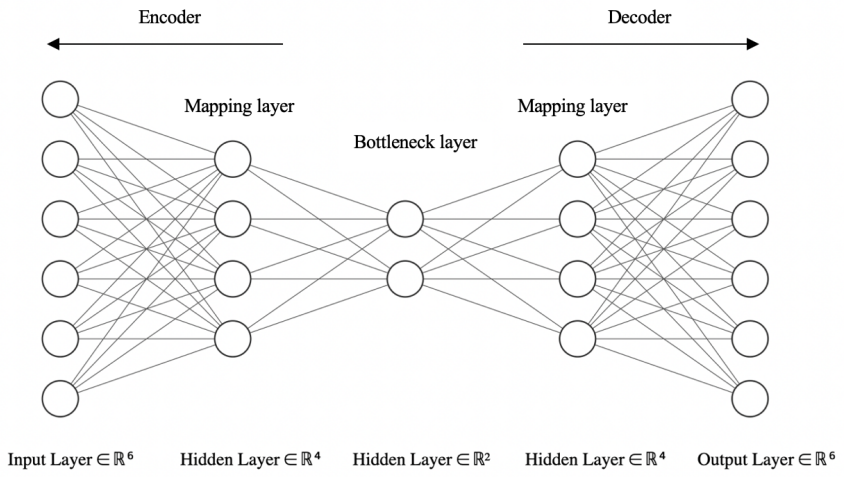



**Figure 2.4:** Stacked autoencoder architecture which is made of an encoder-, bottleneck- and decoder part. This network consists of one input and output layer and three hidden layers.

The metric for the loss between input and output is called reconstruction loss and is a part of the cost function that penalises the model when the inputs differ from the reconstructed outputs. The cost function is selected depending on the type of problem that is being solved. For a regression problem, a popular cost function is given by the Mean Squared Error (MSE) described as follows:

$$\frac{1}{M}\sum_{i=1}^{M}(x_i - \hat{x}))^2 \tag{2.7}$$

where $M$ is the number of samples, $x$ is the inputs and $\hat{x}$ is the predicted output.

Autoencoders were initially used for dimensionality reduction and feature learning but have evolved over the years and are now widely used for learning generative data models. However, an interesting practical application for autoencoders is denoising. A denoising autoencoder (DAE) should be able to find patterns in corrupted data and predict a "clean" version. For a DAE, the model that gives the lowest reconstruction loss is usually not good at generalising to new instances. Therefore, it is essential to balance the model as follows:

1. Sensitive enough to the inputs to build an accurate reconstruction
2. Insensitive enough to the inputs that the model does not memorise or overfit the training data. If the model is trained to reconstruct data perfectly, it will learn any useful data representations.

This sensitivity measure is adjusted by changing the number of neurons in the bottleneck and mapping layers.

Another practical application for AEs is anomaly detection or fault detection. The objective is to train a model based only on normal operational data and then be able to identify the samples not conforming to the normal profile as anomalies. There have been many successful studies on applying autoencoders for system monitoring of rotating machinery [44, 45]. The utilisation of AE for fault detection assumes that a trained autoencoder would learn the latent subspace of normal samples. Once trained, it would result in a low reconstruction loss for normal samples and a high reconstruction loss for anomalies.

# Chapter III

## *Development of autoencoder model*

---

This chapter describes the structure of a elementary machine learning project. The modelling choices are presented and explained in three main parts: preprocessing, training and testing. Different modelling and hyperparameter choices typical for this data are accounted for and discussed.

---

This code aims to create an algorithm capable of denoising measurement data using precisely the measurement data. The quality of specific features received from VUB is not optimal for modelling, and it could be advantageous to enhance them. The task of a neural network is to find non-linear patterns within data; when a model is based on poor data, the system's performance, accuracy and reliability will be affected negatively. The data used for network modelling comes from normal operating conditions, which are generally easily obtainable and accessible. This data can make a baseline model for detecting deviating operations. This method is cheap and effective in studying turbine conditions, potentially giving accurate failure indications within a reasonable time margin.

AEs is an unsupervised ML technique, meaning the training is done without labelling. The task itself can be categorised as a dimensionality reduction and regression task as it tries to reconstruct the inputs to the output by transforming inputs into a low dimension space. The goal is to predict "clean" and noise-free values of the original data. These specifications are essential to define as it determines which activation function and performance measure will be used for modelling.

The training and evaluation of the model have been done in Python with Keras as the framework library. Keras is a Tensorflow API and is easy to use and learn as it offers consistent and straightforward APIs. The code has been written in

Jupyter Notebook, which offers to combine software code, computational output, explanatory text and multimedia resources in a single document. The main code files can be found in Appendix C.

The data used to train the model is received from the lab in VUB. The set-up for data collection has been described in section 2.1. The parameters used in the modelling process are as follows:

| **Parameter** | **Definition** | **Unit** |
|---|---|---|
| $T_1$ | Compressor Inlet Temperature | $°C$ |
| $\Delta P$ | Pressure difference over fine filter | *mbar* |
| $P_{amb}$ | Ambient pressure | *bar* |
| $V_{wSat}$ | Injection flow rate of water in saturation tower | $m^3/h$ |
| $T_{wSatin}$ | Water temperature entering saturation tower | $°C$ |
| $T_{wIn}$ | Water temperature entering system | $°C$ |
| $P_{gen}$ | Generated power | *kW* |
| N | Engine rotational speed | *rpm* |
| TOT | Turbine Outlet Temperature | $°C$ |
| COT | Compressor Outlet Temperature | $°C$ |
| COP | Compressor Outlet Pressure | *bar* |
| $RIT_a$ | Recuperated Air Inlet Temperature | $°C$ |
| CCIT | Combustion Chamber Inlet Temperature | $°C$ |
| $EIT_1$ | Economiser Inlet Temperature 1 | $°C$ |
| $EIT_2$ | Economiser Inlet Temperature 2 | $°C$ |

The process of model development and the structure of the code is divided into the three following parts:

- *Preprocessing*, preparing the data for training which includes splitting and normalising the data
- *Training*, using training and validation to make sure that the model is not overfitting the training set and tune the network to find the optimum model performance
- *Testing*, test the generalisation ability of the network on an unseen testing data set

## 3.1 Preprocessing

Data cleaning and preparation is an essential preprocessing step in modelling, as data is crucial for determining the quality of model predictions and the ability to

generalise. Utilising time to explore preprocessing techniques can increase the model accuracy and save computational costs. The data has already been cleaned for the most significant outliers beforehand. Additionally, the input parameters containing excessive outliers have been left out entirely of the modelling process.

### 3.1.1 Train Test Split

The only way to know how well a model will generalise to new instances is to present it to new instances. Therefore, an essential preprocessing step is to split the data set into two parts: a training set and a testing set. The testing set is only used at the end of the training to assess the performance of a fully-trained model. This set estimates the loss rate, also named generalisation loss, after the final model has been chosen. After assessing the final model on the test set, the model should no longer be tuned. This type of evaluation gives an estimate of how well the model is going to perform on unseen data.

It can be convenient for data sets with many instances to split data randomly, usually a division of 80% training data and 20% testing data. In this study, there has not been generated a significant amount of data; thus, to avoid over-representation of sampling groups and sampling bias, it is convenient to divide the data set using stratified sampling. This method involves dividing the data into representative subgroups (strata) and then splitting these subgroups into a training and testing set. Previously different features have been analysed using stratified sampling, and the parameter for $T_1$ was considered the most appropriate for this task. In addition, $T_1$ pose as a critical parameter in an air aspirated engine such as the mGT, as the ambient air temperature and, in turn, the air density has a significant impact on the engine performance. The feature was divided into four strata and split into 80% training and 20% testing. Table 3.1 gives an overview of the overall average value of each stratum when comparing random and stratified sampling. It can be seen that the % error for stratified sampling is almost identical to the complete training set, while purely random sampling gives skewed values.

**Table 3.1:** The table compares the category proportions for feature $T_1$ generated using stratified sampling and random sampling

| Strata | Overall | Stratified | Random | Strat%error | Rand. %error |
|---|---|---|---|---|---|
| 1 | 0.272759 | 0.272780 | 0.272416 | 0.007774 | -0.125854 |
| 2 | 0.201493 | 0.201487 | 0.202508 | -0.002916 | 0.503582 |
| 3 | 0.273911 | 0.273874 | 0.272853 | -0.013561 | -0.386148 |
| 4 | 0.251837 | 0.251859 | 0.252223 | 0.008663 | 0.153393 |

After applying stratified sampling to split the training and data set, it is important to arrange the indexes chronologically for visualisation purposes.

Finally, the testing set needs to be saved as a separate file and is used after the training is finished. All data exploration must be carried out on the training set separately to avoid data leakage, which is when information from outside the training data set is used to create the model.

### 3.1.2 Feature scaling

The last preprocessing step is feature scaling. This scaling ensures that features with higher magnitudes will not govern or control the trained model. In general, algorithms do not perform well when attributes are scaled differently, and the sensors that are measuring parameters are, by nature, scaled differently. Min-max scaling and standardisation are the two most popular scaling methods. In this thesis, min-max scaling was used to scale the entries between 0 and 1, as this is the preferred method when there are no significant outliers in the data.

$$x_{\text{normalised}} = \frac{x - min(x)}{max(x) - min(x)} \tag{3.1}$$

## 3.2 Training

This is considered the central part of the development task, also known as the modelling part. This part uses the training and validation set to build a model and perform error analysis. Different hyperparameters and optimisation methods are described to understand specific choices and trade-offs.

### 3.2.1 Input and output parameters

The task of an AE is to copy its outputs from its inputs. Thus, all parameters are set as inputs, and the outputs will be the denoised version of the inputs. The visualisation of the data parameters to time can be seen in Figure 3.1. Individual plots of the inputs can be further studied in Appendix A.

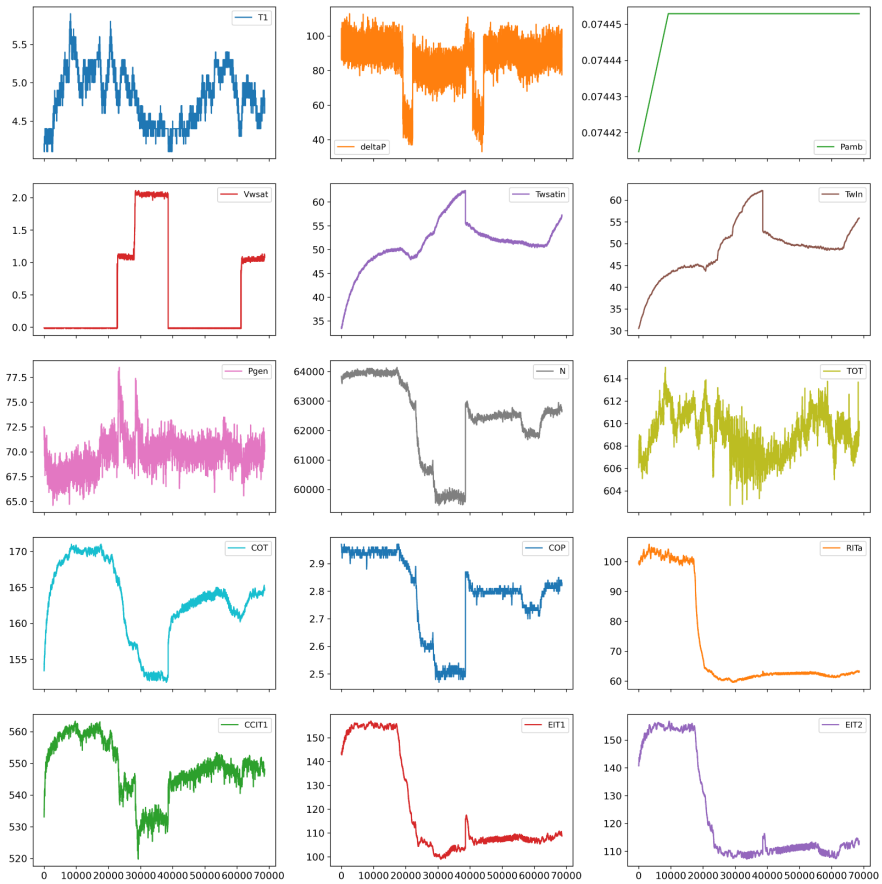



**Figure 3.1:** All input parameters have been plotted with respect to the same time index. This visualisation contains all data measurements with the exception of removed outliers.

### 3.2.2 Selection of training and validation data

Before training the model, the training set has been split once more, introducing a third validation set. While a testing set is used to estimate the generalisation capability for new instances, a validation set provides an unbiased evaluation of the model to find the optimal hyperparameters. The model is trained on the training set and simultaneously evaluated by the validation set after each training epoch. Validation loss is the metric to assess the models' performance on the validation set.

Overfitting is when a model fits precisely to the training data, which means that the model is not likely to perform well on new data samples. Thus, a validation set also ensures that the model will overfit to data by terminating the training process once it detects that the network is sufficiently trained. Overfitting can be verified when testing the final model: if the training loss is low and generalisation is high, the model is overfitted to the training data.

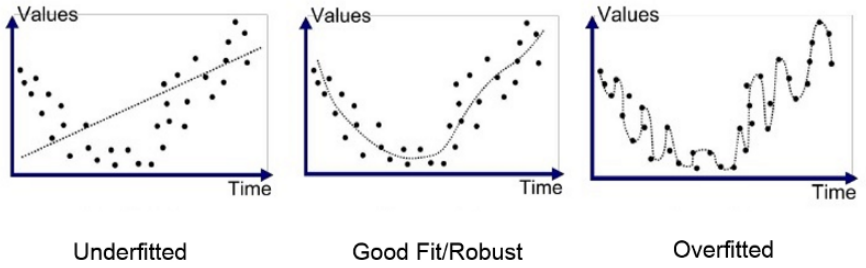



**Figure 3.2:** An image from *Medium* [3] showing different ways to fit a model. The goal of the training is to develop a balanced model that is neither underfitting nor overfitting the data

Balancing the train, validation, and test split is essential. If the training set is too small, there will be high variance in the training set, and the network will not have enough data to learn. On the contrary, if the validation set is too small, evaluation metrics will have a significant variance, and the model will not be tuned adequately. A common ratio has not been established to cover all problems and networks. However, a rule of thumb for smaller data sets is splitting into a 70:30 or 80:20 ratio.

The traning-validation set split was split using randomized sampling. The data set was split into an 80% training part and 20% validation part. The total number of samples is 68 621 and are split as shown in Table 3.2.

**Table 3.2:** The distribution of samples in training-, validation- and test set

| Training | Validation | Testing |
|---|---|---|
| 43 896 | 10 979 | 13 718 |

### 3.2.3 Define Keras model

The model is fully connected (dense) and based on Keras' sequential API. A dense network is a model where layers stack on top of one another, and each neuron is connected to every unit in the next layer. An AE architecture has two independent parameters: the number of neurons in the bottleneck and mapping layers. The goal is to compress the data as much as possible, simultaneously preserving the data reconstruction capability. The number of inputs is equal to the number of parameters that are being modified. The dimensions of the layers are described in Equation 3.2.

$$\Re^I > \Re^M > \Re^B \tag{3.2}$$

Where $\Re^I$, $\Re^M$ and $\Re^B$ are the dimensions of the input-, mapping- and bottleneck layer, respectively.

Several combinations of layer- and neuron numbers have been used to evaluate the model's performance, including models with no mapping layer. There are 15 input parameters; thus, the mapping layer neurons have been tested for 6, 8 and 10. Bottleneck neurons between 1 to 5 have been tested. One neuron in the bottleneck is when the information is the most compressed, and the reconstruction loss is expected to be higher than for a bottleneck of 5. For this setup, the reconstruction loss is based on validation loss values. Another popular choice of cost function besides MSE is the Mean Absolute Error (MAE) as shown in Equation 3.3. Both metrics have been used for the evaluation of the model.

$$\frac{1}{M}\sum_{i=1}^{M}|x_i - \hat{x}| \tag{3.3}$$

The goal is for the network to minimise the cost function, which is done through the optimisation algorithm Gradient Descent (GD). This algorithm iteratively adjusts the parameters until it has reached a minimum. It measures the local gradient of the cost function with regard to $\boldsymbol{\theta}$, which is the vector containing all model parameters and is initialised with random numbers at the start of the training. As the network starts to learn from instances, the parameters are adjusted, and $\boldsymbol{\theta}$ follows the direction of descending gradient by taking the partial derivative of each model parameter $\theta_j$

$$\frac{\partial}{\partial \theta_j} MSE(\boldsymbol{\theta}) = \frac{2}{N} \sum_{i=1}^{N} (\theta^T x_i - \hat{x}) x_j \tag{3.4}$$

An important parameter of GD is the learning rate, a hyperparameter that determines the step of how quickly the model adapts to the problem. The learning rate needs to be set correctly, as a high learning rate can cause the model to converge too quickly to a sub-optimal solution (local minimum). In contrast, with a low learning rate, many iterations are needed for the network to converge, spending unnecessary computational power and time. The best practice when finding the correct learning rate is to start with a low learning rate ($10^{-5}$) and augment with a constant value, for example, the log scale, right until the reconstruction loss starts to increase [46].

For this model, a variation of GD has been implemented, namely Mini-batch Gradient Descent. This algorithm splits the training data set into smaller parts, called mini-batches, each used to calculate the model loss and update the weights. This implementation allows for a more robust convergence, avoiding the local minima [16]. Due to these reasons, mini-batch GD is the most popular variant of GD in DL. The choice of batch size influences both network performance and training time. Though data scientists are still discussing the different effects, using the largest batch size possible for the GPU RAM has been recommended to allow the most effective utilisation of GPU acceleration. However, the practical implementation of large batch sizes tends to lead to modelling instabilities, especially at the beginning of training.

On the contrary, it has been argued that small batch sizes can be more advantageous, yielding better models in less training time. Nevertheless, as previously stated, it is more sensible to use a smaller batch size for this network due to the small number of available data samples. Around one thousand samples were contained in each batch, and a mini-batch size of 42 represents how many times the model parameters were updated in one epoch. Other mini-batch sizes have also been explored to observe the effect of CPU and GPU processing. This change dramatically affected the training duration, both in the number of epochs until convergence and the time spent calculating each epoch.

The number of training epochs is closely related to GD, which quantifies how long the model will train. The epoch number is a critical parameter, as too many epochs can lead to overfitting and poor generalisation. By contrast, too few epochs can lead to underfitting when the model cannot accurately capture the relationship between input and output variables. A simple and practical approach is to implement early stopping, which halts the training when the performance of the validation set no

longer improves. Early stopping is implemented by setting an arbitrarily large number of training epochs and stopping the training after a certain number of epochs with no progress in the validated loss. This parameter is named patience and has been set to 50 for most of the developing process.

ReLU is the preferred activation function when designing deeper neural networks as it is less complex and thus less computationally expensive than the implementation of tanh or sigmoid. However, as the AE works with a relatively shallow network, the convergence time did not play a significant role. Initial test simulations were made with ReLU and tanh as activation functions, and the two functions did not give notably different reconstruction losses. In addition, the tanh activation function was identified to perform the best in the previous project with ANNs using the same data. Thus, this thesis has mainly focused on using tanh for the simulations.

### 3.2.4 Fine-tuning of Hyperparameters

Considering vast network flexibility due to the high number of hyperparameters, finding the combination that provides the best model can be challenging. Keras offers automated grid searches like Grid Search CV and Randomised search CV, which automates the search process of hyperparameter combinations. These functions will give the model parameters with the lowest reconstruction loss. However, the lowest reconstruction loss for a DAE model will not give the optimal model. A low cost function is equivalent to a near-perfect reconstruction, meaning virtually no data compression has been performed, and the model simply maps input to output. The model's objective is to find a sensitivity balance of building a precise reconstruction while not simply copying the inputs. Trial and error is the most efficient way to find the best model in regard to the objective of this work. Several different model configurations have been explored. In the end 20 different bottleneck and mapping neuron combinations have been mapped out based on the MAE and MSE values. This mapping gave a better understanding of the model behaviour and how to identify the best networks.

## 3.3 Testing

The final modelling process is the testing, which consists of the evaluation and prediction. The independent testing set is treated as new unseen data and is used to give the final generalisation loss. The model can be said to generalise well if the predicted values, the generalisation loss, do not deviate from the values obtained from the training process, the reconstruction loss. Detailed tables containing the values for the cost functions for the 20 different simulated model configurations can be found in Appendix B.

The model should be able to make predictions on new instances. However, as no new data is available for this project yet, the same testing data set has been used to make predictions and visualise the models' denoising capability. The prediction has been visualised by plotting the time series data and the corresponding predictions in the same plot to compare the differences. The best model has been chosen through a visualised ranking based on the general denoising ability for all features. Once the best model was found, early stopping, mini-batch size, learning rate and activation function parameters were explored to find the final optimised model.

# Chapter IV

## *Results*

This chapter presents the final predictions from the model and explains the selection process. The best network is presented first, then the best individual features, fine-tuning of the model and the effect of changing the number of input features. Interpretations are based of comparison of the results.

## 4.1 Reconstruction loss

The reconstruction loss measures the difference between input and reconstructed output. However, it is also a value that indicates how much the inputs have been compressed. This is well illustrated when the reconstruction loss is plotted together with the different neuron configurations, as seen in Figure 4.1 and Figure 4.2. Both charts show the reconstruction loss, based on the validation loss and calculated using MSE and MAE, with respect to the number of neurons in the bottleneck- and mapping layer. One detail that must be noted is the scaling of the charts; the loss values are minimal and differ by only thousandths, which further emphasises how sensitive the models are.

Using MAE to calculate the reconstruction loss makes the values more uniformly distributed. The squared sum when using MSE has made it easier to distinguish the loss pillars. Studying the figures, it is hard to initially draw a limit between adequate feature extraction or a direct duplication of input. Nevertheless, the reconstruction loss is high for both charts when there are no mapping layers and data are only being compressed in the bottleneck layer. This loss value shows that the network is successfully compressing the samples into a $\Re^B$-dimensional space but precedes to fail in reconstructing the data. The reconstruction loss significantly decreases

when adding intermediate mapping layers before and after the bottleneck. This drop shows that the information flow drastically increases when mapping neurons are added to the network, and including them is essential to get enough information for detailed reconstructions. From Figure 4.2, it can be seen that when using ten neurons for mapping, the reconstruction is very low, especially with a bottleneck of 3, 4 and 5. It can be assumed that since the loss values are so low and do not differ significantly, these networks do not compress the input data for the network to learn any valuable patterns.

When considering the three-layered models, i.e. zero mapping neurons, the most significant value drop can be observed from 1 to 2 and from 2 to 3 bottleneck neurons with 6 or 8 mapping neurons. These models can be identified as sensitive enough to find underlying patterns while not copying the inputs. In theory, the best model should still have a minimum reconstruction loss if possible. It can be observed that by increasing bottleneck neurons from 3 until 5, the loss does not differ a lot. Thus, the two models picked out to be explored further are the bottleneck of 3 and mapping neurons of 6 (3-6) and 8 (3-8) [1].

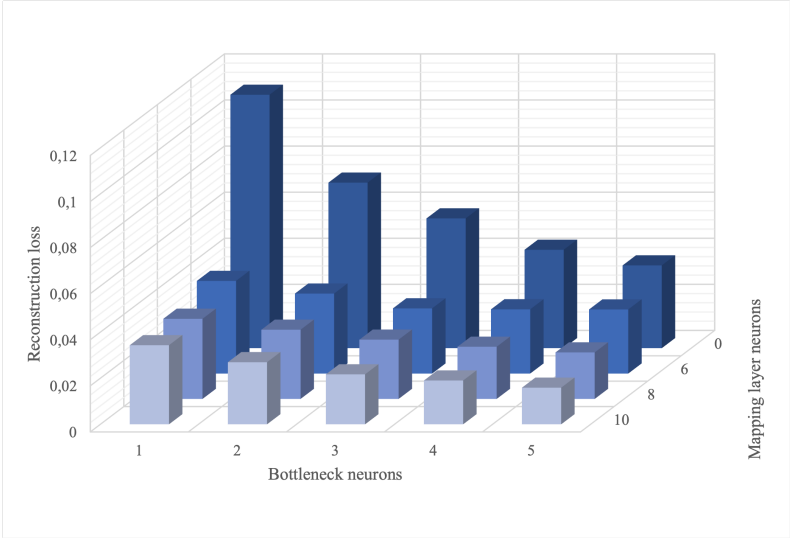



**Figure 4.1:** Mean Absolute Error (MAE) as reconstruction loss function for different model node configurations

[1] Moving forward, two numbers will note the networks: the first represents the number of bottleneck neurons followed by the number of mapping layer neurons

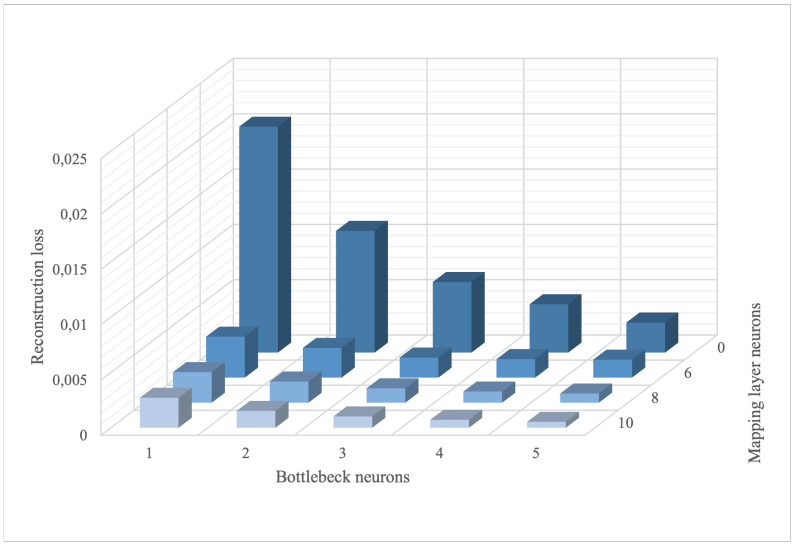
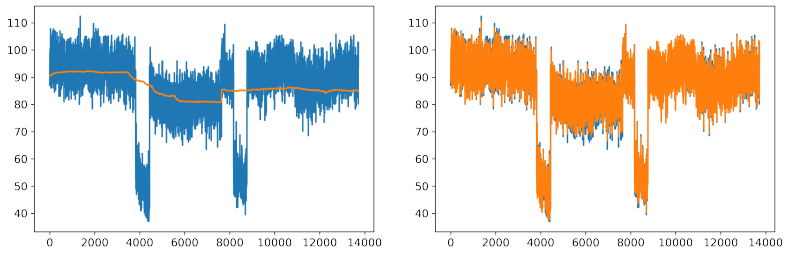



**Figure 4.2:** Mean Squared Error (MSE) as reconstruction loss function for different model node configurations

The preceding step is to evaluate the network using the hold-out testing set. The complete table with an overview of all losses and values can be found in Appendix B. As previously mentioned, the reconstruction loss vales alone are inadequate to measure the models' denoising capability. Therefore, the data before and after denoising have been plotted against each other to compare the denoising performance. It was not sufficient to only compare the two best models that have been picked out. Thus a comparison of all 20 models has been made for all 15 parameters. Only the most relevant results have been presented for the sake of the appendix section not being too lengthy.

A comparison between the model with the highest and lowest reconstruction loss can be seen on Figure 4.3. The blue plotting represents the data points before any modelling, and the orange parts are the predictions after being fed into the algorithm. Figure 4.3(a) is a network where all inputs get compressed into a 1-dimensional space and then reconstructed again. The result shows that barely any information has gotten through the bottleneck giving the model a poor reconstruction capability. Contrarily, Figure 4.3(b) is the visualisation of a network where the inputs are first compressed into a 10-dimensional space and then to 5-dimensions. This figure demonstrates how a low reconstruction loss yields a model where barely any information has been compressed and all data has been reconstructed near

perfectly.




**(a)** Network 1-0 modelling $P_{gen}$ **(b)** Network 5-10 modelling $P_{gen}$

**Figure 4.3:** Example of two networks with vast different denosing capabilities. The reconstruction loss is well reflected through the visualisation.

Following the model plotting, a ranking was carried out to find the best network. The ranking was made visually by comparing the dynamic behaviour of every plot from the same feature, corresponding to 20 images per data feature. The denoised output has been assessed by

- Capability of following the original input pattern and not creating additional deviations. This point includes how well the network can follow measured curve changes and the noise amplitudes.
- How the networks compare against each other. This comparison has been made by identifying the best-denoised plot of each feature, which has been the benchmark and assesses the best reconstruction capabilities.
- Studying certain focus regions within the different features. For example, to observe the network's ability to capture sudden declines or rises or how well it manages to replicate regions with little noise. A broad range of data represents the inputs; several input features do not contain any noise, while others suffer a significant amount of noise.

It was observed that individually or combined features would affect the network denoising ability and the results for the final model. However, since the goal of the network is to improve noise in sensor measurements, the emphasis is on noisy features such as $\Delta P$, $P_{gen}$, TOT, CCIT and $T_1$.

In order to find the network that performed the best, the denoised images were compared with each other, and the network images that were not up to par did not

receive points. A ranking selection can be found in Table 4.1, where only the best networks are presented. This table gives an overview of the level of performance on the various features for different networks. The ranking shows that left-out models with 1 or 2 neurons in the bottleneck layer did not perform the best, particularly the networks that did not have intermediate mapping layers. Despite the presented ranking table, networks 1-6, 1-8, 2-6, 2-8 and 2-10 also gave many reasonable denoised images. The networks with the best denoising ability based on features have 3, 4 or 5 neurons in the bottleneck layer. It can be seen that networks with mapping layers perform well, but networks 4-0 and 5-0 also created promising models. It should also be questioned whether this augmented number of neurons in the bottleneck is high enough for the network to start copying the inputs.

Based on Figure 4.1, Figure 4.2 and Table 4.1 an upper limit for reconstruction loss can be set for when the model starts finding good patterns within the data. Considering the network with the highest loss, 4-0, the limit can be set to approximately 0,0042 for both MSE and MAE as cost functions.

**Table 4.1:** The ranking table in order to identify the best models. The indication of a cross (x) is equivalent to a point, thus represents an acceptable result.

| **Network** | **3-10** | **3-6** | **3-8** | **4-0** | **4-10** | **4-6** | **4-8** | **5-0** | **5-6** | **5-8** |
|---|---|---|---|---|---|---|---|---|---|---|
| $CCIT$ | | x | x | x | x | x | x | x | x | x |
| $COP$ | x | x | x | | x | x | x | | x | x |
| $COT$ | x | x | x | x | x | x | x | x | x | x |
| $\Delta P$ | x | | x | | x | | x | | | |
| $EIT_1$ | x | x | x | x | x | x | x | x | x | x |
| $EIT_2$ | x | x | x | x | | x | x | x | x | x |
| $N$ | x | x | x | x | x | x | x | x | x | x |
| $P_{amb}$ | x | x | | | x | | | | | |
| $P_{gen}$ | | x | | x | | x | | x | x | |
| $RIT_a$ | x | x | | x | x | x | | x | | |
| $T_1$ | x | x | x | x | x | x | x | x | x | x |
| $TOT$ | x | x | x | x | | x | x | x | x | x |
| $T_{wIn}$ | x | x | x | x | x | x | x | x | x | x |
| $T_{wSatIn}$ | x | x | x | x | x | x | x | x | x | x |
| $V_{wSat}$ | x | x | x | x | x | x | x | x | x | x |

Ultimately, only the network with the best scoring and thus the highest number of acceptable images was deemed to have the best general denoising ability. The table shows that 3-6 is the model that gives the best predictions when considering all

the input features. It performs less adequately for $\Delta$P, a feature that typically gave good results when modelled with a higher number of mapping neurons. The 3-6 network was further explored with fine-tuning of hyperparameters. In addition, the classical elimination technique has been used to identify the best individual features, independent of other networks and parameters. These features are presented in the following sections.

## 4.2 Best network

An overview of the best model and its parameter configurations are presented in Table 4.2. This model has been used for hyperparameter fine-tuning. Keras offers a practical tool that automates the task of exploring the hyperparameter combinations, known as grid search. However, this approach will give the parameters that makes the model with the lowest reconstruction loss, which is not the goal of this task. Thus, the hyperparameter space needs to be explored manually and the results compared visually.

**Table 4.2:** The configuration of for the best network before hyperparameter fine-tuning.

| | |
|---|---|
| **Input nodes** | 15 |
| **Output nodes** | 15 |
| **Number of hidden layers** | 3 |
| **Hidden layer neurons** | 6 |
| **Bottlenecklayer neurons** | 3 |
| **Learning rate** | 0.1 |
| **Activation function** | tanh |
| **Optimizer** | Mini-batch GD |
| **Loss Algorithm** | MSE, MAE |
| **Batch size** | 42 |
| **Total training epochs** | 2 273 |

To get a better understanding of the denoising potential of the network the feature plots have been grouped and is presented in the following manner:

- High noise: $\Delta$P, $P_{gen}$, TOT, $T_1$, CCIT
- Low noise: N, COT, COP, $EIT_1$, $EIT_2$, $RIT_a$
- No noise: $P_{amb}$, $V_{wSat}$, $T_{wSatIn}$, $T_{wIn}$

The plots from the first group, "high noise", are the most interesting subjects to study, as these showcases the AEs real ability to improve noisy sensor data. These

are the inputs with the highest amount of noise and contains the data that has a significant effect for baseline network modelling. They are presented in Figure 4.4 ($\Delta$P ), Figure 4.5 ($P_{gen}$), Figure 4.6 (TOT), Figure 4.7 ($T_1$) and Figure 4.8 (CCIT).

The second group represents the features that contains measurement oscillations, but to a smaller degree. The refinement of these data could potentially have a impact on the final accuracy of a baseline ANN model. These images are presented in Figure 4.9 (N), Figure 4.10 (COT), Figure 4.11 (COP), Figure 4.12 ($EIT_1$), Figure 4.13 ($EIT_2$) and Figure 4.14 ($RIT_a$).

The final grouping is the plots with virtually no noise. These parameters could, in practice, be left out of the modelling process as no models could produce any satisfactory replications. On the flip side, leaving out parameters means that the network has less data for the basis of the algorithm. However, a complete set of models, reconstruction losses and a ranking have also been completed leaving out these input features. The results are further discussed later in section 4.5.

### 4.2.1 Parameters with high levels of noise

In the previous chapter, it was pointed out that $\Delta$P (Figure 4.4) was the parameter that did not perform as well as the other networks. This rating was primarily due to 3-6 networks' inability to measure well around time index 1 000 - 4 000, and other models giving superior results for this feature. On the contrary, $P_{gen}$ (Figure 4.5) is a feature with similar noise levels, but the model has been able to dampen nearly all the noise. $\Delta$P and $P_{gen}$ have similar noise patterns to TOT (Figure 4.6), which is the feature with the best denoising outcome among the three noisiest inputs. Although these features have a relatively similar noise pattern, the predictions made by networks 3-6 have demonstrated that the amount of removed noise differs a lot for each feature. For $T_1$ (Figure 4.7), hardly any noise has been removed, which cannot be observed using any other model either, due to lower levels of oscillations in measurements if compared with TOT. Finally, CCIT (Figure 4.8) is the parameter with the highest level of dynamics, which has been hard for the model to replicate perfectly but has managed to filter out the oscillations.

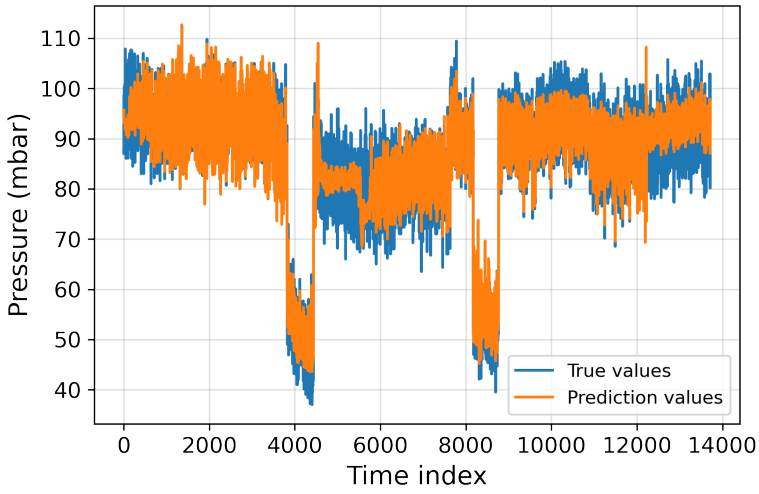



**Figure 4.4:** Best network prediction for $\Delta$P

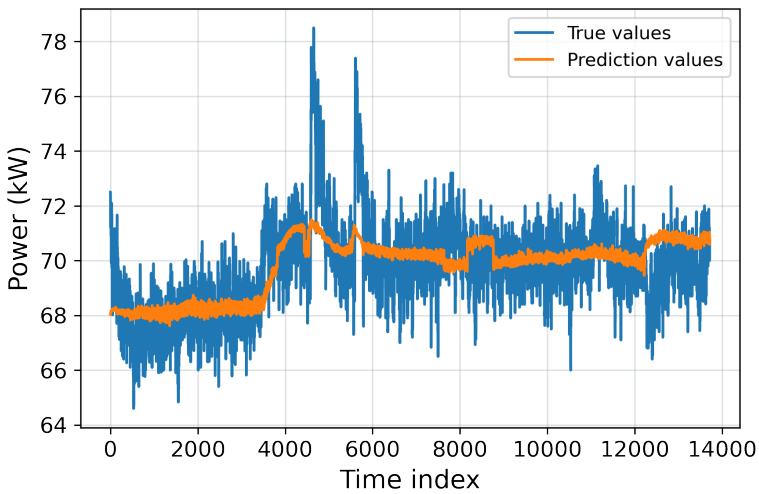



**Figure 4.5:** Best network prediction for $P_{gen}$

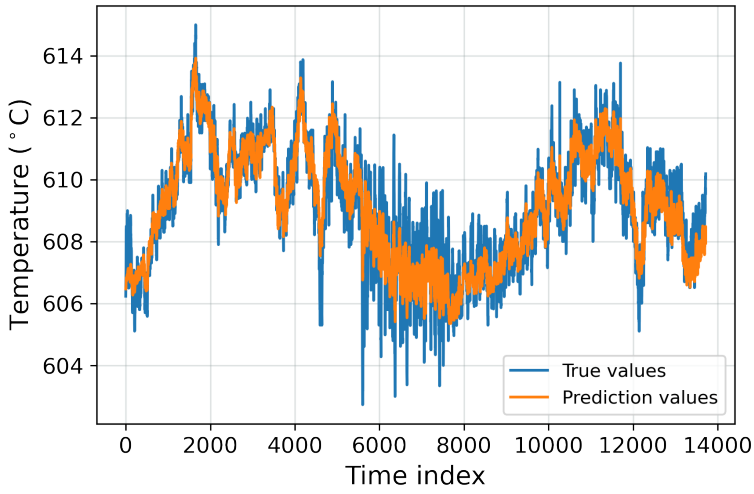



**Figure 4.6:** Best network prediction for TOT

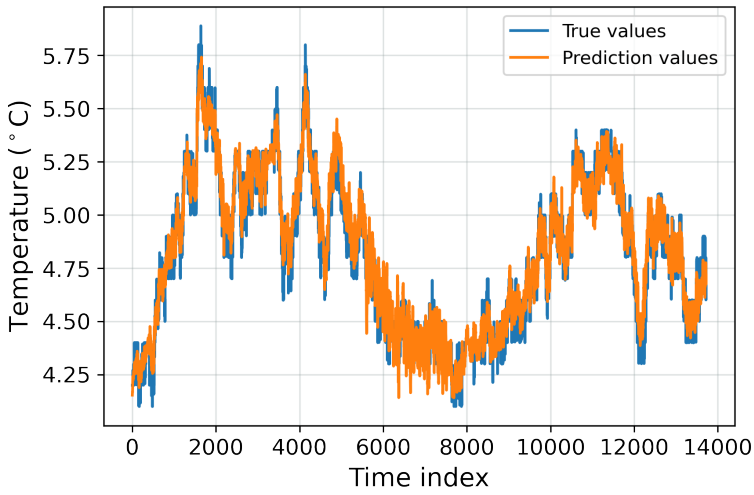



**Figure 4.7:** Best network prediction for $T_1$

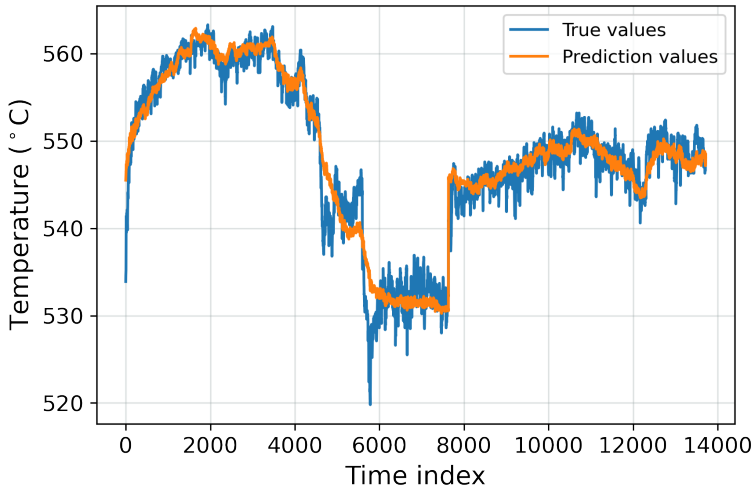



**Figure 4.8:** Best network prediction for CCIT

### 4.2.2 Parameters with low levels noise

This subsection presents the group of features with minor noise. For these features, it is no longer a big emphasis on the denoising capability but rather on how the predictions can replicate the measurement pattern. The pattern has been well captured for N, COT and COP seen on Figure 4.9, Figure 4.10 and Figure 4.11, respectively. They all behave fairly alike in both noise amplitude and measurement curve behaviour. Moreover, a less accurate resemblance between true and predicted values are seen for $EIT_1$ on Figure 4.12, $EIT_2$ on Figure 4.13 and $RIT_a$ on Figure 4.14. Although $EIT_1$ and $EIT_2$ are both measuring the temperature in the economiser, the only difference being the sensors' placement; they do produce different results, with $EIT_2$ being the feature that resembles the true values the most. Studying $RIT_a$, the predictions produce more noise than the true values, and the performance is poorer after the predictions. The issue with this parameter is that a high number of neurons in the mapping- and bottleneck layer is needed to make a good reconstruction, which might stipulate that the underlying data of this feature is not closely related to the others.

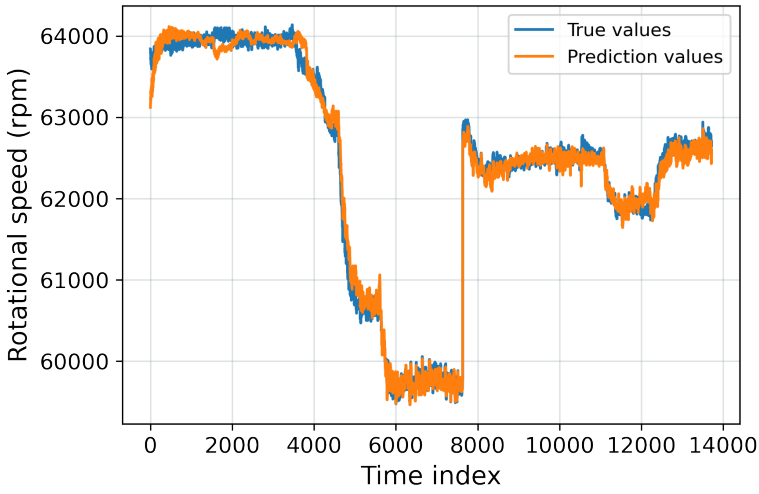



**Figure 4.9:** Best network prediction for N

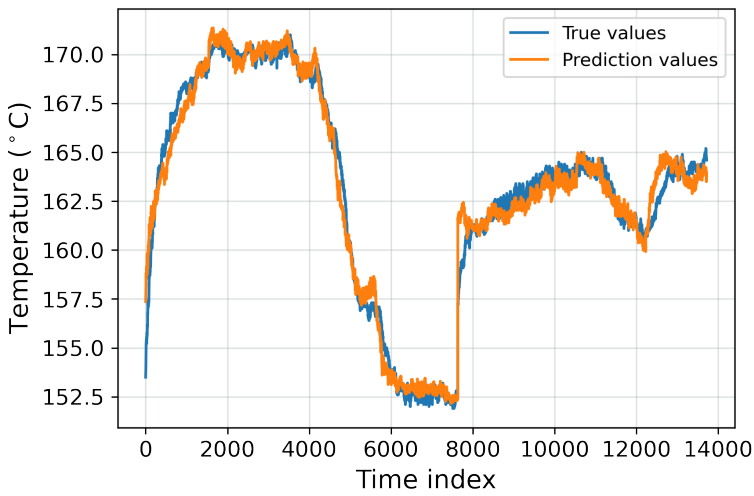



**Figure 4.10:** Best network prediction for COT

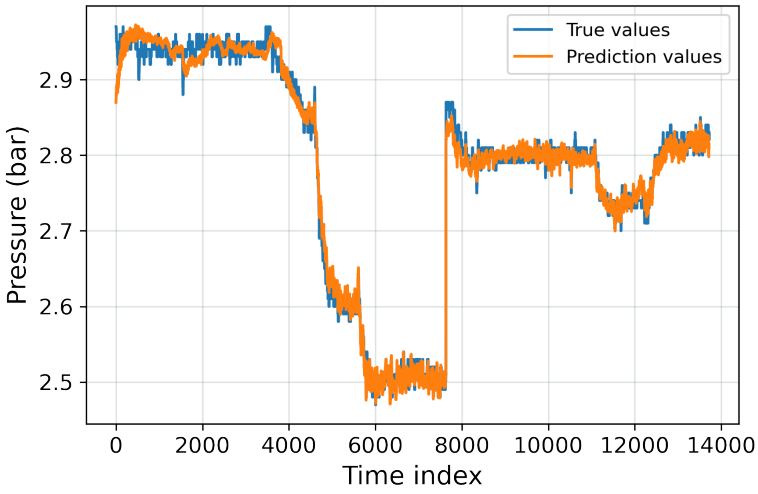



**Figure 4.11:** Best network prediction for COP

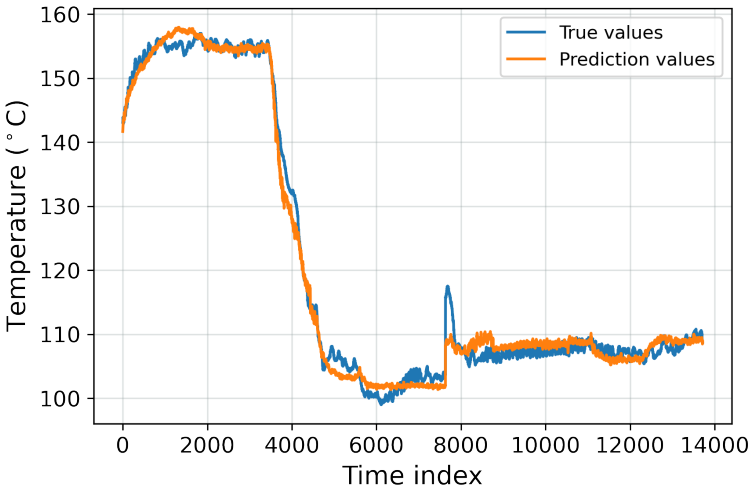



**Figure 4.12:** Best network prediction for $EIT_1$

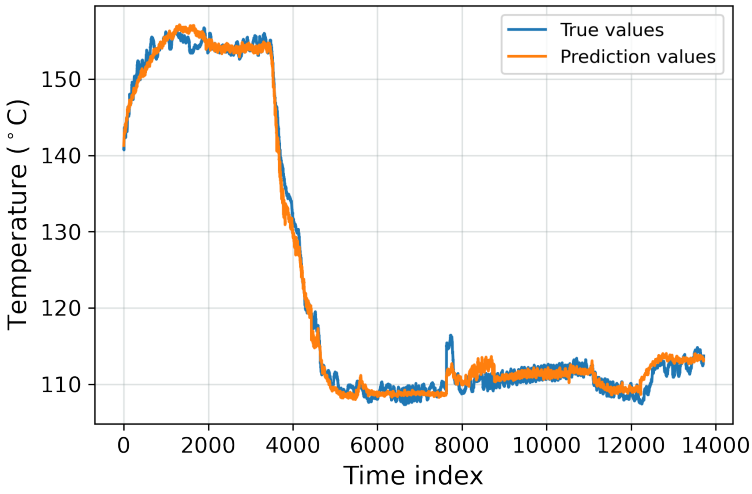



**Figure 4.13:** Best network prediction for $EIT_2$

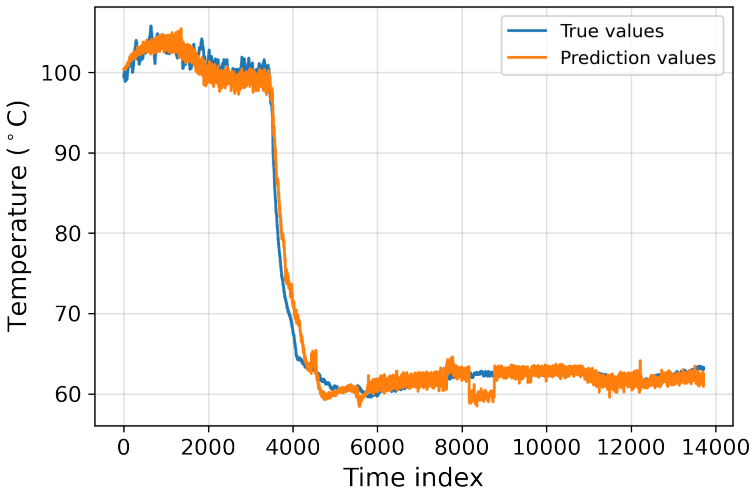



**Figure 4.14:** Best network prediction for $RIT_a$

### 4.2.3 Parameters with zero noise levels

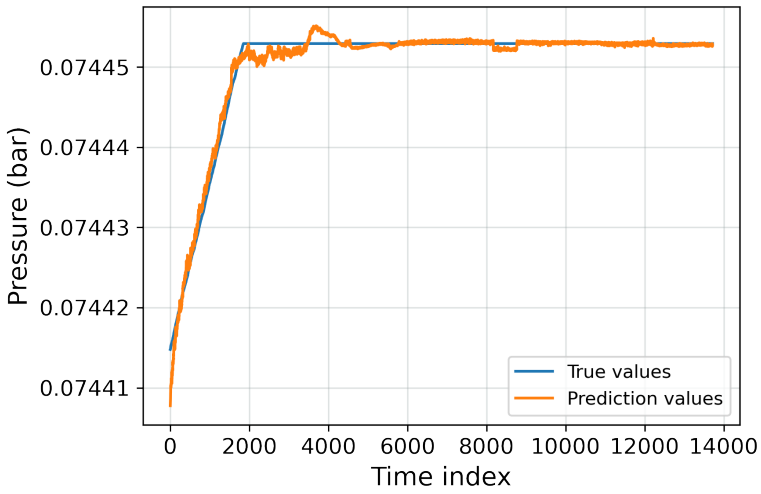



**Figure 4.15:** Best network prediction for $P_{amb}$

This final subsection presents plots that do not contain any noise, like $P_{amb}$, or the noise level is very low like $V_{wSat}$, and in reality, do to be filtered. They are still included for modelling, as networks can learn good features from quality data. Despite most networks' ability to duplicate these features, this network yields good results for $P_{amb}$ depicted on Figure 4.15. These predictions are neither perfect nor far from the best result presented in the next chapter. Furthermore, $V_{wSat}$ on Figure 4.16 does not show better predictions than true values, but the results are not counted as unsatisfactory. Conversely, $T_{wSatIn}$ and $T_{wIn}$, presented on Figure 4.17 and Figure 4.18 respectively, can be said to be the most inferior results from the 3-6 model. The predictions amplify the small oscillations, which might be due to the model not having enough data to recreate oscillations of this magnitude, highlighting the need for diverse data to make a model that can deliver all features.

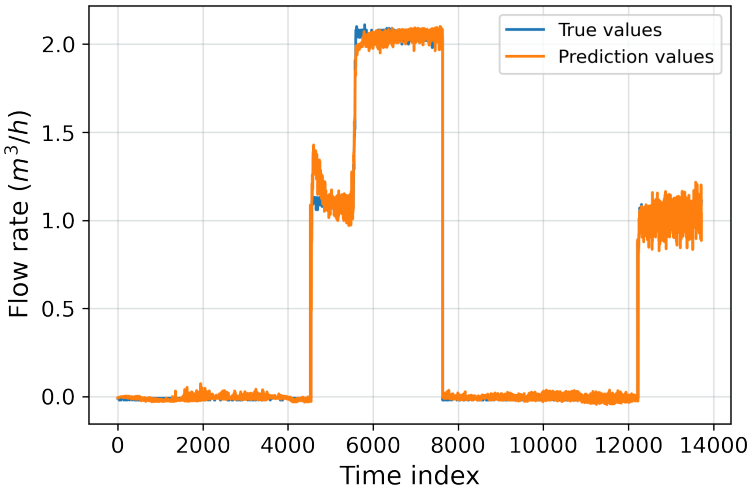



**Figure 4.16:** Best network prediction for $V_{wSat}$

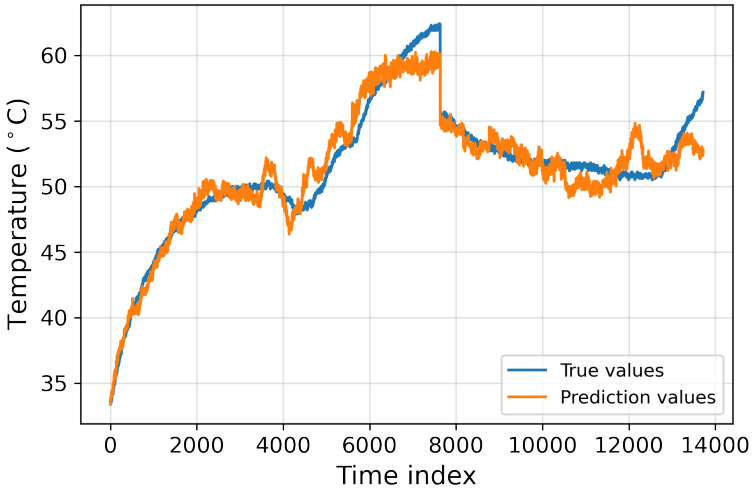



**Figure 4.17:** Best network prediction for $T_{wSatIn}$

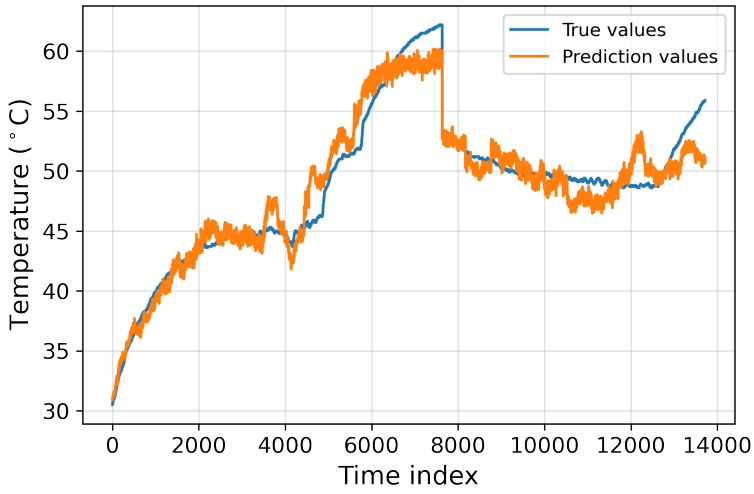
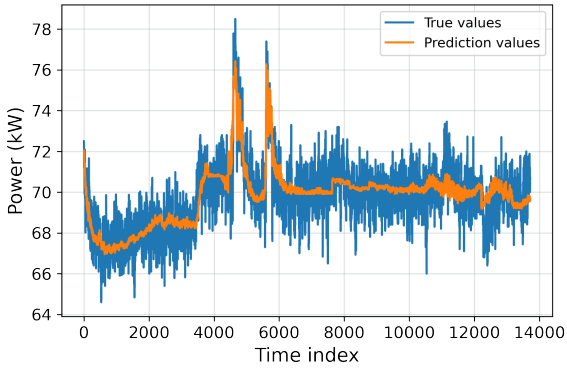



**Figure 4.18:** Best network prediction for $T_{wIn}$

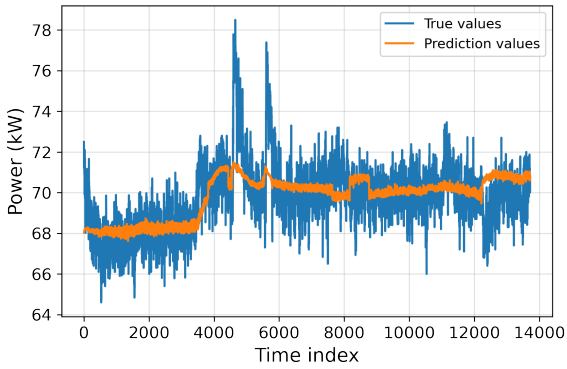
## 4.3 Best parameters

This section presents the individual best parameters and a side-by-side comparison with the same parameters from the best network. These plots have been set as the benchmark when comparing the networks against each other. It can be seen that over half of the plots are modelled with ten neurons in the mapping layer but with the number of bottleneck neurons varying from three to five. Nevertheless, it cannot be concluded that models with ten mapping neurons are superior as they perform poorer for the features that have not been presented. Nevertheless, it can already be presumed that with the number of measured features and the amount of data available, it is impossible to capture all the variations in the features adequately. This assumption is based on the fact that different network configurations have produced the best parameters. However, the reconstruction loss for a good model should be 0,0042 still holds.

Since the plots have been picked out from a visual examination and not, for example, by a loss function, biases are sure to arise. In addition, for a few of the features, there was not a remarkable difference between some of the best parameters, especially for features like $T_1$, $T_{wSatIn}$, $T_{wIn}$, $V_{wSat}$, CCIT, $EIT_1$, $EIT_2$ and TOT, which are also typical inputs features that are less noisy.




**(a)** Network 2-10 prediction for $P_{gen}$

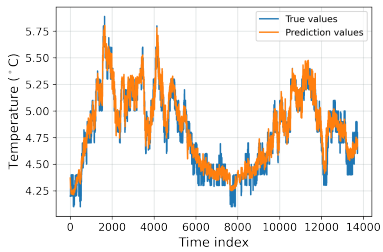



**(b)** Network 3-6 prediction for $P_{gen}$

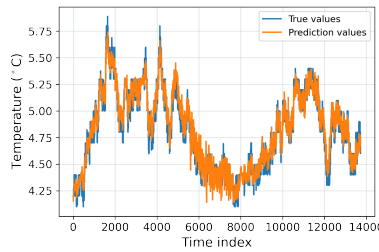
**Figure 4.19:** Network 2-10 has produced the best results for $P_{gen}$.

From $P_{gen}$ on Figure 4.19(a), it can be seen that model 2-10 has decreased the sensors' noisy measurements enormously while also following the pattern of the actual values. It is sensitive enough to be affected by significant disturbances. Additionally, it has managed to capture the two peaks around time index 5000, compared to model 3-6 on Figure 4.19(b). Nonetheless, both models have managed to reduce a substantial amount of noise. Finally, it can be questioned whether the inputs have been excessively denoised. Due to the lack of time, verifying the result functionalities in an ANN has not been possible. In practice, with clean data and thus better quality data, the performance of the ANN should be improved.

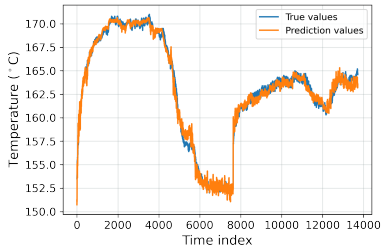
Network 3-10 has produced the model with plots of $T_1$ on Figure 4.20(a) , COT on Figure 4.20(c) and $RIT_a$ on Figure 4.20(e). As previously mentioned, these features, especially $T_1$, did not differ much from plots produced by other networks. This can be seen in comparing images from models 3-6 and 3-10 because there is not a peculiarly high amount of noise among these features, making them less interesting to improve further.



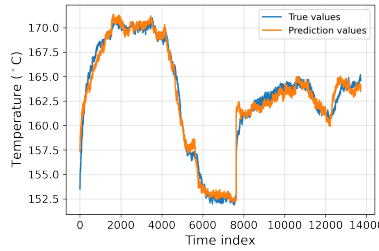

**(a)** Network 3-10 prediction for $T_1$



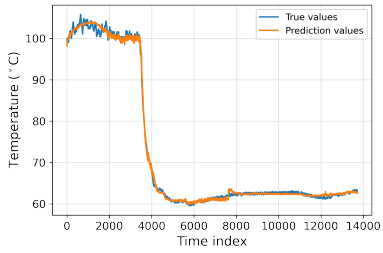

**(b)** Network 3-6 prediction for $T_1$



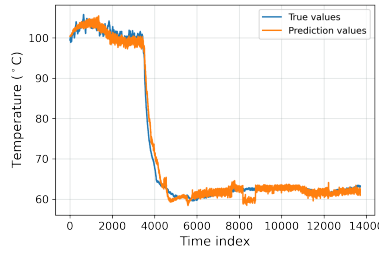

**(c)** Network 3-10 prediction for COT




**(d)** Network 3-6 prediction for COT




**(e)** Network 3-10 prediction for $RIT_a$

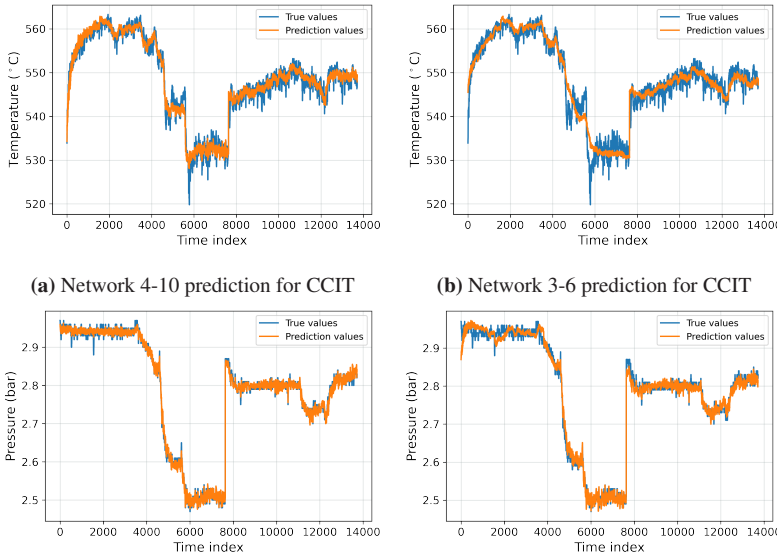



**(f)** Network 3-6 prediction for $RIT_a$

**Figure 4.20:** Network 3-10 has produced the best results for $T_1$, COT and $RIT_a$.

CCIT on Figure 4.21(a) and COP on Figure 4.21(c) have been modelled by network 4-10. Both these features have the same type of noise pattern and amplitude. This

level of detail can be difficult to capture for a network with a number of bottleneck neurons of less than four. Especially CCIT can be considered an excellent result, cutting out unnecessary noise and following the true predictions. COP adds a bit of noise but can generally be considered a satisfactory model.




**(a)** Network 4-10 prediction for CCIT

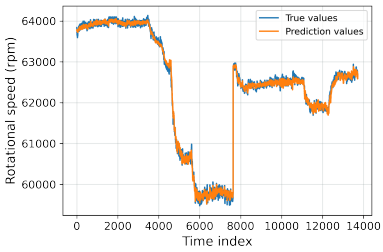
**(b)** Network 3-6 prediction for CCIT

**(c)** Network 4-10 prediction for COP

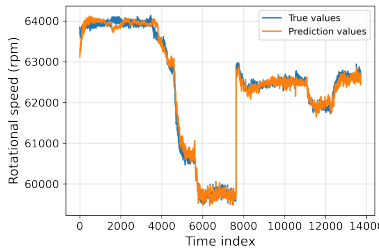
**(d)** Network 3-6 prediction for COP

**Figure 4.21:** Network 4-10 has produced the best results for CCIT and COP

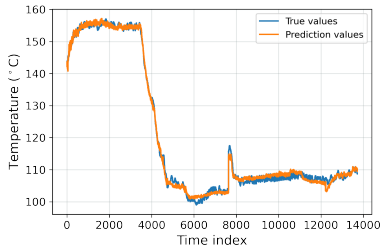
The 5-10 model is the network with the lowest reconstruction loss presented in this work. The reconstructions can be seen for N, $EIT_1$, $EIT_2$ and $T_{SatIn}$ corresponding to Figure 4.22(a), Figure 4.22(c), Figure 4.22(e) and Figure 4.22(g). Compared with network 3-6, it is not a big difference when comparing the two networks of N, $EIT_1$ or $EIT_2$. Moreover, there is a big discrepancy for $T_{wSatIn}$ which can be assumed is due to the network not capable of finding any reasonable underlying patterns to connect to the rest of the features. Earlier, these features have been categorised with "low noise", and it can be assumed that the 5-10 network is merely copying its inputs from outputs.



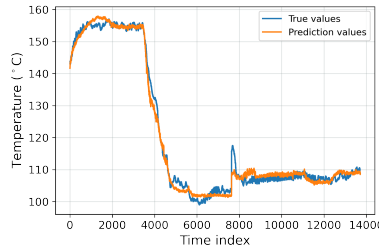

**(a)** Network 5-10 prediction for N



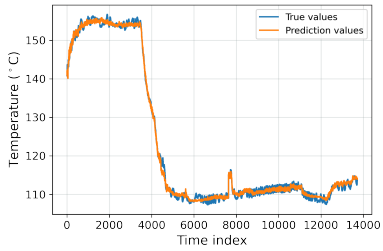

**(b)** Network 3-6 prediction for N



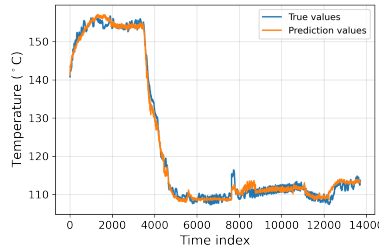

**(c)** Network 5-10 prediction for $EIT_1$



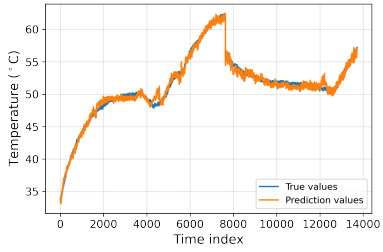

**(d)** Network 3-6 prediction for $EIT_1$



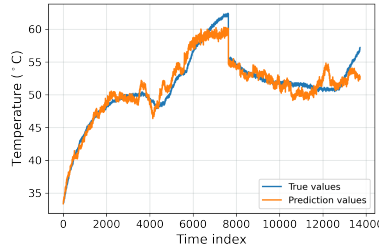

**(e)** Network 5-6 prediction for $EIT_2$




**(f)** Network 3-6 prediction for $EIT_2$



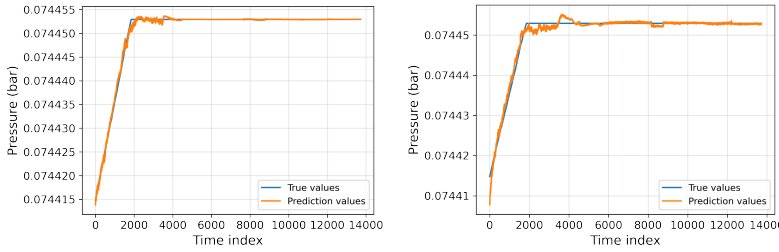

**(g)** Network 5-10 prediction for $T_{wSatIn}$




**(h)** Network 3-6 prediction for $T_{wSatIn}$

**Figure 4.22:** Network 5-10 has produced the best results for N, $EIT_1$, $EIT_2$ and $T_{wSatIn}$.

$P_{amb}$ is depicted on Figure 4.23(a) and is modelled by network 1-6. From the previous plots, it could be reasonable to assume that a network with many nodes should easily replicate a simple figure like this, but this result demonstrates the opposite. The use of complex models, with a large number of nodes in the bottleneck- and mapping layers, have a high level of non-linearity for data. When these models are applied to simple and linear behaviour, the prediction performance deteriorates due to overfitting. Thus, this simple replication ability is because these networks are taught to be insensitive to disturbances in the data.

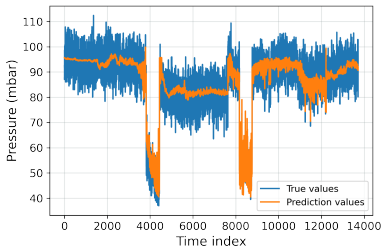



**(a)** Network 1-6 prediction for $P_{amb}$ **(b)** Network 3-6 prediction for $P_{amb}$

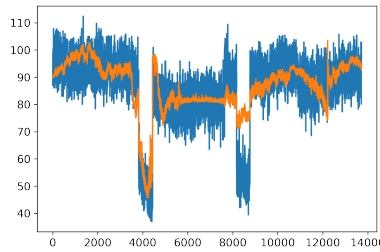
**Figure 4.23:** Network 1-6 has produced the best results for $P_{amb}$.

The model constructs oscillations when the original data becomes stable before time stamp 2 000. These oscillations might appear due to the algorithm learning from other features that the data regularly becomes noisy after a "change of state".

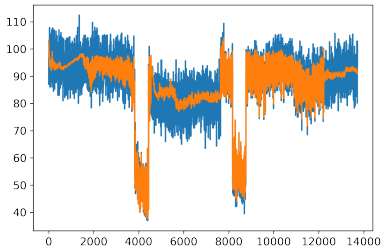
The best plot of $\Delta P$ is a result of network 2-6 as seen on Figure 4.24(a). This plot was chosen as it was the most consistent throughout the measured time period and did not contain any vast inconsistencies in the size of the "orange thread". It is essential for a feature like this not to use a too high number of bottleneck neurons. Additional plots have been presented in Figure 4.24 to show the variations when adding a bottleneck neuron or removing two neurons in the mapping layer. When going from two to three neurons in the bottleneck layer, much more information is featured in Figure 4.24(c), which once more highlights how important this parameter is. When two mapping neurons are removed to the best parameter model in Figure 4.24(b), the model has become slightly more insensitive to the noise. The best network model 3-6 is included on section 4.5, demonstrating how poorly it compares with the other features.



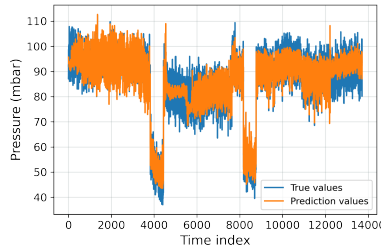

**(a)** Network 2-8 prediction for $\Delta$P




**(b)** Network 2-6 prediction for $\Delta$P

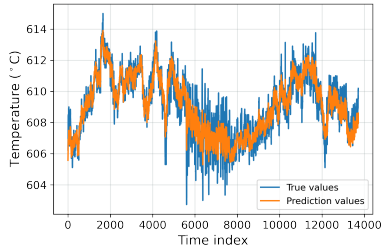



**(c)** Network 3-8 prediction for $\Delta$P

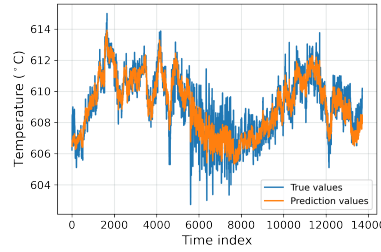



**(d)** Network 3-6 prediction for $\Delta$P

**Figure 4.24:** Network 2-8 has produced the best results for $\Delta$P. Supplementary plots are presented with different neurons in the bottleneck- and mapping giving an insight of the tuning process.




**(a)** Network 4-6 prediction for TOT



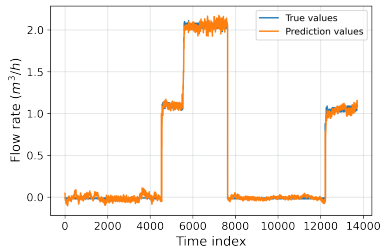

**(b)** Network 3-6 prediction for TOT

**Figure 4.25:** Network 4-6 has produced the best results for TOT

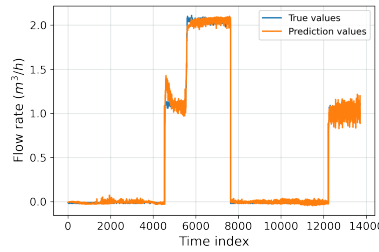
The best feature for TOT is challenging to select, as the model returned relatively similar results for a range of networks. However, all networks were in a bottleneck range of 2, 3 or 4 and mapping neurons equal to 6 or 8. Figure 4.25 compares

the 4-6 network (Figure 4.25(a)) with the 3-6 network (Figure 4.25(b)), and for the naked eye, they can almost be deemed to be the same plot.

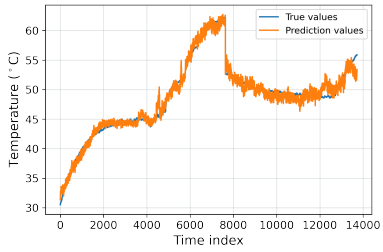
$V_{wSat}$ and $T_{wIn}$ on Figure 4.26 are the two final parameters and have been plotted by network 5-8, which is one of the models with the lowest reconstruction loss. Even though the reconstruction loss values cannot directly give the network the best denoising capability, the loss can indicate where features will perform efficiently. In this case, the features have low noise and variation levels, and it can be assumed that a high number of neurons in the bottleneck and mapping layers is the only network close to reconstructing this type of feature. It can be seen that the models create a lot of excessive, added and unnecessary noise. For $T_{wIn}$ the true values do not contain any oscillations, but the model still reproduces a plot containing noise. This added noise behaviour is similar to the reconstruction of $P_{amb}$, where the model also included noise that should not have been there. $V_{wSat}$ is the parameter with many smooth areas but has not been translated well when fed into the algorithm, and the explanation could be the same as for $T_{wIn}$.



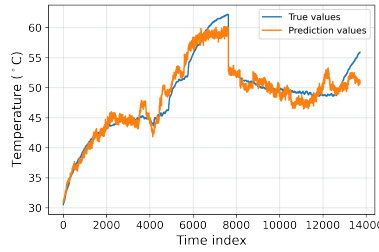

**(a)** Network 5-8 prediction for $V_{wSat}$




**(b)** Network 3-6 prediction for $V_{wSat}$




**(c)** Network 5-8 prediction for $T_{wIn}$




**(d)** Network 3-6 prediction for $T_{wIn}$

**Figure 4.26:** Network 5-8 has produced the best results for $V_{wSat}$ and $T_{wIn}$

Based on these plots of best parameters, nearly all best model parameters (3-6) do not deviate substantially from the benchmark plots. It has been demonstrated

how networks with a high number of neurons, in both mapping- and bottleneck layers, can successfully reproduce plots with great detail. The input contains a certain amount of noise. These networks can be restricted not to copying too much by changing the number of neurons in the mapping layer. Nevertheless, these networks are susceptible to any amplified noise produced and can give varying end results. Thus, choosing a less sensitive model that is not easily influenced by more significant disturbances can be advantageous. This balance is crucial when choosing a model for anomaly detection.

## 4.4 Fine-tuned model

This section is going to address the fine-tuning process of the model. After finding the network that can generalise the best for all features, it is fine-tuned by finding the hyperparameter combinations that give the best final results, namely the plots for this work. The fine-tuning has been performed manually, as test runs with fine-tuning grid search algorithms would run for an excessive amount of time and give the model the lowest reconstruction loss. A manual approach involves individually tuning each parameter in the code and visually comparing the results. The analysed hyperparameters are patience for early stopping, mini-batch size, learning rate and activation function.

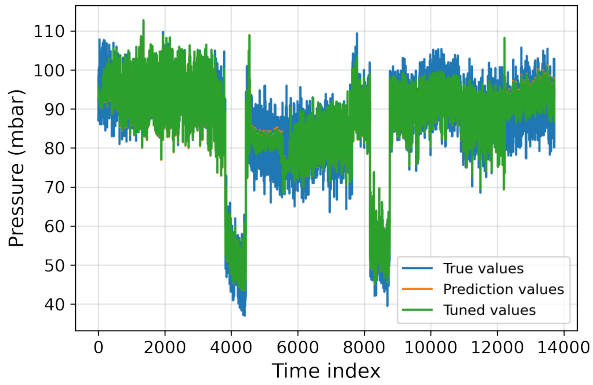
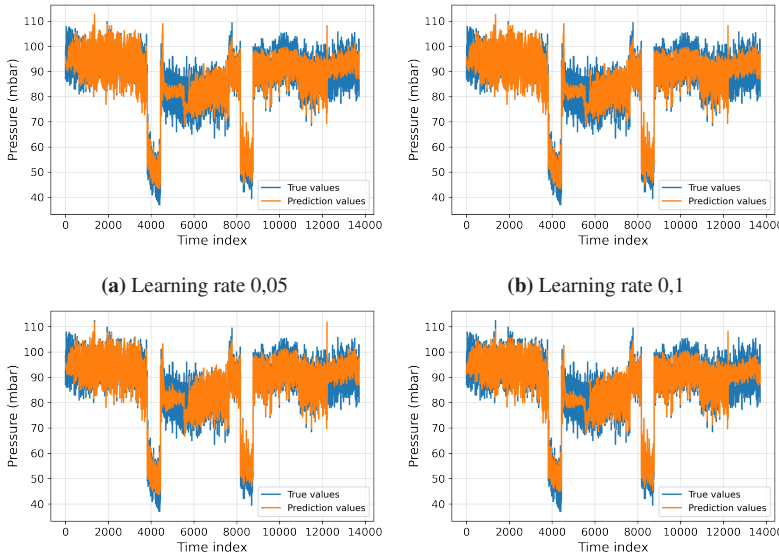



**Figure 4.27:** $P_{gen}$ depicted to compare the visual effect of changing the patience from 50 to 200

The early stopping parameter patience was set to a default of 50 during the modelling part, and the network has been evaluated using patience of 100 and 200. When prolonging the patience takes longer before the model lands on a final loss value.

The patience decides how many stable epochs the results need to be before giving a value and ending the training. From a numerical perspective, the reconstruction loss would improve by less than a thousandth when increasing the patience from 50 to 200. Figure 4.27 presents the model when patience is set to 200, specified by the green parts. It precisely overlaps the model with patience of 50, and it is safe to assume that extending the early stopping does not bring any benefits. In fact, it doubles the training epochs and thus the total simulation duration.

Mini-batch size has been altered to test whether simulation time or results could be improved. The mini-batch number was set to 1024, and the CPU was enabled, but no good results came from this run. The calculation time per step increased substantially with this setup, and the simulation ran beyond a reasonable time frame to continue. The second run was performed with a mini-batch 64 on GPU. The reconstruction loss was identical to when running on 42, except for an increase from 2273 to 3458 epochs before finishing. The last mini-batch size tested was 32, which shortened the simulation time. From the visual aspect, there were no deviations worth documenting.




**(a)** Learning rate 0,05

**(b)** Learning rate 0,1

**(c)** Learning rate 0,1

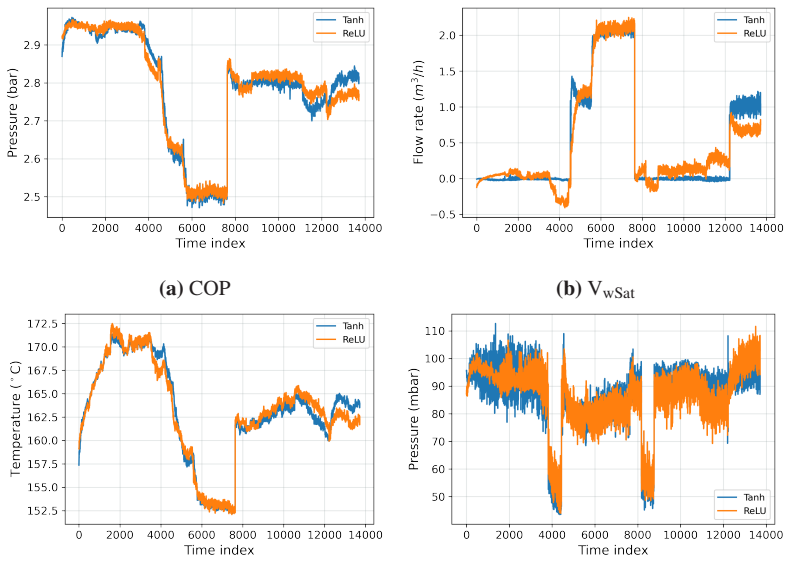
**(d)** Learning rate 1

**Figure 4.28:** $P_{gen}$ depicted to compare the visual effect of changing the learning rate between 0,1 and 1.

A wide range of learning rates has been analysed by exploring values around the

starting point of 0,1. The reconstruction loss would be very stable when increasing the learning rate from 0,1 to 5. However, the solution diverges when increasing the learning rate to 5 and upwards. When opting for a learning rate of 0,005, the values improved insignificantly and tripled the simulation duration. The finding from thus tuning was that the computational costs could be saved by raising the learning rate to 1.

The activation function has been changed from tanh to ReLU, as both are appropriate options for a regression problem. ReLU is considered a more complex function due to the vanishing gradient problem. However, vanishing gradients is only a problem when dealing with deeper neural networks; thus, for this problem, there should not be any significant setbacks by choosing one over the other. When switching to ReLU, the plots are drastically changed, which can be seen from a selection of comparisons when using tanh and ReLU as the activation function for network 3-6. From Figure 4.29, it is evident that by using ReLU, the network is behaving entirely different, and a separate tuning of this network should be completed. Due to the lack of time, the ReLU network has not been explored any further, but there is a potential to find an equivalent, if not better, network with this function if tuned correctly.




**(a)** COP **(b)** $V_{wSat}$

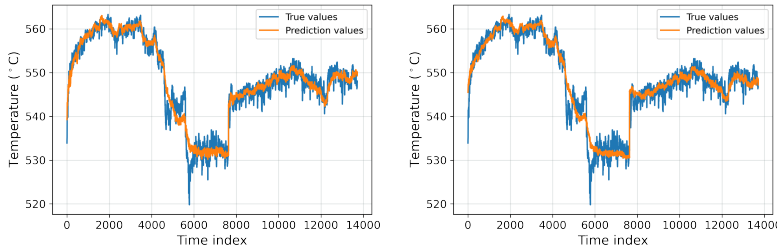
**(c)** COT **(d)** deltaP

**Figure 4.29:** Depictions of the difference when changing between tanh and ReLU.

## 4.5 Changed number of inputs

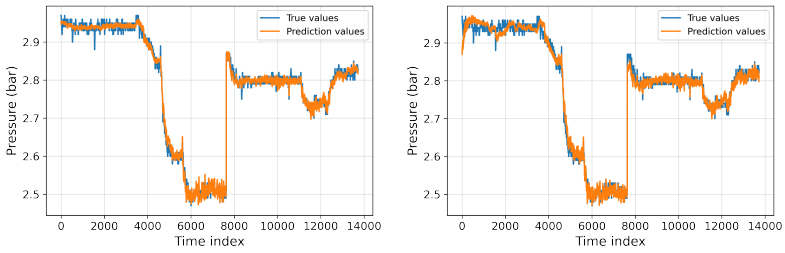
Previously it has been discussed whether including all data inputs to make a denoising model is beneficial or not, particularly the features that do not need to be improved. A full modelling process including reconstruction loss calculations and a ranking has been completed with leaving out $P_{amb}$, $T_{wSatIn}$, $T_{wIn}$ and $V_{wSat}$. This leaves a total of 11 features left to be modelled. The comparison for the 3-6 model for all features can be seen from Figure 4.30 to Figure 4.40.

Some slight improvement can be seen for $P_{gen}$ on Figure 4.38, against apparent performance decline for $\Delta P$ (Figure 4.34). There are minor details that can be pointed out. However, ultimately the network with 11 inputs instead of 15 performs at the same level, each with its minor setbacks. These results demonstrate that leaving out data does not contribute to improving the quality as the network with 11 inputs seems to perform better, especially for COP, COT, $EIT_1$, $EIT_2$ and $RIT_a$. Including these parameters also has a deeper impact than possible to point out only by studying the outputs numerically or visually.s



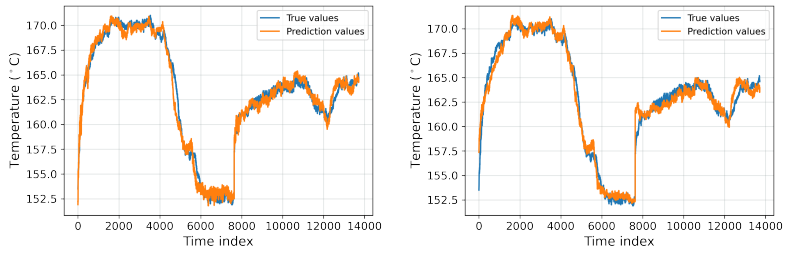

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.30:** CCIT when using different number of input features for modelling



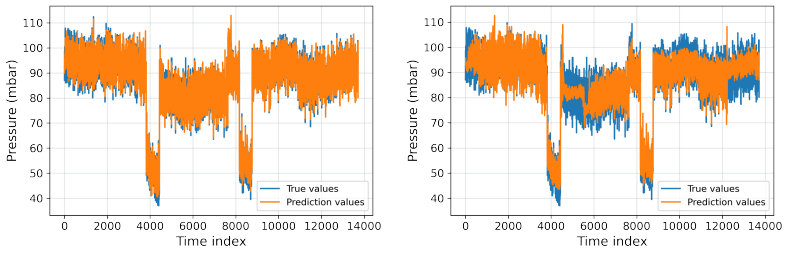

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.31:** COP when using different number of input features for modelling



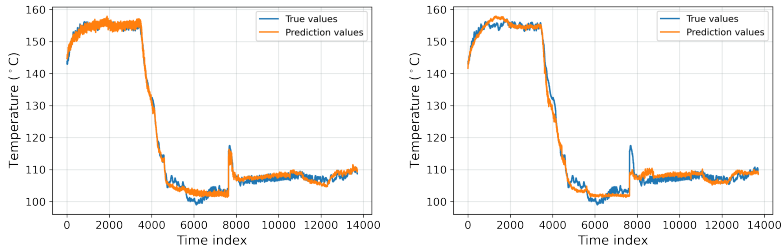

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.32:** COT when using different number of features for modelling



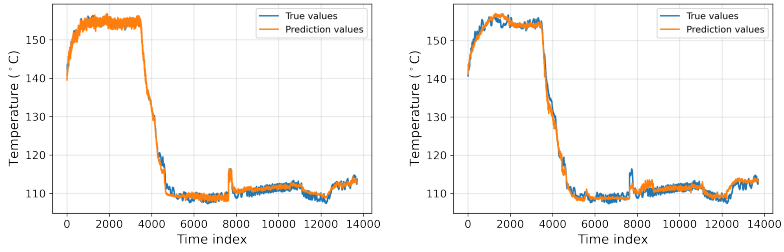

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.33:** deltaP when using different number of input features for modelling



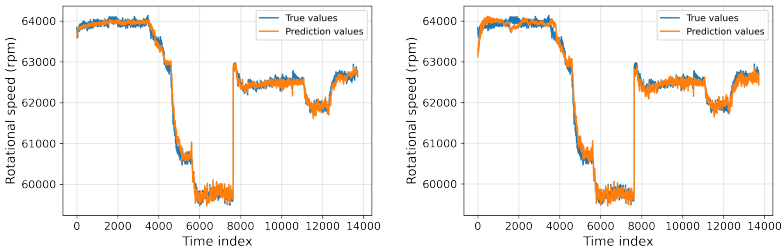

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.34:** $EIT_1$ when using different number of input features for modelling



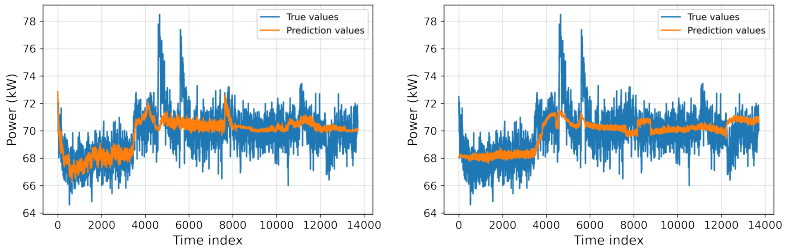

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.35:** $EIT_2$ when using different number of input features for modelling



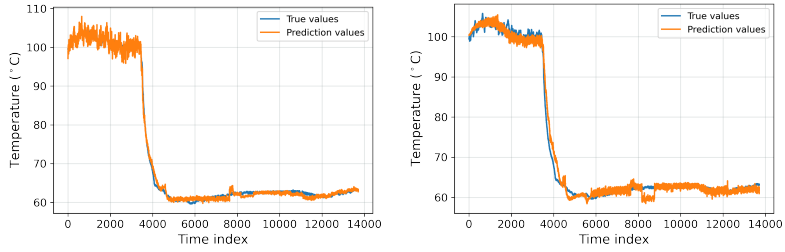

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.36:** N when using different number of input features for modelling



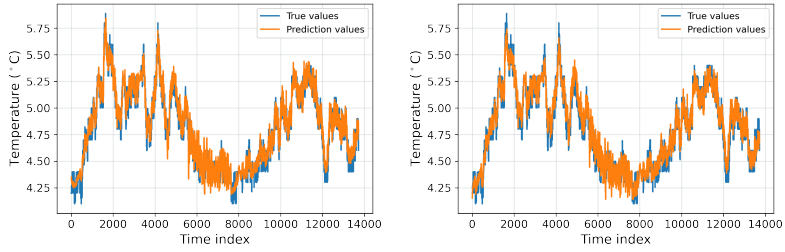

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.37:** $P_{gen}$ when using different number of input features for modelling



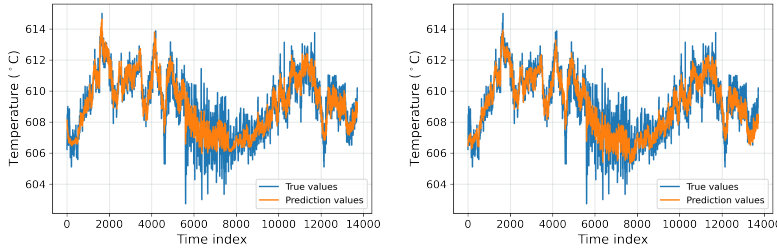

**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.38:** $RIT_a$ when using different number of input features for modelling




**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.39:** $T_1$ when using different number of input features for modelling




**(a)** 11 inputs **(b)** 15 inputs

**Figure 4.40:** TOT when using different number of input features for modelling

# Chapter V

## *Conclusion*

The main objective of this thesis has been to contribute to the research on the benefits of running a mHAT as a complementary technology for future energy production systems.

Machine learning techniques have been used to analyse the primary data obtained from sensor measurements of a mHAT rig at the VUB. A data-driven model has been successfully developed for predictive maintenance application, thus improving the reliability of promising low-emission machinery like the humidified mGT. Two of the primary utilisation areas for this autoencoder have been focused around:

- improving the measurements from the experimental test by filtering noisy inputs, which can be used as a preprocessing step for an ANN.
- creating a predictive monitoring model capable of detecting turbine failure by finding underlying variations in the residual levels.

The network with three neurons in the bottleneck layer and six neurons in each mapping layer has been identified, validated, and tested based on a trial and error method. This network can produce good denoising results across all provided parameter features. However, even with a network that yields satisfying results, it could not predict the best results for any single parameter due to the different identities of the features. Features such as $\Delta$P, $P_{gen}$, TOT and CCIT contains much noise and have been massively improved, others parameters like $V_{wsat}$, $T_{wsatIn}$, $T_{wIn}$ and $P_{amb}$ did not need to be denoised from the beginning. The best reconstruction of individual parameters has been identified, which indicates that more than one network should be chosen to find the best single feature.

Accordingly, tailored networks for each parameter are an excellent solution when the goal is to find the prediction with the lowest amount of noise. On the other

hand, for the case of creating a monitoring model, only one AE network should be selected as a basis for calculating residuals, which potentially is the best network that has been identified. This network efficiently balances how much information is reproduced through the appropriate number of neurons in the bottleneck and mapping layers. However, due to time limitations, the predictive model has not been completely developed; hence, the current study focused on the use of AE for preprocessing and denoising application, which was a demanding research task involving detailed analysis of different network setups and tuning work.

In conclusion, the developed models have successfully improved the sensor measurements. These results have proven the functionality of integrating machine learning models for enhancing data that can be used for predictive maintenance. The outcomes from this work can be utilised to enhance condition monitoring of the mHAT, improving the system reliability and promoting its further utilisation in future energy systems.

# Chapter VI

## *Future work*

While the overall results from this work are satisfactory, some tasks should still be followed up to make the best use of the data. The main reason is the lack of time combined with the possibilities to explore different machine learning techniques. As this is a relatively new field, more effective libraries, methods and algorithms are frequently presented, and there are still debates about best practices. One example is the size of the mini-batch, as discussed prior. In addition to that, data science competence to better compare results would be beneficial for future studies.

Due to the delicate work of finding the best hyperparameters for the denoising task, the time to develop a baseline model using AEs fell short. The setup for making the model has already been established. Therefore, it is only needed to test the sensitivity of the different networks against failure data, which would correspond to measurements when the gas turbine is experiencing a surge. When comparing regular to faulty operation, a residual can be calculated, and a certain threshold, or average network residual, can be established for early detection of compressor surge. Further exploring this method makes it possible to find the correlations between the features and the potential to leave out features, thus saving computational power or adding new sensor measurements that can strengthen the early detection model.

A minor task would be to explore further the denoising model's hyperparameter space and, in particular, the activation function ReLU. The same hyperparameters as the baseline ANN model created earlier at UiS have been set for the DAE model. While some hyperparameter assumptions have been correct for this network, others needed additional tuning. Moreover, the activation function was the last parameter to be fine-tuned; thus, time fell short when ReLU results behaved entirely differently from the tested tanh. An entire reconstruction error analysis should be performed together with a visual comparison to find the best networks

and parameters for ReLU.

The last suggestion for improvement is the development of an algorithm in order to compare the visual results better. The method that has been used for this work is sufficient. However, the possibility for faults and bias is higher for visual comparison. When developing an algorithm or function, it should distinguish between the different types of noise models and set a threshold for deviation from the actual data, which could, for example, be based on the reconstruction error. Such a function would contribute to validating the quality and reliability of the chosen final model.

# *References*

[1] De Paepe, W., Montero Carrero, M., Bram, S., and Contino, F., 2015, "T100 Micro Gas Turbine Converted to Full Humid Air Operation: A Thermodynamic Performance Analysis," American Society of Mechanical Engineers Digital Collection, doi:10.1115/GT2015-43267, https://manufacturingscience.asmedigitalcollection.asme.org/GT/proceedings/GT2015/56673/V003T06A015/236735

[2] Pramoditha, R., 2021, "The Concept of Artificial Neurons (Perceptrons) in Neural Networks," https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[3] Chavhan, A., 2019, "How to determine Overfitting and Underfitting?" https://medium.com/@ankitchavhan212/how-to-determine-overfitting-and-underfitting-eab0c52099b6

[4] "COP26 Goals," https://ukcop26.org/cop26-goals/

[5] "REPowerEU: affordable, secure and sustainable energy for Europe," https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal/repowereu-affordable-secure-and-sustainable-energy-europe_en

[6] Parente, J., Traverso, A., and Massardo, A. F., 2009, "Micro Humid Air Cycle: Part A — Thermodynamic and Technical Aspects," American Society of Mechanical Engineers Digital Collection, pp. 221–229, doi: 10.1115/GT2003-38326, https://thermalscienceapplication.asmedigitalcollection.asme.org/GT/proceedings/GT2003/3686X/221/298394

[7] Dinçer, I., Rosen, M. A., and Ahmadi, P., 2017, *Optimization of Energy Systems*.

[8] Pilavachi, P. A., 2002, "Mini- and micro-gas turbines for combined heat and power," Applied Thermal Engineering, **22**(18), pp. 2003–2014.

[9] Montero Carrero, M., Rodríguez Sánchez, I., De Paepe, W., Parente, A., and Contino, F., 2019, "Is There a Future for Small-Scale Cogeneration in Europe? Economic and Policy Analysis of the Internal Combustion Engine, Micro Gas Turbine and Micro Humid Air Turbine Cycles," Energies, **12**(3), p. 413, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[10] Commission, E. and Energy, D.-G. f., 2021, *EU energy in figures : statistical pocketbook 2021*, Publications Office.

[11] Altmann, M., Brenninkmeijer, A., Lanoix, J.-C., Ellison, D., Crisan, A., Hugyecz, A., Koreneff, G., and Hänninen, S., 2009, "Decentralized Energy Systems, European Parliament's Committee on Industry, Research and Energy," https://www.europarl.europa.eu/document/activities/cont/201106/20110629ATT22897/20110629ATT22897EN.pdf

[12] Lee, J. J., Jeon, M. S., and Kim, T. S., 2010, "The influence of water and steam injection on the performance of a recuperated cycle microturbine for combined heat and power application," Applied Energy, **87**(4), pp. 1307–1316.

[13] Nikpey, H., Mansouri Majoumerd, M., Assadi, M., and Breuhaus, P., 2014, "Thermodynamic Analysis of Innovative Micro Gas Turbine Cycles," American Society of Mechanical Engineers Digital Collection, doi:10.1115/GT2014-26917, https://asmedigitalcollection.asme.org/GT/proceedings/GT2014/45653/V03AT07A029/241950

[14] De Paepe, W., Carrerro, M. M., Bram, S., Parente, A., and Contino, F., 2017, "Advanced Humidified Gas Turbine Cycle Concepts Applied to Micro Gas Turbine Applications for Optimal Waste Heat Recovery," Energy Procedia, **105**, pp. 1712–1718.

[15] Geron, A., 2022, "Machine Learning Notebooks," original-date: 2019-01-08T03:49:07Z, https://github.com/ageron/handson-ml2

[16] Geron, A., 2019, *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed., O'Reilly Media, Inc, Canada.

[17] Montero Carrero, M., De Paepe, W., Bram, S., Musin, F., Parente, A., and Contino, F., 2016, "Humidified micro gas turbines for domestic users: An economic and primary energy savings analysis," Energy, **117**, pp. 429–438.

[18] Jonsson, M. and Yan, J., 2005, "Humidified gas turbines—a review of proposed and implemented cycles," Energy, **30**(7), pp. 1013–1078.

[19] Rao, A. D., 1989, "Process for producing power," https://patents.google.com/patent/US4829763A/en

[20] De Paepe, W., Paepe, W. D., Delattin, F., Bram, S., and Ruyck, J. D., 2013, "Water injection in a micro gas turbine – Assessment of the performance using a black box method," Applied Energy, **112**, pp. 1291–1302.

[21] Montero Carrero, M., De Paepe, W., Parente, A., and Contino, F., 2016, "T100 mGT converted into mHAT for domestic applications: Economic analysis based on hourly demand," Applied Energy, **164**, pp. 1019–1027.

[22] Parente, J., Traverso, A., and Massardo, A. F., 2009, "Micro Humid Air Cycle: Part B — Thermoeconomic Analysis," American Society of Mechanical Engineers Digital Collection, pp. 231–239, doi:10.1115/GT2003-38328, https://turbomachinery.asmedigitalcollection.asme.org/GT/proceedings/GT2003/3686X/231/298404

[23] Zhang, S. and Xiao, Y., 2006, "Steady-State Off-Design Thermodynamic Performance Analysis of a Humid Air Turbine Based on a Micro Turbine," doi:10.1115/GT2006-90335.

[24] Nakano, S., Kishibe, T., Araki, H., Yagi, M., Tsubouchi, K., Ichinose, M., Hayasaka, Y., Sasaki, M., Inoue, T., Yamaguchi, K., and Shiraiwa, H., 2007, "Development of a 150kW Microturbine System Which Applies the Humid Air Turbine Cycle," doi:10.1115/GT2007-28192.

[25] Wei, C. and Zang, S., 2013, "Experimental investigation on the off-design performance of a small-sized humid air turbine cycle," Applied Thermal Engineering, **51**(1), pp. 166–176.

[26] De Paepe, W., Carrero, M. M., Bram, S., Parente, A., and Contino, F., 2014, "Experimental Characterization of a T100 Micro Gas Turbine Converted to Full Humid Air Operation," Energy Procedia, **61**, pp. 2083–2088.

[27] De Paepe, W., Contino, F., Delattin, F., Bram, S., and De Ruyck, J., 2014, "New concept of spray saturation tower for micro Humid Air Turbine applications," Applied Energy, **130**, pp. 723–737.

[28] De Paepe, W., Montero Carrero, M., Bram, S., and Contino, F., 2014, "T100 Micro Gas Turbine Converted to Full Humid Air Operation: Test Rig Evaluation," American Society of Mechanical Engineers Digital Collection, doi:10.1115/GT2014-26123, https://asmedigitalcollection.asme.org/GT/proceedings/GT2014/45653/V03AT07A020/241948

[29] Montero Carrero, M., De Paepe, W., Magnusson, J., Parente, A., Bram, S., and Contino, F., "Experimental characterisation of a micro Humid Air Turbine: assessment of the thermodynamic performance," Applied Thermal Engineering, **118**, pp. 796–806.

[30] Bathie, W. W., 1996, *Fundamentals of gas turbines*, 2nd ed., Wiley, New York.

[31] Jauregui Correa, J. C. A. and Lozano Guzman, A. A., 2020, "Chapter Eight - Condition monitoring," *Mechanical Vibrations and Condition Monitoring*, J. C. A. Jauregui Correa and A. A. Lozano Guzman, eds., Academic Press, pp. 147–168, doi:10.1016/B978-0-12-819796-7.00008-1, https://www.sciencedirect.com/science/article/pii/B9780128197967000081

[32] Malik, H., Fatema, N., and Iqbal, A., 2021, "Chapter 1 - Advances in Machine Learning and Data Analytics," *Intelligent Data-Analytics for Condition Monitoring*, H. Malik, N. Fatema, and A. Iqbal, eds., Academic Press, pp. 3–29, doi:10.1016/B978-0-323-85510-5.00001-6, https://www.sciencedirect.com/science/article/pii/B9780323855105000016

[33] Sina Tayarani-Bathaie, S., Sadough Vanini, Z. N., and Khorasani, K., 2014, "Dynamic neural network-based fault diagnosis of gas turbine engines," Neurocomputing, **125**, pp. 153–165.

[34] Brighenti, G. D., Orts-Gonzalez, P. L., Sanchez-de Leon, L., and Zachos, P. K., 2017, "Design Point Performance and Optimization of Humid Air Turbine Power Plants," Applied Sciences, **7**(4), p. 413, Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[35] Tidriri, K., Chatti, N., Verron, S., and Tiplica, T., 2016, "Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges," Annual Reviews in Control, **42**, pp. 63–81.

[36] Tahan, M., Tsoutsanis, E., Muhammad, M., and Abdul Karim, Z. A., 2017, "Performance-based health monitoring, diagnostics and prognostics for

condition-based maintenance of gas turbines: A review," Applied Energy, **198**, pp. 122–144.

[37] Asgari, H., Chen, X., Menhaj, M. B., and Sainudiin, R., 2013, "Artificial Neural Network–Based System Identification for a Single-Shaft Gas Turbine," Journal of Engineering for Gas Turbines and Power, **135**(9).

[38] Barad, S. G., P.v., R., R.k., G., and G., K., 2012, "Neural network approach for a combined performance and mechanical health monitoring of a gas turbine engine," Mechanical Systems and Signal Processing, **27**, pp. 729–742.

[39] Sampath, S. and Singh, R., 2004, "An Integrated Fault Diagnostics Model Using Genetic Algorithm and Neural Networks," Journal of Engineering for Gas Turbines and Power, **128**(1), pp. 49–56.

[40] Yoon, J. E., Lee, J. J., Kim, T. S., and Sohn, J. L., 2008, "Analysis of performance deterioration of a micro gas turbine and the use of neural network for predicting deteriorated component characteristics," Journal of Mechanical Science and Technology, **22**(12), p. 2516.

[41] El Naqa, I. and Murphy, M. J., 2015, "What Is Machine Learning?" *Machine Learning in Radiation Oncology: Theory and Applications*, I. El Naqa, R. Li, and M. J. Murphy, eds., Springer International Publishing, Cham, pp. 3–11, doi:10.1007/978-3-319-18305-3_1, https://doi.org/10.1007/978-3-319-18305-3_1

[42] McCulloch, W. S. and Pitts, W., 1990, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biology, **52**(1), pp. 99–115.

[43] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A., 2008, "Extracting and composing robust features with denoising autoencoders," *Proceedings of the 25th international conference on Machine learning - ICML '08*, ACM Press, Helsinki, Finland, pp. 1096–1103, doi:10.1145/1390156.1390294, http://portal.acm.org/citation.cfm?doid=1390156.1390294

[44] Shao, H., Jiang, H., Zhao, H., and Wang, F., 2017, "A novel deep autoencoder feature learning method for rotating machinery fault diagnosis," Mechanical Systems and Signal Processing, **95**, pp. 187–204.

[45] Lu, C., Wang, Z.-Y., Qin, W.-L., and Ma, J., 2017, "Fault diagnosis of rotary machinery components using a stacked denoising autoencoder-based health state identification," Signal Processing, **130**, pp. 377–388.

[46] Goodfellow, I., Bengio, Y., and Courville, A., 2016, *Deep Learning*, MIT Press, Google-Books-ID: omivDQAAQBAJ.

# Appendix - A

## *Unprocessed features*

This section presents the plots of the original features that has been used for modelling. The only modifications done to these plots is the removal of significant outliers.

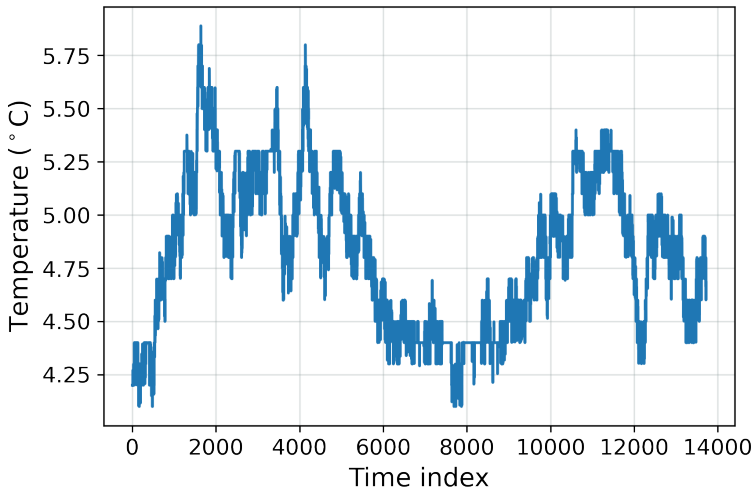



**Figure A.1:** $T_1$ before denosing

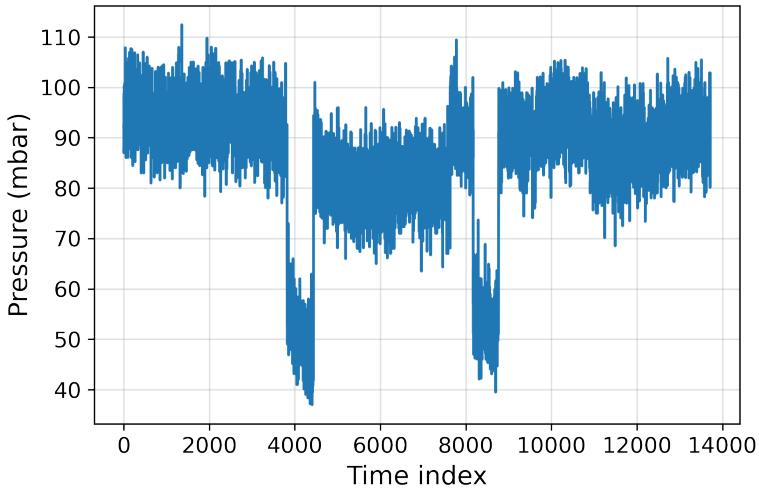



**Figure A.2:** $\Delta$P before denosing

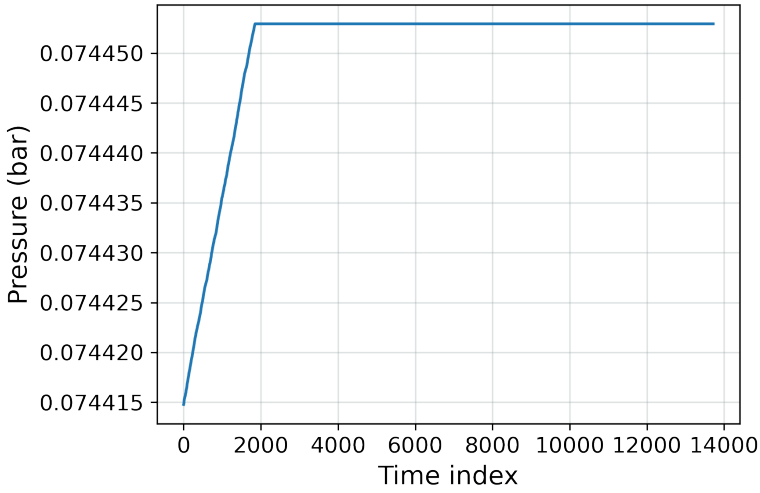



**Figure A.3:** $P_{amb}$ before denosing

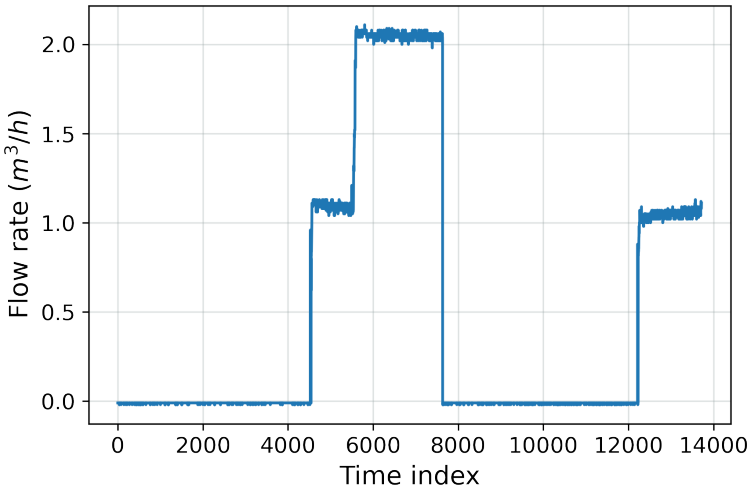



**Figure A.4:** $V_{wsat}$ before denosing

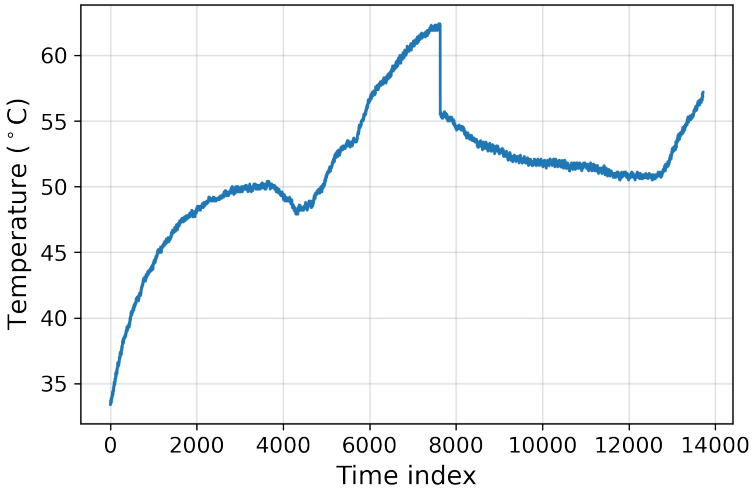



**Figure A.5:** $T_{wsatin}$ before denosing

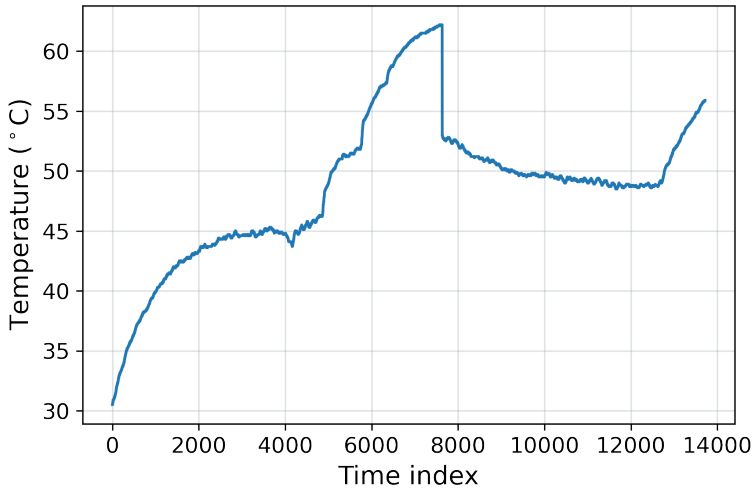



**Figure A.6:** $T_{wIn}$ before denosing

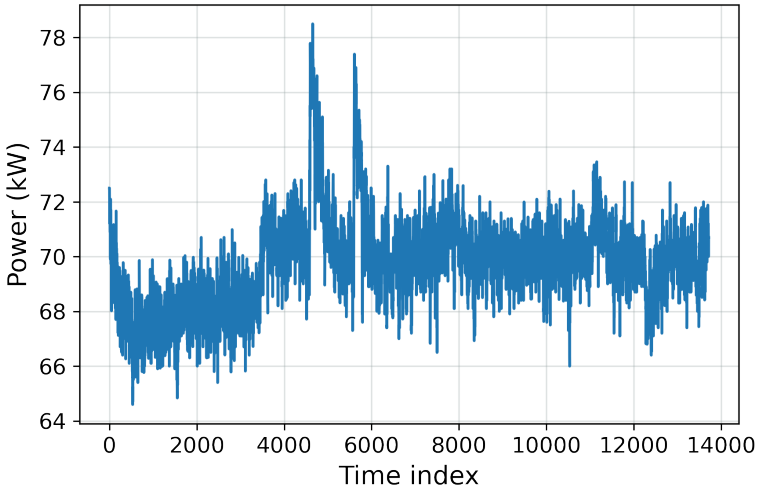



**Figure A.7:** P before denosing

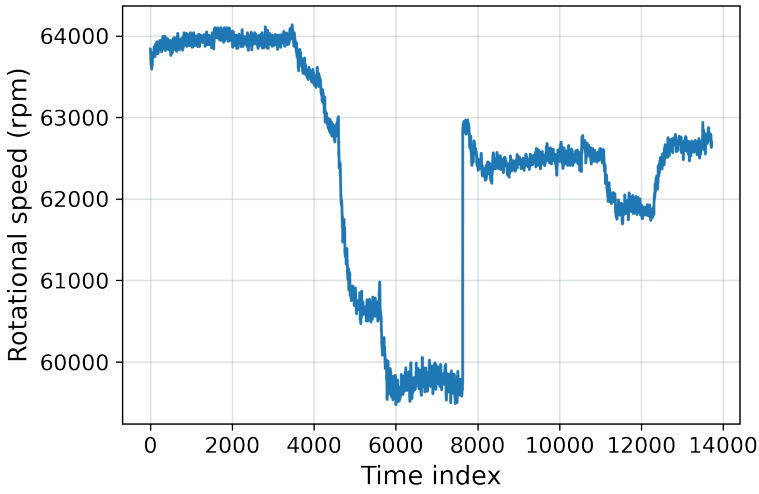



**Figure A.8:** N before denosing

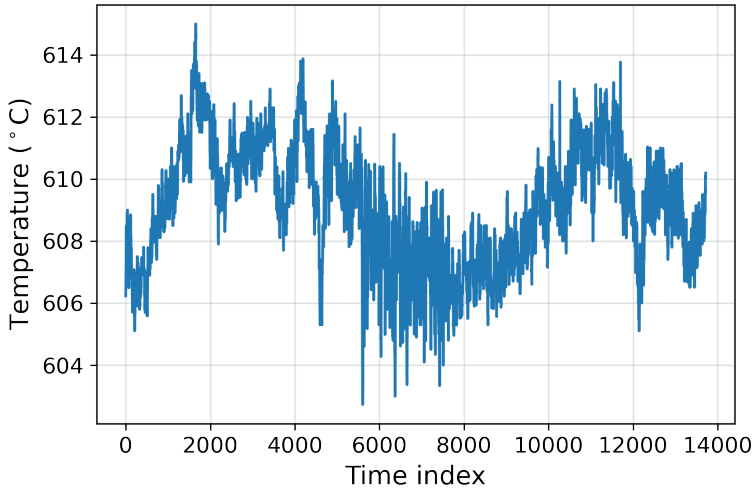



**Figure A.9:** TOT before denosing

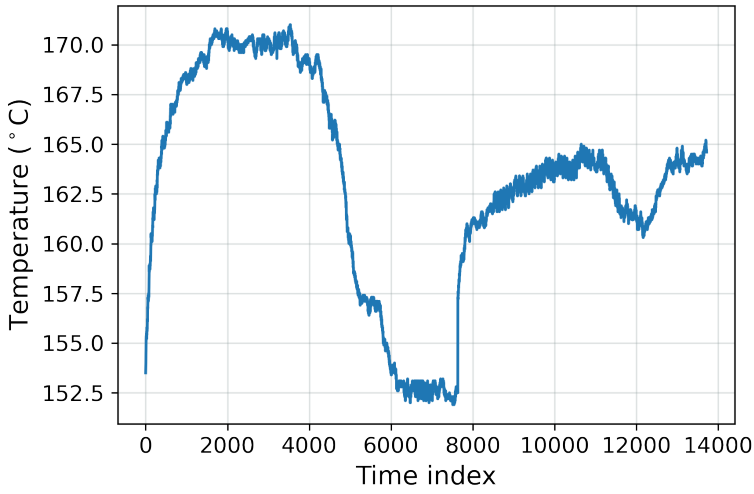



**Figure A.10:** COT before denosing

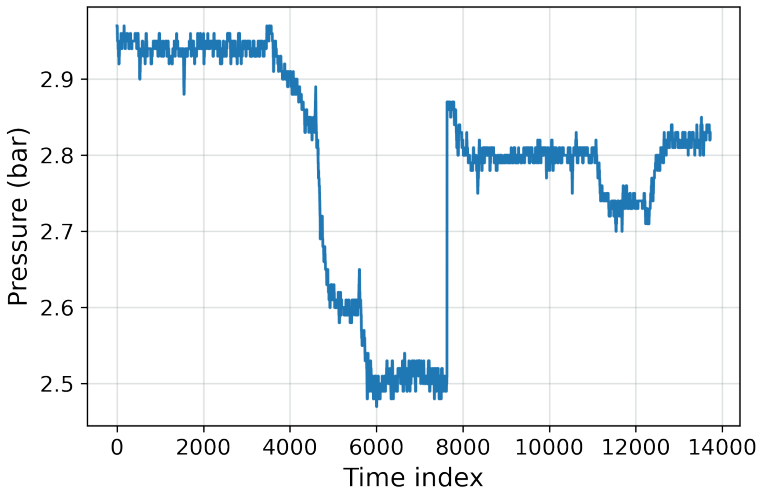



**Figure A.11:** COP before denosing

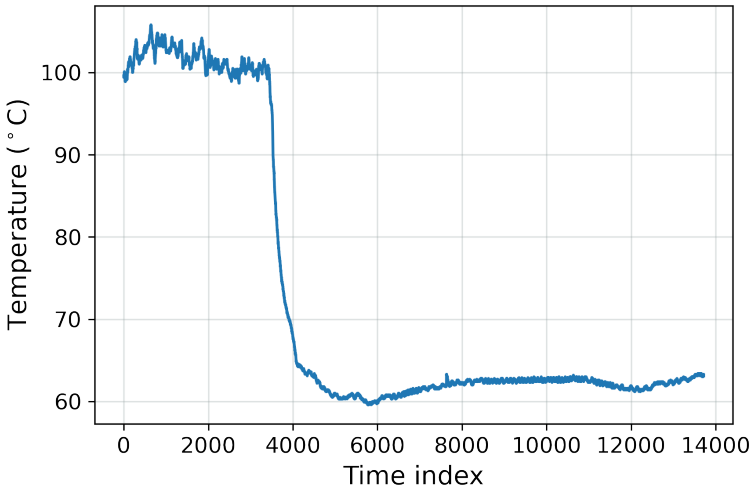



**Figure A.12:** RITa before denosing

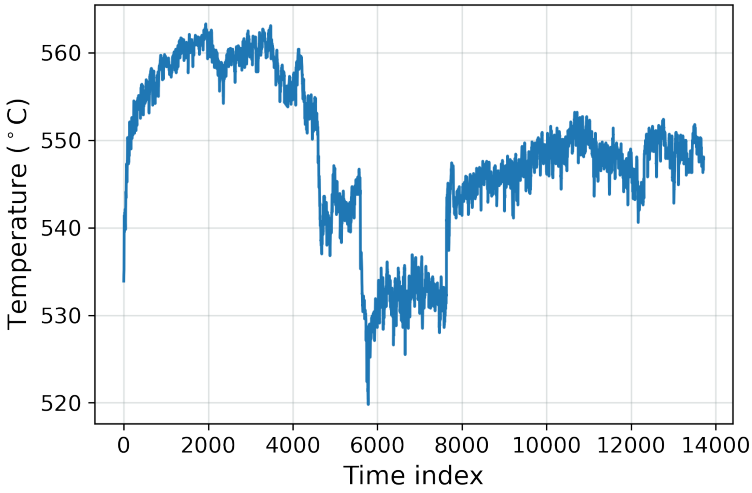



**Figure A.13:** CCIT1 before denosing

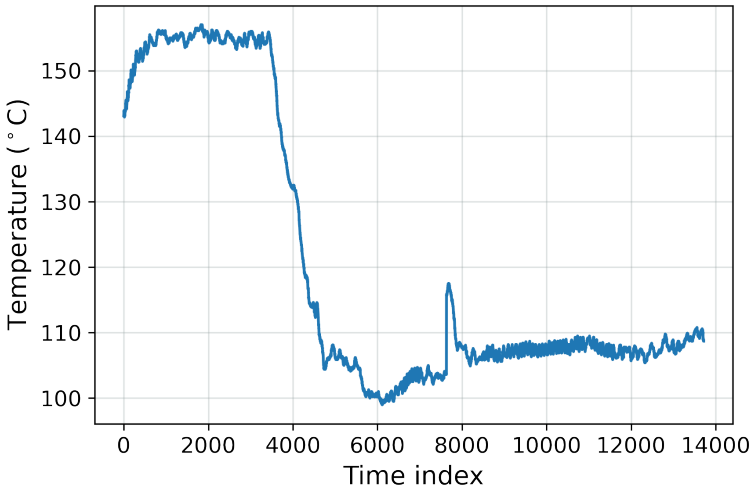



**Figure A.14:** EIT1 before denosing

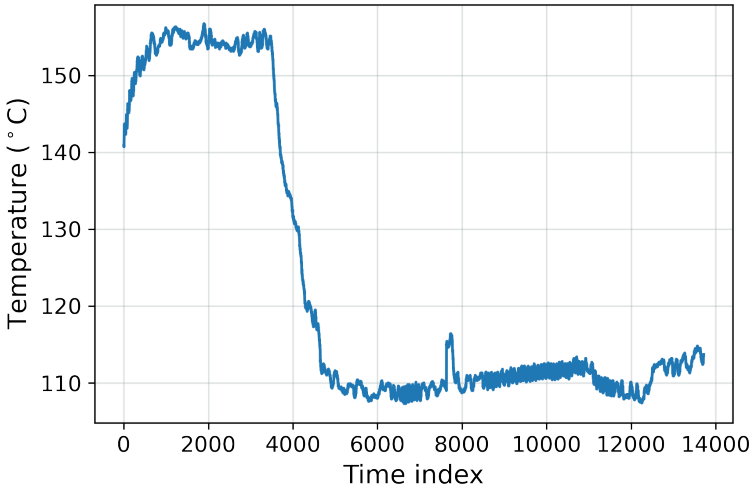



**Figure A.15:** EIT2 before denosing

# Appendix - B

## *Results from reconstruction loss calculations*

The following tables presents the different model configurations. The tables are specified with the number of neurons in the bottle- and mapping layer, together with the corresponding errors calculated from the training-, validation- and testing set. The % error column is the error percentage between validation and evaluation loss.

**Table B.1:** Overview over MSE loss with varying number of neuron configurations

| Bottle | Mapping | MSE | MSE Val. | MSE E. | % Error |
|---|---|---|---|---|---|
| 1 | 0 | 0.0203 | 0.0203 | 0.0204 | 0.5825 |
| 1 | 6 | 0.0037 | 0.0037 | 0.0037 | 0.1171 |
| 1 | 8 | 0.0027 | 0.0032 | 0.0027 | -13.5874 |
| 1 | 10 | 0.0027 | 0.0027 | 0.0027 | 0.4249 |
| | | | | | |
| 2 | 0 | 0.0108 | 0.011 | 0.0110 | 0.0085 |
| 2 | 6 | 0.0026 | 0.0027 | 0.0026 | -0.6335 |
| 2 | 8 | 0.0019 | 0.0019 | 0.0019 | 0.6476 |
| 2 | 10 | 0.0015 | 0.0015 | 0.0015 | 1.7767 |
| | | | | | |
| 3 | 0 | 0.0062 | 0.0062 | 0.0063 | 2.8799 |
| 3 | 6 | 0.0018 | 0.0018 | 0.0018 | 0.8494 |
| 3 | 8 | 0.0013 | 0.0013 | 0.0013 | 0.0605 |
| 3 | 10 | 0.0010 | 0.0010 | 0.0010 | 2.9075 |
| | | | | | |
| 4 | 0 | 0.0042 | 0.0042 | 0.0043 | 3.9355 |
| 4 | 6 | 0.0017 | 0.0017 | 0.0016 | -1.5511 |
| 4 | 8 | 0.0009 | 0.0009 | 0.0010 | 2.1125 |
| 4 | 10 | 0.0007 | 0.0007 | 0.0007 | 1.4185 |
| | | | | | |
| 5 | 0 | 0.0027 | 0.0027 | 0.0027 | 0.7304 |
| 5 | 6 | 0.0016 | 0.0016 | 0.0016 | 1.6099 |
| 5 | 8 | 0.0008 | 0.0008 | 0.0008 | 0.4226 |
| 5 | 10 | 0.0005 | 0.0005 | 0.0005 | -0.4241 |

**Table B.2:** Overview over MAE loss with varying number of neuron configurations

| Bottle | Mapping | MAE | Val. MAE | E. MAE | % Error |
|---|---|---|---|---|---|
| 1 | 0 | 0.1098 | 0.1099 | 0.1099 | 0.0788 |
| 1 | 6 | 0.0401 | 0.0401 | 0.0402 | 0.2716 |
| 1 | 8 | 0.0345 | 0.0375 | 0.0347 | 7.8381 |
| 1 | 10 | 0.034 | 0.0339 | 0.0343 | 1.2572 |
| | | | | | |
| 2 | 0 | 0.0715 | 0.0717 | 0.0718 | 0.1970 |
| 2 | 6 | 0.0346 | 0.0345 | 0.0347 | 0.5860 |
| 2 | 8 | 0.0298 | 0.0297 | 0.0300 | 1.1083 |
| 2 | 10 | 0.0266 | 0.0266 | 0.0269 | 1.2937 |
| | | | | | |
| 3 | 0 | 0.0558 | 0.0561 | 0.0563 | 0.3810 |
| 3 | 6 | 0.0282 | 0.0284 | 0.0282 | 0.3990 |
| 3 | 8 | 0.0255 | 0.0255 | 0.0257 | 0.8706 |
| 3 | 10 | 0.0215 | 0.0217 | 0.0216 | 0.0351 |
| | | | | | |
| 4 | 0 | 0.0423 | 0.0424 | 0.0427 | 0.7148 |
| 4 | 6 | 0.0279 | 0.0278 | 0.0278 | 0.2428 |
| 4 | 8 | 0.0224 | 0.0225 | 0.0226 | 0.6123 |
| 4 | 10 | 0.0188 | 0.0189 | 0.0190 | 0.5725 |
| | | | | | |
| 5 | 0 | 0.0357 | 0.0359 | 0.0359 | 0.0398 |
| 5 | 6 | 0.0278 | 0.0277 | 0.0278 | 0.4194 |
| 5 | 8 | 0.0201 | 0.0201 | 0.0201 | 0.4116 |
| 5 | 10 | 0.0158 | 0.0158 | 0.0158 | 0.6259 |

# Appendix - C

## *Jupyter Notebook code*

The following code explains the whole operation in order to get comparable results. From preprocessing of the data, to training the different networks using the data and testing in order to evaluate and predict visualisations.

# Preprocessing

June 7, 2022

## 1 Processing of all data no outliers

This part of the project is set up for importing and preprocessing of the raw data. This includes using all available data without outliers. Following the book "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow" by Aurélien Géron:

- Get an overview of the available data, loading the excel values into pandas dataframe, generated as a separate file.
- The data has been described in tables and plots. Linear relationships have been researched.
- The dataset has been separated into a training set and a test set, stored as two new separate files: "unprocessed_test_data_no_outliers.csv" and "processed_training_data_no_outliers.csv"
- The training set has been scaled using normalization method

### 1.1 Setup

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# TensorFlow 2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. LSTMs and CNNs can be very slow without a GPU.")
    if IS_COLAB:
```

```
        print("Go to Runtime > Change runtime and select a GPU hardware␣
↪accelerator.")
    if IS_KAGGLE:
        print("Go to Settings > Accelerator and select GPU.")

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "autoencoder"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
```

### 1.2 Load data

[2]:
```
# returns pandas DatamFrame containing all the data

import pandas as pd
data = pd.read_csv('data/all_data_no_outliers.csv')

data.head()
```

[2]:
```
   Index  Preq   T1  deltaP      Pamb  Vwsat  Twsatin  TwIn    Pgen        N  \
0    195    70  4.2  91.785  0.074415  -0.01     33.5  30.5  72.504  63840.0
1    196    70  4.2  90.482  0.074415  -0.01     33.4  30.5  72.500  63840.0
2    197    70  4.2  89.614  0.074415  -0.01     33.4  30.5  72.500  63840.0
3    198    70  4.2  88.919  0.074415  -0.01     33.5  30.5  72.500  63840.0
4    199    70  4.2  88.381  0.074415  -0.01     33.5  30.5  72.500  63840.0

       TOT    COT   COP  RITa  CCIT1   EIT1   EIT2
0  606.079  153.4  2.97  99.5  533.1  143.2  140.8
1  606.116  153.5  2.97  99.5  533.2  143.2  140.8
2  606.139  153.4  2.97  99.5  533.4  143.2  140.8
3  606.157  153.6  2.97  99.6  533.4  143.2  140.8
4  606.176  153.5  2.97  99.5  533.6  143.2  140.8
```

[3]:
```
# Set the time column as the index, because outliers were taken out

data = data.drop('Index', axis=1)
```

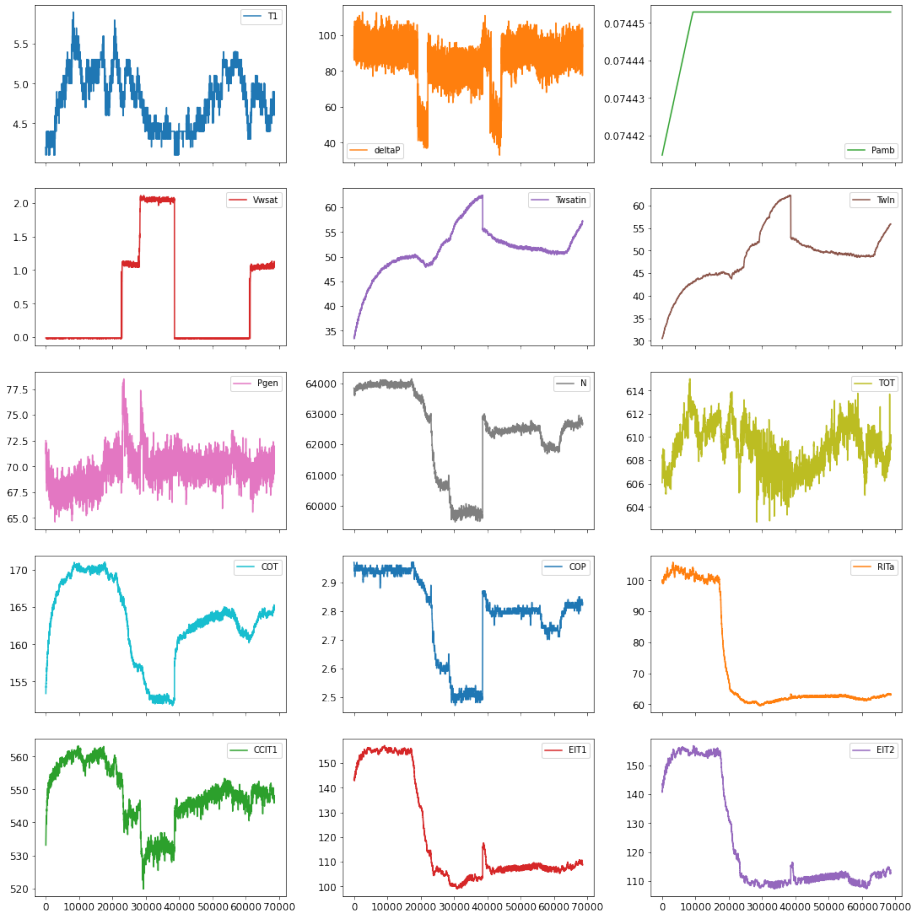
[4]:
```
# drop the Preq column as it is set by

data = data.drop('Preq', axis=1)
```

[5]:
```
data.head()
```

[5]:
```
    T1  deltaP      Pamb  Vwsat  Twsatin  TwIn    Pgen        N      TOT  \
0  4.2  91.785  0.074415  -0.01     33.5  30.5  72.504  63840.0  606.079
1  4.2  90.482  0.074415  -0.01     33.4  30.5  72.500  63840.0  606.116
2  4.2  89.614  0.074415  -0.01     33.4  30.5  72.500  63840.0  606.139
3  4.2  88.919  0.074415  -0.01     33.5  30.5  72.500  63840.0  606.157
4  4.2  88.381  0.074415  -0.01     33.5  30.5  72.500  63840.0  606.176

    COT   COP  RITa  CCIT1   EIT1   EIT2
0  153.4  2.97  99.5  533.1  143.2  140.8
1  153.5  2.97  99.5  533.2  143.2  140.8
2  153.4  2.97  99.5  533.4  143.2  140.8
3  153.6  2.97  99.6  533.4  143.2  140.8
4  153.5  2.97  99.5  533.6  143.2  140.8
```

[28]:
```
#plot features

%matplotlib inline
data.plot(subplots=True, layout=(6,3), figsize=(15,18))
plt.grid()
save_fig("attribute_plots")
```

```
Saving figure attribute_plots
```




### 1.3 Create stratified training + test set

Need to create a stable test/train split every time the dataset gets updated. But is stratified sampling necessary for gas turbine data, can this data be biased?

[7]:
```
#split T1 into 1 categories & view count of each category

data["T1_cat"] = pd.cut(data["T1"],
                        bins=[0, 4.5, 4.8, 5.1, np.inf],
                        labels=[1, 2, 3, 4])

data["T1_cat"].value_counts()
```

[7]:
```
3    18787
1    18708
4    17273
2    13820
Name: T1_cat, dtype: int64
```

[8]:
```
#split the train and test sets

from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(data, data["T1_cat"]):
    strat_train_set = data.loc[train_index]
    strat_test_set = data.loc[test_index]
```

[9]:
```
strat_train_set.head()
```

[9]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | EIT2 | T1_cat |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 25564 | 5.1 | 87.000 | 0.074453 | 1.09 | 51.1 | 49.9 | 70.300 | 60660.079 | 609.599 | 159.2 | 2.60 | 60.4 | 542.0 | 106.2 | 110.1 | 3 |
| 22533 | 4.9 | 82.162 | 0.074453 | -0.01 | 48.4 | 45.7 | 69.120 | 62776.852 | 609.437 | 166.0 | 2.83 | 63.0 | 553.2 | 114.1 | 118.0 | 3 |
| 51133 | 5.0 | 85.789 | 0.074453 | -0.02 | 51.7 | 49.5 | 69.403 | 62483.433 | 609.618 | 163.5 | 2.80 | 62.4 | 546.5 | 108.1 | 110.6 | 3 |
| 8834 | 5.3 | 93.004 | 0.074451 | -0.01 | 47.7 | 42.8 | 68.700 | 64056.694 | 612.576 | 170.5 | 2.95 | 102.1 | 561.5 | 156.3 | 154.2 | 4 |
| 10162 | 5.3 | 91.527 | 0.074453 | -0.01 | 48.3 | 43.5 | 68.900 | 64042.635 | 612.200 | 169.9 | 2.95 | 101.4 | 562.2 | 154.3 | 155.9 | 4 |

[10]:
```
strat_test_set.shape
```

[10]:
```
(13718, 16)
```

[11]:
```
#look at T1 category proportions in test set

strat_test_set["T1_cat"].value_counts() / len(strat_test_set)
```

[11]:
```
3    0.273874
1    0.272780
4    0.251859
2    0.201487
Name: T1_cat, dtype: float64
```

[12]:
```
#look at T1 category proportions in overall data set

data["T1_cat"].value_counts() / len(data)
```

[12]:
```
3    0.273911
1    0.272759
4    0.251837
2    0.201493
Name: T1_cat, dtype: float64
```

[13]:
```
#table comparing overall and test stratification and error
from sklearn.model_selection import train_test_split

def T1_cat_proportions(data):
    return data["T1_cat"].value_counts() / len(data)

train_set, test_set = train_test_split(data, test_size=0.2, random_state=42)

compare_props = pd.DataFrame({
    "Overall": T1_cat_proportions(data),
    "Stratified": T1_cat_proportions(strat_test_set),
    "Random": T1_cat_proportions(test_set),
}).sort_index()
compare_props["Strat. %error"] = 100 * compare_props["Stratified"] /␣
 ↪compare_props["Overall"] - 100
compare_props["Rand. %error"] = 100 * compare_props["Random"] /␣
 ↪compare_props["Overall"] - 100

compare_props
```

[13]:

| | Overall | Stratified | Random | Strat. %error | Rand. %error |
|---|---|---|---|---|---|
| 1 | 0.272759 | 0.272780 | 0.272416 | 0.007774 | -0.125854 |
| 2 | 0.201493 | 0.201487 | 0.202508 | -0.002916 | 0.503582 |
| 3 | 0.273911 | 0.273874 | 0.272853 | -0.013561 | -0.386148 |
| 4 | 0.251837 | 0.251859 | 0.252223 | 0.008663 | 0.153393 |

[14]:
```
#remove the T1_cat attribute
for set_ in (strat_train_set, strat_test_set):
    set_.drop("T1_cat", axis=1, inplace=True)
```

## 2 Arrange Indexes

[15]:
```
strat_test_set.head()
```

[15]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | N | \ |
|---|---|---|---|---|---|---|---|---|---|
| 34687 | 4.400 | 79.351 | 0.074453 | 2.02 | 60.4 | 60.7 | 70.500 | 59860.000 | |
| 31021 | 4.450 | 80.745 | 0.074453 | 2.04 | 57.1 | 56.5 | 68.624 | 59644.969 | |
| 40663 | 4.400 | 92.048 | 0.074453 | -0.01 | 54.6 | 52.1 | 70.287 | 62502.826 | |
| 5271 | 4.957 | 98.268 | 0.074437 | -0.01 | 44.4 | 40.3 | 67.505 | 64000.000 | |
| 1974 | 4.400 | 94.445 | 0.074423 | -0.01 | 39.2 | 35.5 | 67.200 | 63878.899 | |

| | TOT | COT | COP | RITa | CCIT1 | EIT1 | EIT2 |
|---|---|---|---|---|---|---|---|
| 34687 | 607.100 | 152.1 | 2.52 | 61.1 | 533.1 | 104.2 | 107.8 |
| 31021 | 609.200 | 153.4 | 2.49 | 60.5 | 534.3 | 99.1 | 109.5 |
| 40663 | 606.072 | 160.9 | 2.81 | 62.2 | 544.9 | 107.8 | 109.4 |
| 5271 | 609.511 | 168.3 | 2.94 | 102.6 | 559.1 | 154.1 | 155.6 |
| 1974 | 607.278 | 164.3 | 2.95 | 101.7 | 553.8 | 152.7 | 150.1 |

[16]:
```
sorted_test = strat_test_set.sort_index('index')
```

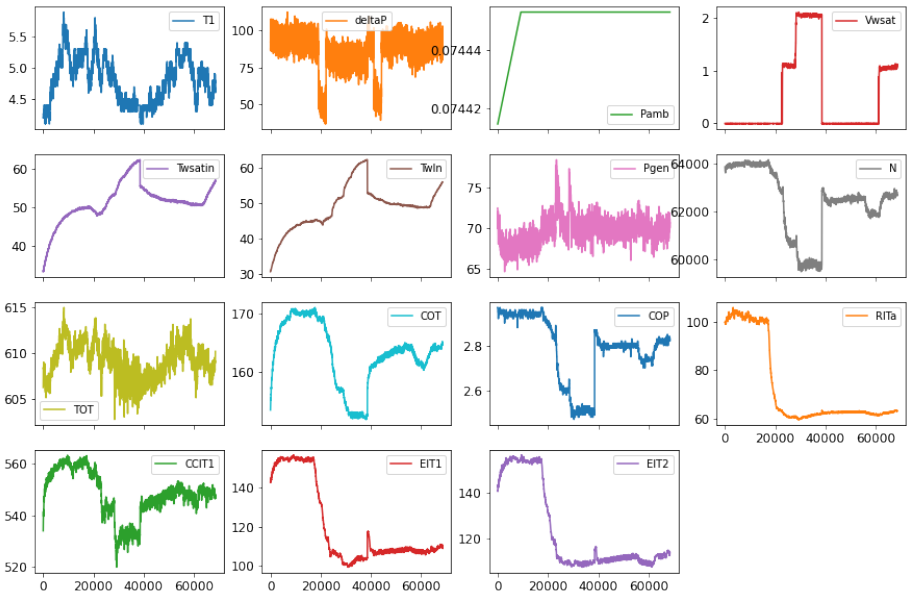
```
/var/folders/3x/ssqftytd2xz72_28lg_8dbjw0000gn/T/ipykernel_83000/2039066215.py:1
: FutureWarning: In a future version of pandas all arguments of
DataFrame.sort_index will be keyword-only.
  sorted_test = strat_test_set.sort_index('index')
```

[17]:
```
sorted_test.shape
```

[17]:
```
(13718, 15)
```

[18]:
```
#plot features

%matplotlib inline
sorted_test.plot(subplots=True, layout=(4,4), figsize=(15,10))
#save_fig("attribute_plots")
```

[18]:
```
array([[<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>],
       [<AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>, <AxesSubplot:>]],
      dtype=object)
```




# 3 Save Test Data

[19]:
```
#save the test data to separate file

sorted_test.to_csv("data/unprocessed_strat_test_data.csv")
```

## 3.1 Prepare the Data for ML Algorithms

### 3.1.1 Feature scaling

p. 109 One of the most iportant transformations need to be applied to data. ML algorithms don't perform well when input attributes have very different scales. This part of the book does, or the rest of the book, does not describe how to actually normalize a dataset.

- Min-max scaling (normalization)
- Standardization = for many outliers

Fit the scalers to the training data only, not the full dataset. Only then can you use them to transform the training set and the test set.

We normalize training and test data separately to avoid *data leakage*.

[20]:
```
strat_train_set
```

[20]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | N | \ |
|---|---|---|---|---|---|---|---|---|---|
| 25564 | 5.100 | 87.000 | 0.074453 | 1.09 | 51.1 | 49.9 | 70.300 | 60660.079 | |
| 22533 | 4.900 | 82.162 | 0.074453 | -0.01 | 48.4 | 45.7 | 69.120 | 62776.852 | |
| 51133 | 5.000 | 85.789 | 0.074453 | -0.02 | 51.7 | 49.5 | 69.403 | 62483.433 | |
| 8834 | 5.300 | 93.004 | 0.074451 | -0.01 | 47.7 | 42.8 | 68.700 | 64056.694 | |
| 10162 | 5.300 | 91.527 | 0.074453 | -0.01 | 48.3 | 43.5 | 68.900 | 64042.635 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 66208 | 4.600 | 90.203 | 0.074453 | 1.07 | 54.1 | 53.2 | 69.469 | 62650.000 | |
| 68396 | 4.898 | 90.958 | 0.074453 | 1.07 | 56.7 | 55.7 | 69.815 | 62790.000 | |
| 4193 | 4.900 | 94.639 | 0.074432 | -0.01 | 43.2 | 38.8 | 68.274 | 63951.806 | |
| 48833 | 4.742 | 92.015 | 0.074453 | -0.01 | 52.0 | 49.6 | 69.700 | 62541.687 | |
| 45982 | 4.600 | 90.449 | 0.074453 | -0.01 | 52.3 | 50.1 | 70.447 | 62463.327 | |

| | TOT | COT | COP | RITa | CCIT1 | EIT1 | EIT2 |
|---|---|---|---|---|---|---|---|
| 25564 | 609.599 | 159.2 | 2.60 | 60.4 | 542.0 | 106.2 | 110.1 |
| 22533 | 609.437 | 166.0 | 2.83 | 63.0 | 553.2 | 114.1 | 118.0 |
| 51133 | 609.618 | 163.5 | 2.80 | 62.4 | 546.5 | 108.1 | 110.6 |
| 8834 | 612.576 | 170.5 | 2.95 | 102.1 | 561.5 | 156.3 | 154.2 |
| 10162 | 612.200 | 169.9 | 2.95 | 101.4 | 562.2 | 154.3 | 155.9 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 66208 | 607.998 | 163.8 | 2.82 | 62.9 | 550.8 | 108.0 | 113.3 |
| 68396 | 608.202 | 165.0 | 2.84 | 63.1 | 547.9 | 110.6 | 112.9 |
| 4193 | 608.363 | 168.0 | 2.95 | 103.7 | 558.3 | 155.9 | 153.8 |
| 48833 | 609.158 | 162.8 | 2.80 | 62.4 | 548.2 | 108.2 | 110.5 |
| 45982 | 608.200 | 163.0 | 2.80 | 62.5 | 545.0 | 106.7 | 109.9 |

[54870 rows x 15 columns]

[21]:
```
sorted_train = strat_train_set.sort_index('index')
```

```
/var/folders/3x/ssqftytd2xz72_28lg_8dbjw0000gn/T/ipykernel_83000/905008516.py:1:
FutureWarning: In a future version of pandas all arguments of
DataFrame.sort_index will be keyword-only.
  sorted_train = strat_train_set.sort_index('index')
```

[22]:
```
sorted_train
```

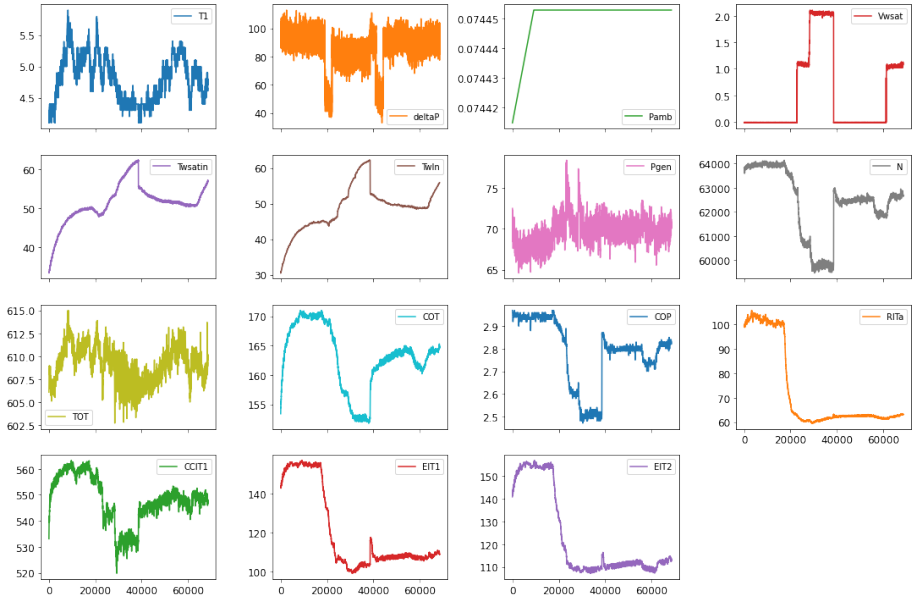
[22]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | N | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 4.200 | 91.785 | 0.074415 | -0.01 | 33.5 | 30.5 | 72.504 | 63840.000 | |
| 1 | 4.200 | 90.482 | 0.074415 | -0.01 | 33.4 | 30.5 | 72.500 | 63840.000 | |
| 2 | 4.200 | 89.614 | 0.074415 | -0.01 | 33.4 | 30.5 | 72.500 | 63840.000 | |
| 3 | 4.200 | 88.919 | 0.074415 | -0.01 | 33.5 | 30.5 | 72.500 | 63840.000 | |
| 4 | 4.200 | 88.381 | 0.074415 | -0.01 | 33.5 | 30.5 | 72.500 | 63840.000 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 68580 | 4.798 | 103.955 | 0.074453 | 1.09 | 57.2 | 55.9 | 70.700 | 62660.812 | |
| 68581 | 4.753 | 102.979 | 0.074453 | 1.11 | 57.2 | 55.9 | 70.700 | 62663.435 | |
| 68582 | 4.680 | 101.190 | 0.074453 | 1.11 | 57.3 | 55.9 | 70.700 | 62665.482 | |
| 68585 | 4.620 | 96.244 | 0.074453 | 1.11 | 57.2 | 55.9 | 70.700 | 62678.297 | |

```
68586  4.651   94.933  0.074453   1.11     57.1  55.9  70.700  62687.332

           TOT    COT   COP  RITa  CCIT1   EIT1   EIT2
0      606.079  153.4  2.97  99.5  533.1  143.2  140.8
1      606.116  153.5  2.97  99.5  533.2  143.2  140.8
2      606.139  153.4  2.97  99.5  533.4  143.2  140.8
3      606.157  153.6  2.97  99.6  533.4  143.2  140.8
4      606.176  153.5  2.97  99.5  533.6  143.2  140.8
...        ...    ...   ...   ...    ...    ...    ...
68580  610.001  164.7  2.82  63.4  548.2  108.7  113.6
68581  610.017  164.7  2.82  63.2  548.2  108.7  113.6
68582  610.047  164.7  2.82  63.2  548.0  108.7  113.6
68585  610.135  164.6  2.82  63.1  547.9  108.7  113.6
68586  610.164  164.6  2.83  63.3  547.9  108.7  113.7

[54870 rows x 15 columns]
```

[23]:
```
#plot features

%matplotlib inline
sorted_train.plot(subplots=True, layout=(4,4), figsize=(15,10))
save_fig("attribute_plots")
```

```
Saving figure attribute_plots
```



[24]:
```
sorted_train.shape
```

[24]:
```
(54870, 15)
```

[25]:
```
from sklearn.preprocessing import MinMaxScaler

# define min max scaler
scaler = MinMaxScaler()

# define data to be scaled
normalize_these = ['T1', 'deltaP','Pgen', 'N',
                   'TOT','COT', 'COP', 'RITa',
                   'CCIT1', 'EIT1', 'EIT2',
                  'Pamb', 'Twsatin', 'TwIn', 'Vwsat']

# transform data
sorted_train[normalize_these] = scaler.
 ↪fit_transform(sorted_train[normalize_these])

sorted_train
```

[25]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.055556 | 0.735371 | 0.000000 | 0.004695 | 0.003448 | 0.000000 | 0.572754 | |
| 1 | 0.055556 | 0.719058 | 0.000105 | 0.004695 | 0.000000 | 0.000000 | 0.572464 | |
| 2 | 0.055556 | 0.708190 | 0.000210 | 0.004695 | 0.000000 | 0.000000 | 0.572464 | |
| 3 | 0.055556 | 0.699489 | 0.000314 | 0.004695 | 0.003448 | 0.000000 | 0.572464 | |
| 4 | 0.055556 | 0.692754 | 0.000419 | 0.004695 | 0.003448 | 0.000000 | 0.572464 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68580 | 0.387778 | 0.887736 | 1.000000 | 0.521127 | 0.820690 | 0.801262 | 0.442029 | |
| 68581 | 0.362778 | 0.875516 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.442029 | |
| 68582 | 0.322222 | 0.853119 | 1.000000 | 0.530516 | 0.824138 | 0.801262 | 0.442029 | |
| 68585 | 0.288889 | 0.791196 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.442029 | |
| 68586 | 0.306111 | 0.774783 | 1.000000 | 0.530516 | 0.817241 | 0.801262 | 0.442029 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.935906 | 0.274701 | 0.083333 | 1.00 | 0.863636 | 0.306452 | 0.762069 | |
| 1 | 0.935906 | 0.277710 | 0.088542 | 1.00 | 0.863636 | 0.308756 | 0.762069 | |
| 2 | 0.935906 | 0.279580 | 0.083333 | 1.00 | 0.863636 | 0.313364 | 0.762069 | |
| 3 | 0.935906 | 0.281044 | 0.093750 | 1.00 | 0.865801 | 0.313364 | 0.762069 | |
| 4 | 0.935906 | 0.282589 | 0.088542 | 1.00 | 0.863636 | 0.317972 | 0.762069 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68580 | 0.683934 | 0.593641 | 0.671875 | 0.70 | 0.082251 | 0.654378 | 0.167241 | |
| 68581 | 0.684495 | 0.594942 | 0.671875 | 0.70 | 0.077922 | 0.654378 | 0.167241 | |
| 68582 | 0.684932 | 0.597381 | 0.671875 | 0.70 | 0.077922 | 0.649770 | 0.167241 | |
| 68585 | 0.687670 | 0.604538 | 0.666667 | 0.70 | 0.075758 | 0.647465 | 0.167241 | |
| 68586 | 0.689601 | 0.606896 | 0.666667 | 0.72 | 0.080087 | 0.647465 | 0.167241 | |

```
           EIT2
0      0.677419
1      0.677419
2      0.677419
3      0.677419
4      0.677419
...         ...
68580  0.129032
68581  0.129032
68582  0.129032
68585  0.129032
68586  0.131048

[54870 rows x 15 columns]
```

[26]:
```
#save the test data to separate file

sorted_train.to_csv("data/processed_strat_train_data.csv")
```

# Training

June 7, 2022

## 0.1 Part II: Create Model

This part of the project focusing on creating a model with training data in the file "processed_training_data_no_outliers.csv".

- The training data is loaded, then split into a training and validation set
- Create AE model (layer, activation functions, initializer, regularizer)
- Compile model (loss function, optimizer, metrics)
- Train model

## 0.2 Setup

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# TensorFlow 2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. LSTMs and CNNs can be very slow without a GPU.")
    if IS_COLAB:
        print("Go to Runtime > Change runtime and select a GPU hardware␣
 ↪accelerator.")
    if IS_KAGGLE:
        print("Go to Settings > Accelerator and select GPU.")

# Common imports
import numpy as np
```

```
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "autoencoder"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
```

[2]:
```
# disable GPU for faster calculations
# for bigger batch sized (=>1024) GPU will be faster

tf.config.set_visible_devices([], 'GPU')
```

### 0.3 Load data

[3]:
```
# load the csv into a pandas dataframe

import pandas as pd
def load_data(data_path):
    csv_path = pd.read_csv(data_path)
    return csv_path
```

[4]:
```
# returns pandas DatamFrame containing all the data

data = pd.read_csv('data/processed_strat_train_data.csv', index_col=0)
```

```
data
```

[4]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.055556 | 0.735371 | 0.000000 | 0.004695 | 0.003448 | 0.000000 | 0.572754 | |
| 1 | 0.055556 | 0.719058 | 0.000105 | 0.004695 | 0.000000 | 0.000000 | 0.572464 | |
| 2 | 0.055556 | 0.708190 | 0.000210 | 0.004695 | 0.000000 | 0.000000 | 0.572464 | |
| 3 | 0.055556 | 0.699489 | 0.000314 | 0.004695 | 0.003448 | 0.000000 | 0.572464 | |
| 4 | 0.055556 | 0.692754 | 0.000419 | 0.004695 | 0.003448 | 0.000000 | 0.572464 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68580 | 0.387778 | 0.887736 | 1.000000 | 0.521127 | 0.820690 | 0.801262 | 0.442029 | |
| 68581 | 0.362778 | 0.875516 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.442029 | |
| 68582 | 0.322222 | 0.853119 | 1.000000 | 0.530516 | 0.824138 | 0.801262 | 0.442029 | |
| 68585 | 0.288889 | 0.791196 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.442029 | |
| 68586 | 0.306111 | 0.774783 | 1.000000 | 0.530516 | 0.817241 | 0.801262 | 0.442029 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.935906 | 0.274701 | 0.083333 | 1.00 | 0.863636 | 0.306452 | 0.762069 | |
| 1 | 0.935906 | 0.277710 | 0.088542 | 1.00 | 0.863636 | 0.308756 | 0.762069 | |
| 2 | 0.935906 | 0.279580 | 0.083333 | 1.00 | 0.863636 | 0.313364 | 0.762069 | |
| 3 | 0.935906 | 0.281044 | 0.093750 | 1.00 | 0.865801 | 0.313364 | 0.762069 | |
| 4 | 0.935906 | 0.282589 | 0.088542 | 1.00 | 0.863636 | 0.317972 | 0.762069 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68580 | 0.683934 | 0.593641 | 0.671875 | 0.70 | 0.082251 | 0.654378 | 0.167241 | |
| 68581 | 0.684495 | 0.594942 | 0.671875 | 0.70 | 0.077922 | 0.654378 | 0.167241 | |
| 68582 | 0.684932 | 0.597381 | 0.671875 | 0.70 | 0.077922 | 0.649770 | 0.167241 | |
| 68585 | 0.687670 | 0.604538 | 0.666667 | 0.70 | 0.075758 | 0.647465 | 0.167241 | |
| 68586 | 0.689601 | 0.606896 | 0.666667 | 0.72 | 0.080087 | 0.647465 | 0.167241 | |

| | EIT2 |
|---|---|
| 0 | 0.677419 |
| 1 | 0.677419 |
| 2 | 0.677419 |
| 3 | 0.677419 |
| 4 | 0.677419 |
| ... | ... |
| 68580 | 0.129032 |
| 68581 | 0.129032 |
| 68582 | 0.129032 |
| 68585 | 0.129032 |
| 68586 | 0.131048 |

[54870 rows x 15 columns]

### 0.4 Split into training and validation set

[5]:
```
from sklearn.model_selection import train_test_split

X_train, X_val = train_test_split(data, test_size=0.2, random_state=42)
```

[6]:
```
data.shape
```

[6]: (54870, 15)

[7]:
```
X_train.shape
```

[7]: (43896, 15)

[8]:
```
X_val.shape
```

[8]: (10974, 15)

[9]:
```
X_train.shape[1]
```

[9]: 15

[10]:
```
X_train
```

[10]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 10465 | 0.573889 | 0.794276 | 1.000000 | 0.004695 | 0.520690 | 0.413249 | 0.365362 | |
| 59759 | 0.500000 | 0.650424 | 1.000000 | 0.004695 | 0.606897 | 0.580442 | 0.449275 | |
| 59396 | 0.444444 | 0.690162 | 1.000000 | 0.004695 | 0.606897 | 0.570978 | 0.268116 | |
| 17936 | 0.444444 | 0.669868 | 1.000000 | 0.004695 | 0.572414 | 0.457413 | 0.479565 | |
| 45207 | 0.256667 | 0.766657 | 1.000000 | 0.004695 | 0.679310 | 0.630915 | 0.463768 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 55818 | 0.635000 | 0.780667 | 1.000000 | 0.004695 | 0.627586 | 0.593060 | 0.575000 | |
| 67928 | 0.333333 | 0.725505 | 1.000000 | 0.516432 | 0.782759 | 0.776025 | 0.376957 | |
| 47572 | 0.295000 | 0.769449 | 1.000000 | 0.004695 | 0.648276 | 0.605678 | 0.427536 | |
| 1052 | 0.111111 | 0.825337 | 0.114274 | 0.004695 | 0.110345 | 0.088328 | 0.253623 | |
| 19766 | 0.645000 | 0.339873 | 1.000000 | 0.004695 | 0.555172 | 0.447950 | 0.391304 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 10465 | 0.954410 | 0.753517 | 0.973958 | 0.96 | 0.906926 | 0.944700 | 0.982759 | |
| 59759 | 0.525919 | 0.561031 | 0.494792 | 0.54 | 0.041126 | 0.571429 | 0.141379 | |
| 59396 | 0.513928 | 0.539481 | 0.494792 | 0.52 | 0.041126 | 0.631336 | 0.143103 | |
| 17936 | 0.922536 | 0.508498 | 0.953125 | 0.98 | 0.532468 | 0.894009 | 0.858621 | |
| 45207 | 0.660256 | 0.427096 | 0.552083 | 0.66 | 0.067100 | 0.617512 | 0.132759 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 55818 | 0.644402 | 0.595836 | 0.656250 | 0.68 | 0.067100 | 0.670507 | 0.158621 | |
| 67928 | 0.705057 | 0.422786 | 0.645833 | 0.72 | 0.077922 | 0.684332 | 0.182759 | |
| 47572 | 0.658247 | 0.443929 | 0.598958 | 0.66 | 0.067100 | 0.656682 | 0.132759 | |
| 1052 | 0.919091 | 0.300886 | 0.520833 | 0.94 | 0.891775 | 0.758065 | 0.862069 | |

```
19766  0.882485  0.556884  0.911458  0.88  0.201299  0.838710  0.572414

           EIT2
10465  0.945565
59759  0.024194
59396  0.018145
17936  0.798387
45207  0.084677
...         ...
55818  0.106855
67928  0.149194
47572  0.100806
1052   0.848790
19766  0.532258

[43896 rows x 15 columns]
```

[11]:
```
X_val
```

[11]:
```
              T1    deltaP  Pamb     Vwsat   Twsatin      TwIn      Pgen  \
44269  0.166667  0.715277   1.0  0.004695  0.675862  0.643533  0.325507
23736  0.555556  0.615094   1.0  0.521127  0.541379  0.482650  0.565870
38886  0.110000  0.651626   1.0  0.004695  0.751724  0.697161  0.414275
34711  0.166667  0.575306   1.0  0.976526  0.937931  0.952681  0.451957
62136  0.491667  0.711295   1.0  0.488263  0.600000  0.580442  0.173913
...         ...       ...   ...       ...       ...       ...       ...
45260  0.222222  0.613116   1.0  0.004695  0.672414  0.627760  0.297246
51517  0.445556  0.713023   1.0  0.004695  0.627586  0.596215  0.528986
68271  0.388889  0.755039   1.0  0.497653  0.793103  0.791798  0.434783
65565  0.475556  0.748253   1.0  0.502347  0.689655  0.687697  0.434783
34526  0.166667  0.673623   1.0  0.957746  0.927586  0.946372  0.480507

               N       TOT       COT   COP      RITa     CCIT1      EIT1  \
44269  0.611841  0.237863  0.552083  0.62  0.064935  0.573733  0.113793
23736  0.460801  0.592340  0.671875  0.46  0.045455  0.490783  0.125862
38886  0.735045  0.325201  0.364583  0.78  0.051948  0.410138  0.317241
34711  0.097673  0.271123  0.026042  0.12  0.032468  0.290323  0.086207
62136  0.598202  0.561031  0.510417  0.58  0.041126  0.672811  0.132759
...         ...       ...       ...   ...       ...       ...       ...
45260  0.640992  0.325201  0.546875  0.64  0.060606  0.638249  0.139655
51517  0.657660  0.560706  0.635417  0.68  0.064935  0.654378  0.143103
68271  0.707266  0.517281  0.661458  0.74  0.077922  0.695853  0.191379
65565  0.687682  0.617956  0.661458  0.70  0.060606  0.682028  0.162069
34526  0.107731  0.211352  0.057292  0.12  0.038961  0.387097  0.086207

           EIT2
44269  0.074597
```

```
23736  0.098790
38886  0.149194
34711  0.014113
62136  0.030242
...         ...
45260  0.070565
51517  0.092742
68271  0.129032
65565  0.088710
34526  0.040323

[10974 rows x 15 columns]
```

[12]:
```
X_val.shape
```

[12]:
```
(10974, 15)
```

## 1 Create model + Compiling

[13]:
```
# defining root log directory for TensorBoard logs
# for visualization of data

root_logdir = os.path.join(os.curdir, "my_logs")

def get_run_logdir():
    import time
    run_id = time.strftime("run_%Y_%m_%d-%H_%M_%S")
    return os.path.join(root_logdir, run_id)

run_logdir = get_run_logdir()
run_logdir
```

[13]:
```
'./my_logs/run_2022_06_07-14_20_25'
```

[14]:
```
# callback for TensorBoard
# creates log directory, create event files and summaries
# during training

tensorboard = keras.callbacks.TensorBoard(run_logdir)
```

```
2022-06-07 14:20:25.110222: I
tensorflow/core/profiler/lib/profiler_session.cc:126] Profiler session
initializing.
2022-06-07 14:20:25.110234: I
tensorflow/core/profiler/lib/profiler_session.cc:141] Profiler session started.
2022-06-07 14:20:25.110533: I
tensorflow/core/profiler/lib/profiler_session.cc:159] Profiler session tear
```

down.

[15]:
```
# Create model
# With hidden layers

n_neurons = 6
b_neurons = 3
hl_activation = "tanh"

encoder_input = keras.Input(shape=(X_train.shape[1]))
x = keras.layers.Dense(n_neurons, hl_activation)(encoder_input)
encoder_output = keras.layers.Dense(b_neurons, hl_activation)(x)

encoder = keras.Model(encoder_input, encoder_output)

decoder_input = keras.layers.Dense(n_neurons, hl_activation)(encoder_output)
decoder_output = keras.layers.Dense(X_train.shape[1])(decoder_input)

autoencoder = keras.Model(encoder_input, decoder_output)

autoencoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 15)]              0
_________________________________________________________________
dense (Dense)                (None, 6)                 96
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 21
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 24
_________________________________________________________________
dense_3 (Dense)              (None, 15)                105
=================================================================
Total params: 246
Trainable params: 246
Non-trainable params: 0
_________________________________________________________________
```

Create model  No hidden layers, only bottleneck

$\mathrm{b}_neurons = 6hl_activation = "tanh"$

$\mathrm{encoder}_input = keras.Input(shape = (X_train.shape[1]))encoder_output = keras.layers.Dense(b_neurons, hl_activation)(encoder_input)$

$\mathrm{encoder} = \mathrm{keras.Model}(\mathrm{encoder}_input, encoder_output)$

$\mathrm{decoder}_output = keras.layers.Dense(X_train.shape[1])(encoder_output)$

autoencoder = keras.Model(encoder$_input$, $decoder_output$)

autoencoder.summary()

## 2 Training and Evaluating

[16]:
```
# early stopping callback method, should also be added in fit see below.

checkpoint_cb = keras.callbacks.ModelCheckpoint("ae_keras_model_3-6.h5",
                                                save_best_only=True)

early_stopping_cb = keras.callbacks.EarlyStopping(patience=50,
                                                  restore_best_weights=True)
```

[17]:
```
opt = keras.optimizers.SGD(learning_rate=1)

autoencoder.compile(loss="mse",
                    optimizer = opt,
                    metrics=['mae'])
```

[18]:
```
history = autoencoder.fit(X_train, X_train, epochs=100000,
                          batch_size=42, #implement mini-batch GD
                          validation_data=(X_val, X_val),
                          callbacks=[checkpoint_cb, early_stopping_cb]) # early␣
 ↪stopping
```

```
Epoch 1/100000
   1/1046 [...] - ETA: 1:53 - loss: 0.3656 - mae:
0.5168

2022-06-07 14:20:25.184747: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR
Optimization Passes are enabled (registered 2)
2022-06-07 14:20:25.187293: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz

1046/1046 [==============================] - 0s 328us/step - loss: 0.0121 - mae:
0.0743 - val_loss: 0.0067 - val_mae: 0.0579
Epoch 2/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0062 - mae:
0.0552 - val_loss: 0.0059 - val_mae: 0.0532
Epoch 3/100000
1046/1046 [==============================] - 0s 267us/step - loss: 0.0052 - mae:
0.0487 - val_loss: 0.0051 - val_mae: 0.0472
Epoch 4/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0048 - mae:
0.0452 - val_loss: 0.0050 - val_mae: 0.0463
Epoch 5/100000
```

```
1046/1046 [==============================] - 0s 274us/step - loss: 0.0046 - mae:
0.0439 - val_loss: 0.0048 - val_mae: 0.0457
Epoch 6/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0044 - mae:
0.0436 - val_loss: 0.0043 - val_mae: 0.0431
Epoch 7/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0042 - mae:
0.0437 - val_loss: 0.0041 - val_mae: 0.0433
Epoch 8/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0039 - mae:
0.0435 - val_loss: 0.0039 - val_mae: 0.0424
Epoch 9/100000
1046/1046 [==============================] - 0s 279us/step - loss: 0.0038 - mae:
0.0432 - val_loss: 0.0039 - val_mae: 0.0444
Epoch 10/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0037 - mae:
0.0427 - val_loss: 0.0038 - val_mae: 0.0446
Epoch 11/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0036 - mae:
0.0423 - val_loss: 0.0038 - val_mae: 0.0422
Epoch 12/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0035 - mae:
0.0419 - val_loss: 0.0036 - val_mae: 0.0438
Epoch 13/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0033 - mae:
0.0411 - val_loss: 0.0033 - val_mae: 0.0405
Epoch 14/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0031 - mae:
0.0399 - val_loss: 0.0030 - val_mae: 0.0391
Epoch 15/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0029 - mae:
0.0379 - val_loss: 0.0031 - val_mae: 0.0384
Epoch 16/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0027 - mae:
0.0364 - val_loss: 0.0027 - val_mae: 0.0355
Epoch 17/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0026 - mae:
0.0352 - val_loss: 0.0025 - val_mae: 0.0352
Epoch 18/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0025 - mae:
0.0345 - val_loss: 0.0028 - val_mae: 0.0370
Epoch 19/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0024 - mae:
0.0341 - val_loss: 0.0024 - val_mae: 0.0339
Epoch 20/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0024 - mae:
0.0337 - val_loss: 0.0024 - val_mae: 0.0335
Epoch 21/100000
```

```
1046/1046 [==============================] - 0s 274us/step - loss: 0.0023 - mae:
0.0335 - val_loss: 0.0023 - val_mae: 0.0331
Epoch 22/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0023 - mae:
0.0332 - val_loss: 0.0024 - val_mae: 0.0337
Epoch 23/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0023 - mae:
0.0329 - val_loss: 0.0023 - val_mae: 0.0329
Epoch 24/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0023 - mae:
0.0328 - val_loss: 0.0022 - val_mae: 0.0321
Epoch 25/100000
1046/1046 [==============================] - 0s 288us/step - loss: 0.0022 - mae:
0.0325 - val_loss: 0.0023 - val_mae: 0.0336
Epoch 26/100000
1046/1046 [==============================] - 0s 318us/step - loss: 0.0022 - mae:
0.0324 - val_loss: 0.0024 - val_mae: 0.0339
Epoch 27/100000
1046/1046 [==============================] - 0s 239us/step - loss: 0.0022 - mae:
0.0321 - val_loss: 0.0022 - val_mae: 0.0325
Epoch 28/100000
1046/1046 [==============================] - 0s 244us/step - loss: 0.0022 - mae:
0.0319 - val_loss: 0.0025 - val_mae: 0.0344
Epoch 29/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0022 - mae:
0.0318 - val_loss: 0.0022 - val_mae: 0.0328
Epoch 30/100000
1046/1046 [==============================] - 0s 268us/step - loss: 0.0021 - mae:
0.0317 - val_loss: 0.0022 - val_mae: 0.0327
Epoch 31/100000
1046/1046 [==============================] - 0s 229us/step - loss: 0.0021 - mae:
0.0314 - val_loss: 0.0022 - val_mae: 0.0322
Epoch 32/100000
1046/1046 [==============================] - 0s 239us/step - loss: 0.0021 - mae:
0.0314 - val_loss: 0.0022 - val_mae: 0.0321
Epoch 33/100000
1046/1046 [==============================] - 0s 260us/step - loss: 0.0021 - mae:
0.0311 - val_loss: 0.0022 - val_mae: 0.0318
Epoch 34/100000
1046/1046 [==============================] - 0s 255us/step - loss: 0.0021 - mae:
0.0310 - val_loss: 0.0022 - val_mae: 0.0320
Epoch 35/100000
1046/1046 [==============================] - 0s 290us/step - loss: 0.0021 - mae:
0.0311 - val_loss: 0.0021 - val_mae: 0.0316
Epoch 36/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0021 - mae:
0.0309 - val_loss: 0.0021 - val_mae: 0.0314
Epoch 37/100000
```

```
1046/1046 [==============================] - 0s 278us/step - loss: 0.0021 - mae:
0.0308 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 38/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0021 - mae:
0.0308 - val_loss: 0.0027 - val_mae: 0.0375
Epoch 39/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0021 - mae:
0.0306 - val_loss: 0.0026 - val_mae: 0.0365
Epoch 40/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0305 - val_loss: 0.0023 - val_mae: 0.0335
Epoch 41/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0305 - val_loss: 0.0020 - val_mae: 0.0303
Epoch 42/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0305 - val_loss: 0.0026 - val_mae: 0.0356
Epoch 43/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0304 - val_loss: 0.0020 - val_mae: 0.0299
Epoch 44/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0303 - val_loss: 0.0021 - val_mae: 0.0306
Epoch 45/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0020 - mae:
0.0305 - val_loss: 0.0022 - val_mae: 0.0320
Epoch 46/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0302 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 47/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0020 - mae:
0.0302 - val_loss: 0.0020 - val_mae: 0.0301
Epoch 48/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0020 - mae:
0.0302 - val_loss: 0.0020 - val_mae: 0.0300
Epoch 49/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0302 - val_loss: 0.0020 - val_mae: 0.0304
Epoch 50/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0302 - val_loss: 0.0020 - val_mae: 0.0300
Epoch 51/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0301 - val_loss: 0.0022 - val_mae: 0.0311
Epoch 52/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0023 - val_mae: 0.0325
Epoch 53/100000
```

```
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 54/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0301 - val_loss: 0.0022 - val_mae: 0.0313
Epoch 55/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 56/100000
1046/1046 [==============================] - 0s 271us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0021 - val_mae: 0.0304
Epoch 57/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0021 - val_mae: 0.0309
Epoch 58/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0299 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 59/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0021 - val_mae: 0.0304
Epoch 60/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0300 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 61/100000
1046/1046 [==============================] - 0s 269us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0022 - val_mae: 0.0328
Epoch 62/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0020 - mae:
0.0299 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 63/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0299 - val_loss: 0.0022 - val_mae: 0.0316
Epoch 64/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0305
Epoch 65/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 66/100000
1046/1046 [==============================] - 0s 279us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0301
Epoch 67/100000
1046/1046 [==============================] - 0s 266us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 68/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0021 - val_mae: 0.0308
Epoch 69/100000
```

```
1046/1046 [==============================] - 0s 270us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 70/100000
1046/1046 [==============================] - 0s 267us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0020 - val_mae: 0.0293
Epoch 71/100000
1046/1046 [==============================] - 0s 265us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0027 - val_mae: 0.0366
Epoch 72/100000
1046/1046 [==============================] - 0s 264us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 73/100000
1046/1046 [==============================] - 0s 264us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0020 - val_mae: 0.0296
Epoch 74/100000
1046/1046 [==============================] - 0s 266us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0021 - val_mae: 0.0305
Epoch 75/100000
1046/1046 [==============================] - 0s 267us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0020 - val_mae: 0.0293
Epoch 76/100000
1046/1046 [==============================] - 0s 265us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0025 - val_mae: 0.0341
Epoch 77/100000
1046/1046 [==============================] - 0s 263us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 78/100000
1046/1046 [==============================] - 0s 263us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0299
Epoch 79/100000
1046/1046 [==============================] - 0s 262us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 80/100000
1046/1046 [==============================] - 0s 262us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0022 - val_mae: 0.0319
Epoch 81/100000
1046/1046 [==============================] - 0s 260us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0020 - val_mae: 0.0302
Epoch 82/100000
1046/1046 [==============================] - 0s 259us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0317
Epoch 83/100000
1046/1046 [==============================] - 0s 257us/step - loss: 0.0020 - mae:
0.0298 - val_loss: 0.0021 - val_mae: 0.0306
Epoch 84/100000
1046/1046 [==============================] - 0s 260us/step - loss: 0.0020 - mae:
0.0294 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 85/100000
```

```
1046/1046 [==============================] - 0s 260us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0022 - val_mae: 0.0310
Epoch 86/100000
1046/1046 [==============================] - 0s 258us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 87/100000
1046/1046 [==============================] - 0s 259us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 88/100000
1046/1046 [==============================] - 0s 257us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0025 - val_mae: 0.0356
Epoch 89/100000
1046/1046 [==============================] - 0s 259us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0021 - val_mae: 0.0314
Epoch 90/100000
1046/1046 [==============================] - 0s 260us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0308
Epoch 91/100000
1046/1046 [==============================] - 0s 279us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0019 - val_mae: 0.0291
Epoch 92/100000
1046/1046 [==============================] - 0s 282us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0022 - val_mae: 0.0327
Epoch 93/100000
1046/1046 [==============================] - 0s 289us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 94/100000
1046/1046 [==============================] - 0s 265us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0288
Epoch 95/100000
1046/1046 [==============================] - 0s 261us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0302
Epoch 96/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0019 - val_mae: 0.0292
Epoch 97/100000
1046/1046 [==============================] - 0s 269us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0027 - val_mae: 0.0380
Epoch 98/100000
1046/1046 [==============================] - 0s 264us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0025 - val_mae: 0.0349
Epoch 99/100000
1046/1046 [==============================] - 0s 261us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0022 - val_mae: 0.0309
Epoch 100/100000
1046/1046 [==============================] - 0s 278us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0020 - val_mae: 0.0302
Epoch 101/100000
```

```
1046/1046 [==============================] - 0s 277us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 102/100000
1046/1046 [==============================] - 0s 266us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 103/100000
1046/1046 [==============================] - 0s 265us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0308
Epoch 104/100000
1046/1046 [==============================] - 0s 267us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0304
Epoch 105/100000
1046/1046 [==============================] - 0s 260us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0027 - val_mae: 0.0373
Epoch 106/100000
1046/1046 [==============================] - 0s 333us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0314
Epoch 107/100000
1046/1046 [==============================] - 0s 326us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0300
Epoch 108/100000
1046/1046 [==============================] - 0s 285us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 109/100000
1046/1046 [==============================] - 0s 270us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 110/100000
1046/1046 [==============================] - 0s 301us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 111/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0021 - val_mae: 0.0309
Epoch 112/100000
1046/1046 [==============================] - 0s 269us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0294
Epoch 113/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 114/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0291
Epoch 115/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0020 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 116/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0287
Epoch 117/100000
```

```
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0297
Epoch 118/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0020 - mae:
0.0297 - val_loss: 0.0024 - val_mae: 0.0345
Epoch 119/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0302
Epoch 120/100000
1046/1046 [==============================] - 0s 268us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 121/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 122/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0023 - val_mae: 0.0326
Epoch 123/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 124/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0289
Epoch 125/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0296 - val_loss: 0.0019 - val_mae: 0.0291
Epoch 126/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 127/100000
1046/1046 [==============================] - 0s 278us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0021 - val_mae: 0.0307
Epoch 128/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 129/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0295
Epoch 130/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0021 - val_mae: 0.0302
Epoch 131/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0296
Epoch 132/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0303
Epoch 133/100000
```

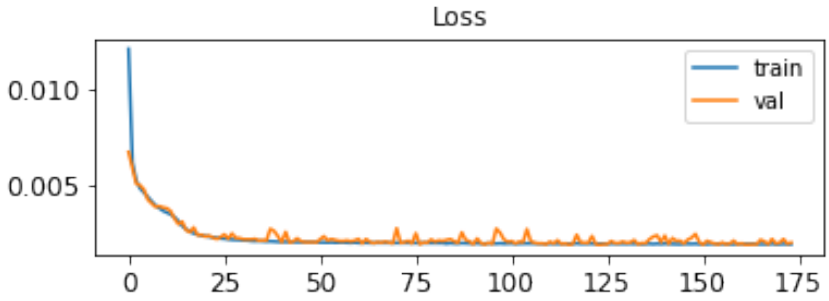
```
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0021 - val_mae: 0.0313
Epoch 134/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0020 - mae:
0.0296 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 135/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0020 - val_mae: 0.0301
Epoch 136/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 137/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0022 - val_mae: 0.0320
Epoch 138/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0023 - val_mae: 0.0321
Epoch 139/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0024 - val_mae: 0.0346
Epoch 140/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0020 - val_mae: 0.0306
Epoch 141/100000
1046/1046 [==============================] - 0s 277us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0024 - val_mae: 0.0341
Epoch 142/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 143/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0022 - val_mae: 0.0326
Epoch 144/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 145/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0021 - val_mae: 0.0319
Epoch 146/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0020 - val_mae: 0.0299
Epoch 147/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0022 - val_mae: 0.0313
Epoch 148/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0023 - val_mae: 0.0329
Epoch 149/100000
```

```
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0025 - val_mae: 0.0343
Epoch 150/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0020 - val_mae: 0.0305
Epoch 151/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0289
Epoch 152/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0021 - val_mae: 0.0306
Epoch 153/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0301
Epoch 154/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 155/100000
1046/1046 [==============================] - 0s 272us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0291
Epoch 156/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0020 - val_mae: 0.0300
Epoch 157/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 158/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0020 - val_mae: 0.0297
Epoch 159/100000
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 160/100000
1046/1046 [==============================] - 0s 287us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0019 - val_mae: 0.0292
Epoch 161/100000
1046/1046 [==============================] - 0s 276us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0298
Epoch 162/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0286
Epoch 163/100000
1046/1046 [==============================] - 0s 290us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0019 - val_mae: 0.0292
Epoch 164/100000
1046/1046 [==============================] - 0s 279us/step - loss: 0.0019 - mae:
0.0296 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 165/100000
```

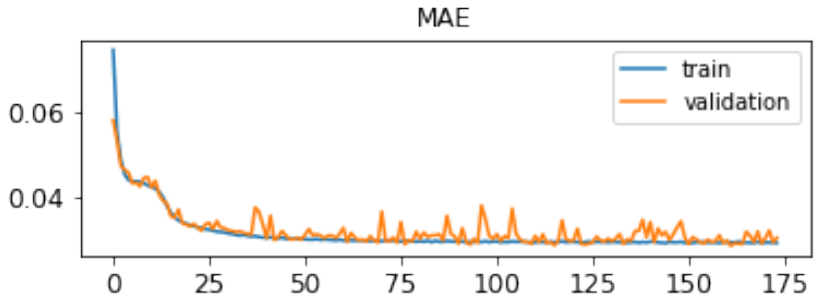
```
1046/1046 [==============================] - 0s 273us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0019 - val_mae: 0.0290
Epoch 166/100000
1046/1046 [==============================] - 0s 274us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0021 - val_mae: 0.0318
Epoch 167/100000
1046/1046 [==============================] - 0s 275us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0021 - val_mae: 0.0310
Epoch 168/100000
1046/1046 [==============================] - 0s 317us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 169/100000
1046/1046 [==============================] - 0s 290us/step - loss: 0.0019 - mae:
0.0295 - val_loss: 0.0022 - val_mae: 0.0319
Epoch 170/100000
1046/1046 [==============================] - 0s 281us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0020 - val_mae: 0.0291
Epoch 171/100000
1046/1046 [==============================] - 0s 292us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0020 - val_mae: 0.0303
Epoch 172/100000
1046/1046 [==============================] - 0s 283us/step - loss: 0.0019 - mae:
0.0294 - val_loss: 0.0022 - val_mae: 0.0321
Epoch 173/100000
1046/1046 [==============================] - 0s 280us/step - loss: 0.0019 - mae:
0.0293 - val_loss: 0.0019 - val_mae: 0.0294
Epoch 174/100000
1046/1046 [==============================] - 0s 288us/step - loss: 0.0019 - mae:
0.0292 - val_loss: 0.0020 - val_mae: 0.0304
```

[19]:
```
from matplotlib import pyplot

# plot loss during training

pyplot.subplot(211)
pyplot.title('Loss')
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='val')
pyplot.legend()
```

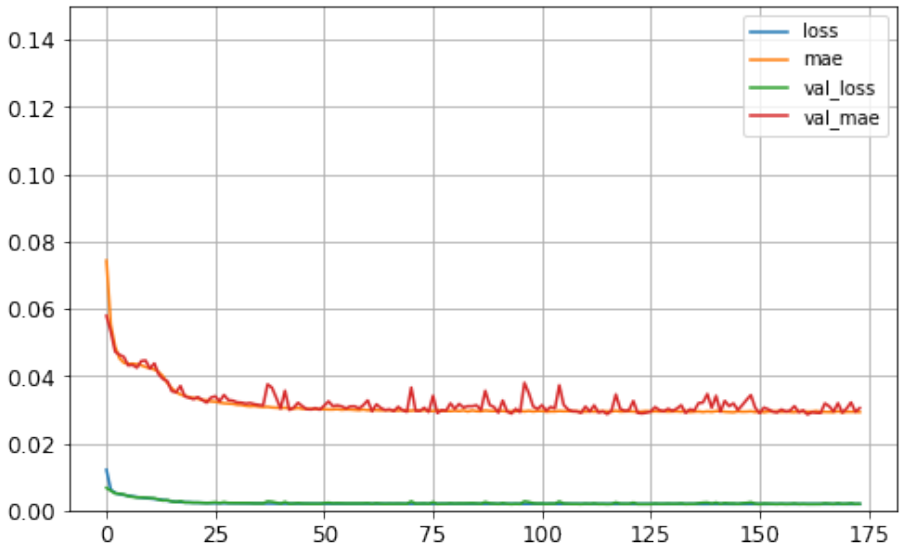
[19]:
```
<matplotlib.legend.Legend at 0x157223220>
```




[20]:
```
# plot absolute error during training

pyplot.subplot(212)
pyplot.title('MAE')
pyplot.plot(history.history['mae'], label='train')
pyplot.plot(history.history['val_mae'], label='validation')
pyplot.legend()
pyplot.show()
```




[21]:
```
# plot learning curve using pandas

pd.DataFrame(history.history).plot(figsize=(8, 5))
plt.grid(True)
plt.gca().set_ylim(0, 0.150) # set the vertical range to [0-1]
plt.show()
```




## 3 Evaluate the model

[22]:
```
# load test set
test_data = pd.read_csv('data/unprocessed_strat_test_data.csv', index_col=0)
```

[23]:
```
# normalize test data

from sklearn.preprocessing import MinMaxScaler

# define min max scaler
scaler = MinMaxScaler()

# define data to be scaled
normalize_these = ['T1', 'deltaP', 'Pamb', 'Vwsat',
                   'Twsatin', 'TwIn', 'Pgen', 'N','TOT',
                   'COT', 'COP', 'RITa', 'CCIT1', 'EIT1',
                   'EIT2']

# transform data
test_data[normalize_these] = scaler.fit_transform(test_data[normalize_these])
```

[24]:
```
test_data
```

[24]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 6 | 0.055897 | 0.671608 | 0.000000 | 0.004695 | 0.000000 | 0.000000 | 0.568550 | |
| 9 | 0.055897 | 0.663082 | 0.000315 | 0.004695 | 0.003448 | 0.000000 | 0.568550 | |
| 11 | 0.055897 | 0.688156 | 0.000524 | 0.004695 | 0.003448 | 0.000000 | 0.554804 | |
| 13 | 0.055897 | 0.738010 | 0.000760 | 0.004695 | 0.006897 | 0.000000 | 0.527744 | |
| 16 | 0.055897 | 0.728026 | 0.001075 | 0.004695 | 0.010345 | 0.003155 | 0.533213 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68577 | 0.391280 | 0.773572 | 1.000000 | 0.516432 | 0.817241 | 0.801262 | 0.394243 | |
| 68579 | 0.391280 | 0.873056 | 1.000000 | 0.521127 | 0.820690 | 0.801262 | 0.432890 | |
| 68583 | 0.288988 | 0.824699 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.439007 | |
| 68584 | 0.280045 | 0.803485 | 1.000000 | 0.525822 | 0.820690 | 0.801262 | 0.439007 | |
| 68587 | 0.325321 | 0.750805 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.439007 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 6 | 0.936372 | 0.285435 | 0.083770 | 1.00 | 0.865801 | 0.324138 | 0.760345 | |
| 9 | 0.936372 | 0.296852 | 0.089005 | 1.00 | 0.863636 | 0.335632 | 0.762069 | |
| 11 | 0.935999 | 0.311287 | 0.094241 | 1.00 | 0.861472 | 0.340230 | 0.763793 | |
| 13 | 0.934714 | 0.328821 | 0.099476 | 1.00 | 0.863636 | 0.344828 | 0.765517 | |
| 16 | 0.927944 | 0.337628 | 0.099476 | 1.00 | 0.861472 | 0.354023 | 0.770690 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68577 | 0.679291 | 0.599821 | 0.670157 | 0.70 | 0.080087 | 0.645977 | 0.167241 | |
| 68579 | 0.682321 | 0.593541 | 0.670157 | 0.70 | 0.077922 | 0.650575 | 0.167241 | |
| 68583 | 0.684861 | 0.599087 | 0.670157 | 0.72 | 0.080087 | 0.645977 | 0.167241 | |
| 68584 | 0.685606 | 0.601289 | 0.670157 | 0.70 | 0.080087 | 0.645977 | 0.167241 | |
| 68587 | 0.690805 | 0.607894 | 0.664921 | 0.72 | 0.075758 | 0.645977 | 0.167241 | |

| | EIT2 |
|---|---|
| 6 | 0.680162 |
| 9 | 0.682186 |
| 11 | 0.682186 |
| 13 | 0.678138 |
| 16 | 0.680162 |
| ... | ... |
| 68577 | 0.125506 |
| 68579 | 0.125506 |
| 68583 | 0.125506 |
| 68584 | 0.127530 |
| 68587 | 0.129555 |

[13718 rows x 15 columns]

[25]:
```
X_test = test_data
```

[26]:
```
autoencoder.evaluate(X_test, X_test)
```

```
429/429 [==============================] - 0s 195us/step - loss: 0.0019 - mae:
0.0291
```

[26]: [0.0018976136343553662, 0.029116839170455933]

[ ]:

# Testing

June 7, 2022

## 1 Part III: Testing

Individual file made to: - Load test data - Load model.h5 with weights

Evaluate the different models, and make predictions with these readings.

[1]:
```
# Python 3.5 is required
import sys
assert sys.version_info >= (3, 5)

# Is this notebook running on Colab or Kaggle?
IS_COLAB = "google.colab" in sys.modules
IS_KAGGLE = "kaggle_secrets" in sys.modules

# Scikit-Learn 0.20 is required
import sklearn
assert sklearn.__version__ >= "0.20"

# TensorFlow 2.0 is required
import tensorflow as tf
from tensorflow import keras
assert tf.__version__ >= "2.0"

if not tf.config.list_physical_devices('GPU'):
    print("No GPU was detected. LSTMs and CNNs can be very slow without a GPU.")
    if IS_COLAB:
        print("Go to Runtime > Change runtime and select a GPU hardware␣
 ↪accelerator.")
    if IS_KAGGLE:
        print("Go to Settings > Accelerator and select GPU.")

# Common imports
import numpy as np
import os

# to make this notebook's output stable across runs
np.random.seed(42)
tf.random.set_seed(42)
```

```
# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "HPtweaking"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)
```

```
Init Plugin
Init Graph Optimizer
Init Kernel
```

[2]:
```
# disable GPU for faster calculations
# for bigger batch sized (=>1024) GPU will be faster

tf.config.set_visible_devices([], 'GPU')
```

[3]:
```
# load test set

import pandas as pd
test_data = pd.read_csv('data/unprocessed_strat_test_data.csv', index_col=0)
```

[4]:
```
test_data.head()
```

[4]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | N | TOT | \ |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 4.2 | 87.657 | 0.074415 | -0.01 | 33.4 | 30.5 | 72.500 | 63840.000 | 606.234 | |
| 9 | 4.2 | 87.014 | 0.074415 | -0.01 | 33.5 | 30.5 | 72.500 | 63840.000 | 606.374 | |
| 11 | 4.2 | 88.905 | 0.074415 | -0.01 | 33.5 | 30.5 | 72.309 | 63838.258 | 606.551 | |
| 13 | 4.2 | 92.665 | 0.074415 | -0.01 | 33.6 | 30.5 | 71.933 | 63832.270 | 606.766 | |
| 16 | 4.2 | 91.912 | 0.074415 | -0.01 | 33.7 | 30.6 | 72.009 | 63800.720 | 606.874 | |

| | COT | COP | RITa | CCIT1 | EIT1 | EIT2 |
|---|---|---|---|---|---|---|
| 6 | 153.5 | 2.97 | 99.6 | 533.9 | 143.1 | 140.9 |
| 9 | 153.6 | 2.97 | 99.5 | 534.4 | 143.2 | 141.0 |

```
11  153.7  2.97  99.4  534.6  143.3  141.0
13  153.8  2.97  99.5  534.8  143.4  140.8
16  153.8  2.97  99.4  535.2  143.7  140.9
```

#plot features

%matplotlib inline test_data.plot(subplots=True, layout=(4,4), figsize=(15,10)) save_fig("attribute_plots")

[5]:
```
# normalize test data

from sklearn.preprocessing import MinMaxScaler

# define min max scaler
scaler = MinMaxScaler()

# define data to be scaled
normalize_these = ['T1', 'deltaP', 'Pamb', 'Vwsat',
                   'Twsatin', 'TwIn', 'Pgen', 'N','TOT',
                   'COT', 'COP', 'RITa', 'CCIT1', 'EIT1',
                   'EIT2']

# transform data
test_data[normalize_these] = scaler.fit_transform(test_data[normalize_these])
```

[6]:
```
test_data
```

[6]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 6 | 0.055897 | 0.671608 | 0.000000 | 0.004695 | 0.000000 | 0.000000 | 0.568550 | |
| 9 | 0.055897 | 0.663082 | 0.000315 | 0.004695 | 0.003448 | 0.000000 | 0.568550 | |
| 11 | 0.055897 | 0.688156 | 0.000524 | 0.004695 | 0.003448 | 0.000000 | 0.554804 | |
| 13 | 0.055897 | 0.738010 | 0.000760 | 0.004695 | 0.006897 | 0.000000 | 0.527744 | |
| 16 | 0.055897 | 0.728026 | 0.001075 | 0.004695 | 0.010345 | 0.003155 | 0.533213 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68577 | 0.391280 | 0.773572 | 1.000000 | 0.516432 | 0.817241 | 0.801262 | 0.394243 | |
| 68579 | 0.391280 | 0.873056 | 1.000000 | 0.521127 | 0.820690 | 0.801262 | 0.432890 | |
| 68583 | 0.288988 | 0.824699 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.439007 | |
| 68584 | 0.280045 | 0.803485 | 1.000000 | 0.525822 | 0.820690 | 0.801262 | 0.439007 | |
| 68587 | 0.325321 | 0.750805 | 1.000000 | 0.530516 | 0.820690 | 0.801262 | 0.439007 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 6 | 0.936372 | 0.285435 | 0.083770 | 1.00 | 0.865801 | 0.324138 | 0.760345 | |
| 9 | 0.936372 | 0.296852 | 0.089005 | 1.00 | 0.863636 | 0.335632 | 0.762069 | |
| 11 | 0.935999 | 0.311287 | 0.094241 | 1.00 | 0.861472 | 0.340230 | 0.763793 | |
| 13 | 0.934714 | 0.328821 | 0.099476 | 1.00 | 0.863636 | 0.344828 | 0.765517 | |
| 16 | 0.927944 | 0.337628 | 0.099476 | 1.00 | 0.861472 | 0.354023 | 0.770690 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 68577 | 0.679291 | 0.599821 | 0.670157 | 0.70 | 0.080087 | 0.645977 | 0.167241 | |

```
68579  0.682321  0.593541  0.670157  0.70  0.077922  0.650575  0.167241
68583  0.684861  0.599087  0.670157  0.72  0.080087  0.645977  0.167241
68584  0.685606  0.601289  0.670157  0.70  0.080087  0.645977  0.167241
68587  0.690805  0.607894  0.664921  0.72  0.075758  0.645977  0.167241

           EIT2
6      0.680162
9      0.682186
11     0.682186
13     0.678138
16     0.680162
...         ...
68577  0.125506
68579  0.125506
68583  0.125506
68584  0.127530
68587  0.129555

[13718 rows x 15 columns]
```

[7]:
```
X_test = test_data
```

[8]:
```
model = keras.models.load_model("Keras.h5_models/ae_keras_model_3-6.h5")
```

[9]:
```
model.evaluate(X_test, X_test)
```

```
429/429 [==============================] - 0s 194us/step - loss: 0.0018 - mae:
0.0283

2022-06-07 14:34:48.622091: I
tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR
Optimization Passes are enabled (registered 2)
2022-06-07 14:34:48.622240: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
```

[9]:
```
[0.001815290073864162, 0.02828710898756981]
```

## 2 Predict

[10]:
```
# use X_test data to make predictions

prediction = model.predict(X_test)
```

[11]:
```
prediction
```

[11]:
```
array([[ 0.02983552,  0.77770114, -0.18384627, ...,  0.5908542 ,
         0.73574334,  0.687926  ],
```

```
       [ 0.03215212,  0.77497536, -0.18029156, ...,  0.5921429 ,
         0.7371589 ,  0.68956214],
       [ 0.03471732,  0.77433187, -0.17627841, ...,  0.59538627,
         0.7392304 ,  0.6918317 ],
       ...,
       [ 0.32124677,  0.7725279 ,  0.99784327, ...,  0.64910287,
         0.1727834 ,  0.11896542],
       [ 0.32032055,  0.7656436 ,  0.9974869 , ...,  0.6457251 ,
         0.17085572,  0.11741717],
       [ 0.3445112 ,  0.73623514,  0.9953272 , ...,  0.6474879 ,
         0.1705122 ,  0.1193455 ]], dtype=float32)
```

[12]:
```
# Transform prediction from array to pandas dataframe

new_pred = pd.DataFrame(prediction, columns = ['T1', 'deltaP', 'Pamb', 'Vwsat',
                                               'Twsatin', 'TwIn', 'Pgen',␣
 ↪'N','TOT',
                                               'COT', 'COP', 'RITa', 'CCIT1',␣
 ↪'EIT1',
                                               'EIT2'])
```

[13]:
```
new_pred
```

[13]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | Pgen | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.029836 | 0.777701 | -0.183846 | 0.005558 | -0.001267 | 0.013378 | 0.247462 | |
| 1 | 0.032152 | 0.774975 | -0.180292 | 0.005650 | 0.000154 | 0.014244 | 0.247796 | |
| 2 | 0.034717 | 0.774332 | -0.176278 | 0.005747 | 0.001082 | 0.014574 | 0.248410 | |
| 3 | 0.037210 | 0.775621 | -0.171459 | 0.005859 | 0.001932 | 0.014747 | 0.249217 | |
| 4 | 0.038438 | 0.773193 | -0.169673 | 0.005908 | 0.002918 | 0.015439 | 0.249292 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 13713 | 0.376725 | 0.756784 | 0.996698 | 0.468293 | 0.653505 | 0.634777 | 0.441525 | |
| 13714 | 0.371364 | 0.801496 | 0.999627 | 0.425008 | 0.654937 | 0.635656 | 0.431595 | |
| 13715 | 0.321247 | 0.772528 | 0.997843 | 0.463099 | 0.669182 | 0.650599 | 0.437188 | |
| 13716 | 0.320321 | 0.765644 | 0.997487 | 0.469088 | 0.670245 | 0.651858 | 0.438180 | |
| 13717 | 0.344511 | 0.736235 | 0.995327 | 0.496024 | 0.661360 | 0.642915 | 0.446352 | |

| | N | TOT | COT | COP | RITa | CCIT1 | EIT1 | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.782192 | 0.304519 | 0.286109 | 0.799265 | 0.883091 | 0.590854 | 0.735743 | |
| 1 | 0.782794 | 0.305988 | 0.288673 | 0.799756 | 0.883175 | 0.592143 | 0.737159 | |
| 2 | 0.786726 | 0.307299 | 0.294449 | 0.803639 | 0.883328 | 0.595386 | 0.739230 | |
| 3 | 0.793717 | 0.308204 | 0.303173 | 0.810643 | 0.883637 | 0.600390 | 0.742025 | |
| 4 | 0.792640 | 0.309123 | 0.303255 | 0.809476 | 0.883646 | 0.600289 | 0.742535 | |
| ... | ... | ... | ... | ... | ... | ... | ... | |
| 13713 | 0.653397 | 0.469130 | 0.627312 | 0.671032 | 0.051216 | 0.652332 | 0.168549 | |
| 13714 | 0.670425 | 0.466279 | 0.634371 | 0.687531 | 0.063753 | 0.660927 | 0.173029 | |
| 13715 | 0.671947 | 0.424230 | 0.624209 | 0.692468 | 0.056838 | 0.649103 | 0.172783 | |
| 13716 | 0.666502 | 0.423622 | 0.619756 | 0.687110 | 0.054540 | 0.645725 | 0.170856 | |

```
13717  0.659191  0.441395  0.625841  0.679215  0.047003  0.647488  0.170512

           EIT2
0      0.687926
1      0.689562
2      0.691832
3      0.694819
4      0.695462
...         ...
13713  0.115463
13714  0.116748
13715  0.118965
13716  0.117417
13717  0.119346

[13718 rows x 15 columns]
```

[14]:
```
# Undo the scaling of X according to feature_range.

X_trans = scaler.inverse_transform(new_pred)
```

[15]:
```
X_trans
```

[15]:
```
array([[4.1533756e+00, 9.5658447e+01, 7.4407756e-02, ..., 5.4550214e+02,
        1.4167311e+02, 1.4128355e+02],
       [4.1575203e+00, 9.5452866e+01, 7.4407898e-02, ..., 5.4555823e+02,
        1.4175522e+02, 1.4136436e+02],
       [4.1621094e+00, 9.5404335e+01, 7.4408047e-02, ..., 5.4569928e+02,
        1.4187537e+02, 1.4147649e+02],
       ...,
       [4.6747108e+00, 9.5268280e+01, 7.4452847e-02, ..., 5.4803595e+02,
        1.0902144e+02, 1.1317689e+02],
       [4.6730537e+00, 9.4749077e+01, 7.4452832e-02, ..., 5.4788904e+02,
        1.0890964e+02, 1.1310040e+02],
       [4.7163305e+00, 9.2531120e+01, 7.4452750e-02, ..., 5.4796576e+02,
        1.0888971e+02, 1.1319566e+02]], dtype=float32)
```

[16]:
```
# Transform prediction from array to pandas

new_pred = pd.DataFrame(X_trans, columns = ['T1', 'deltaP', 'Pamb', 'Vwsat',
                                            'Twsatin', 'TwIn', 'Pgen',␣
 ↪'N','TOT',
                                            'COT', 'COP', 'RITa', 'CCIT1',␣
 ↪'EIT1',
                                            'EIT2'])
```

[17]:
```
new_pred
```

[17]:

| | T1 | deltaP | Pamb | Vwsat | Twsatin | TwIn | \ |
|---|---|---|---|---|---|---|---|
| 0 | 4.153376 | 95.658447 | 0.074408 | -0.008161 | 33.363251 | 30.924086 | |
| 1 | 4.157520 | 95.452866 | 0.074408 | -0.007965 | 33.404480 | 30.951527 | |
| 2 | 4.162109 | 95.404335 | 0.074408 | -0.007758 | 33.431389 | 30.962006 | |
| 3 | 4.166569 | 95.501587 | 0.074408 | -0.007520 | 33.456024 | 30.967468 | |
| 4 | 4.168765 | 95.318436 | 0.074408 | -0.007416 | 33.484612 | 30.989410 | |
| ... | ... | ... | ... | ... | ... | ... | |
| 13713 | 4.773962 | 94.080879 | 0.074453 | 0.977465 | 52.351639 | 50.622425 | |
| 13714 | 4.764370 | 97.453026 | 0.074453 | 0.885267 | 52.393169 | 50.650307 | |
| 13715 | 4.674711 | 95.268280 | 0.074453 | 0.966401 | 52.806267 | 51.123989 | |
| 13716 | 4.673054 | 94.749077 | 0.074453 | 0.979157 | 52.837101 | 51.163891 | |
| 13717 | 4.716331 | 92.531120 | 0.074453 | 1.036531 | 52.579445 | 50.880390 | |

| | Pgen | N | TOT | COT | COP | RITa | \ |
|---|---|---|---|---|---|---|---|
| 0 | 68.038483 | 63121.398438 | 606.468018 | 157.364685 | 2.869632 | 100.398819 | |
| 1 | 68.043129 | 63124.207031 | 606.486023 | 157.413651 | 2.869878 | 100.402672 | |
| 2 | 68.051651 | 63142.535156 | 606.502075 | 157.523987 | 2.871819 | 100.409729 | |
| 3 | 68.062866 | 63175.121094 | 606.513245 | 157.690598 | 2.875322 | 100.424042 | |
| 4 | 68.063911 | 63170.097656 | 606.524414 | 157.692184 | 2.874738 | 100.424423 | |
| ... | ... | ... | ... | ... | ... | ... | |
| 13713 | 70.734993 | 62521.117188 | 608.486450 | 163.881653 | 2.805516 | 61.966187 | |
| 13714 | 70.597015 | 62600.480469 | 608.451538 | 164.016495 | 2.813766 | 62.545376 | |
| 13715 | 70.674736 | 62607.574219 | 607.935913 | 163.822388 | 2.816234 | 62.225910 | |
| 13716 | 70.688515 | 62582.199219 | 607.928467 | 163.737335 | 2.813555 | 62.119724 | |
| 13717 | 70.802055 | 62548.125000 | 608.146423 | 163.853577 | 2.809608 | 61.771530 | |

| | CCIT1 | EIT1 | EIT2 |
|---|---|---|---|
| 0 | 545.502136 | 141.673111 | 141.283554 |
| 1 | 545.558228 | 141.755219 | 141.364365 |
| 2 | 545.699280 | 141.875366 | 141.476486 |
| 3 | 545.916992 | 142.037460 | 141.624039 |
| 4 | 545.912598 | 142.067032 | 141.655792 |
| ... | ... | ... | ... |
| 13713 | 548.176453 | 108.775841 | 113.003899 |
| 13714 | 548.550354 | 109.035667 | 113.067329 |
| 13715 | 548.035950 | 109.021439 | 113.176888 |
| 13716 | 547.889038 | 108.909637 | 113.100403 |
| 13717 | 547.965759 | 108.889709 | 113.195663 |

[13718 rows x 15 columns]

#plot features

%matplotlib inline new_pred.plot(subplots=True, layout=(4,4), figsize=(15,10))

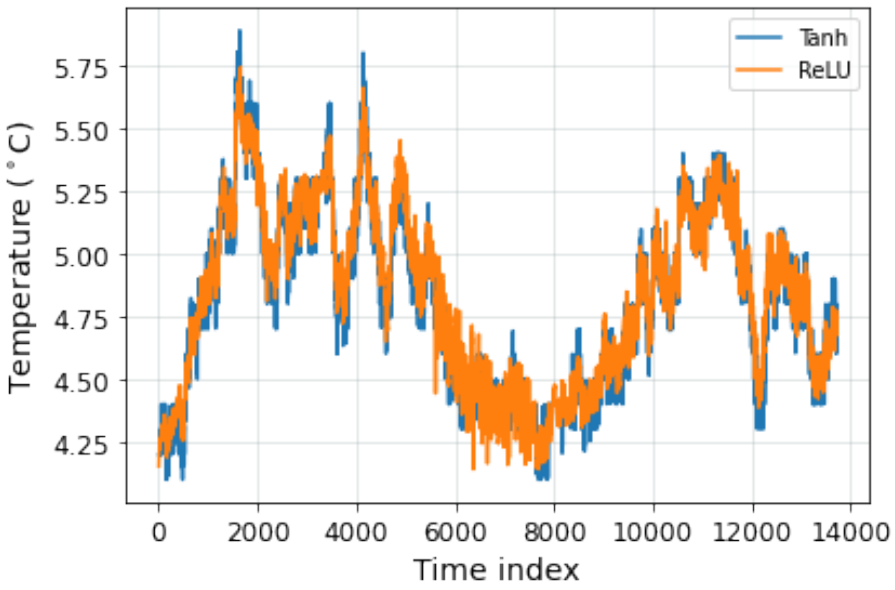
[18]:
```
new_pred.to_csv("data/Predict3-6.csv")
```

[19]:
```
data = pd.read_csv('data/Predict3-6.csv', index_col=0)

all_data = pd.read_csv('data/unprocessed_strat_test_data.csv')
```

[20]:
```
from IPython.display import Math

plt.plot(all_data['T1']) #blue
plt.plot(data['T1']) #orange
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("T1-3-6")
plt.show
```

```
Saving figure T1-3-6
```

[20]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```

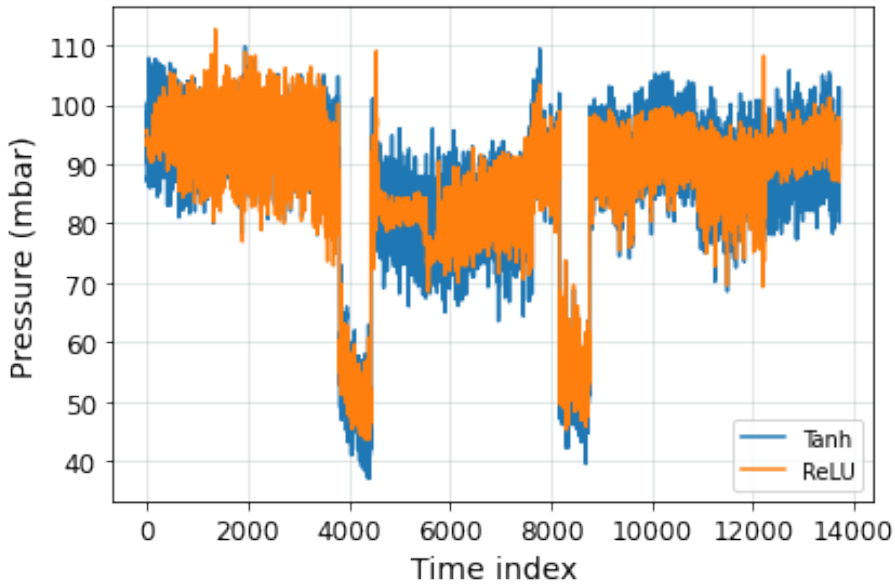



[21]:
```
plt.plot(all_data['deltaP'])
plt.plot(data['deltaP'])
plt.legend(['Tanh', 'ReLU'], loc='lower right')
plt.xlabel('Time index')
plt.ylabel("Pressure (mbar)")
```

```
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("deltaP-3-6")
plt.show
```

```
Saving figure deltaP-3-6
```

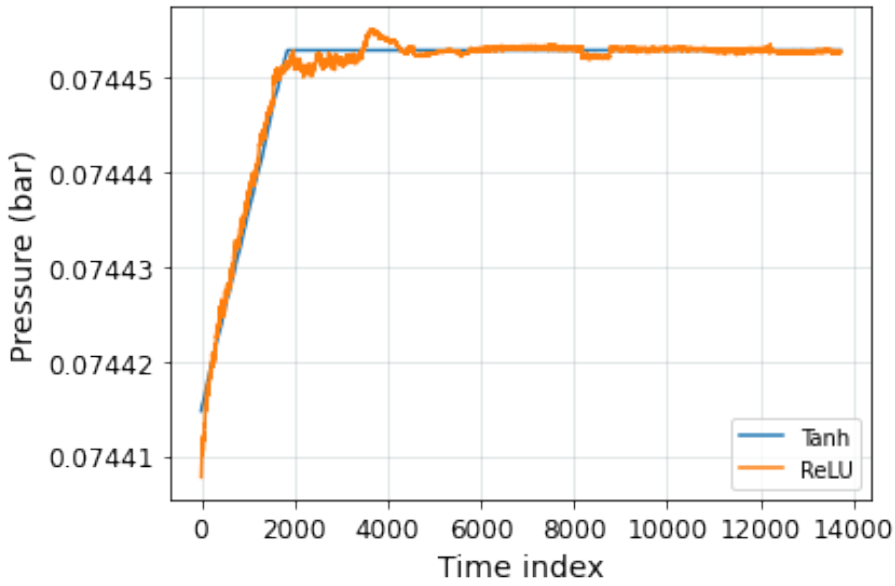
[21]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[22]:
```
plt.plot(all_data['Pamb'])
plt.plot(data['Pamb'])
plt.legend(['Tanh', 'ReLU'], loc='lower right')
plt.xlabel('Time index')
plt.ylabel("Pressure (bar)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("Pamb-3-6")
plt.show
```

```
Saving figure Pamb-3-6
```

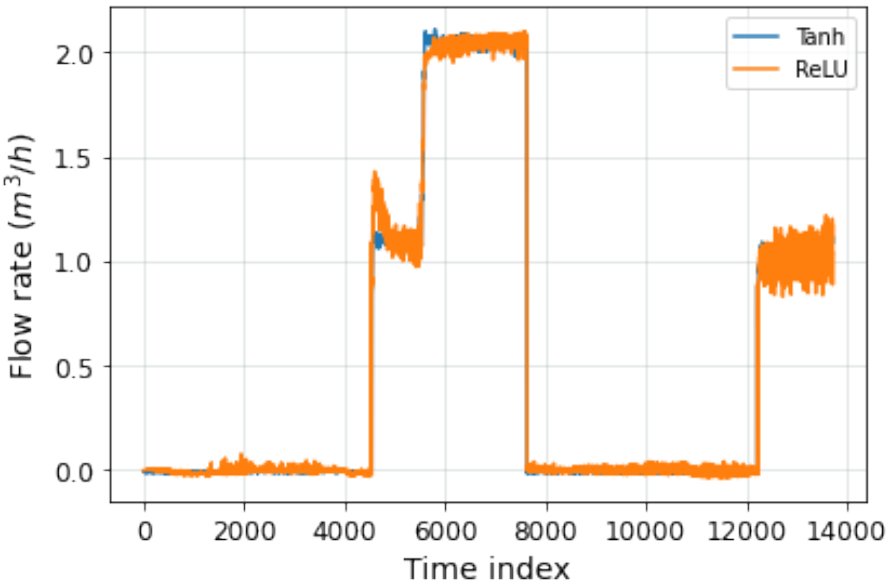
[22]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[23]:
```
plt.plot(all_data['Vwsat'])
plt.plot(data['Vwsat'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Flow rate ($m^3/h$)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("Vwsat-3-6")
plt.show
```

```
Saving figure Vwsat-3-6
```

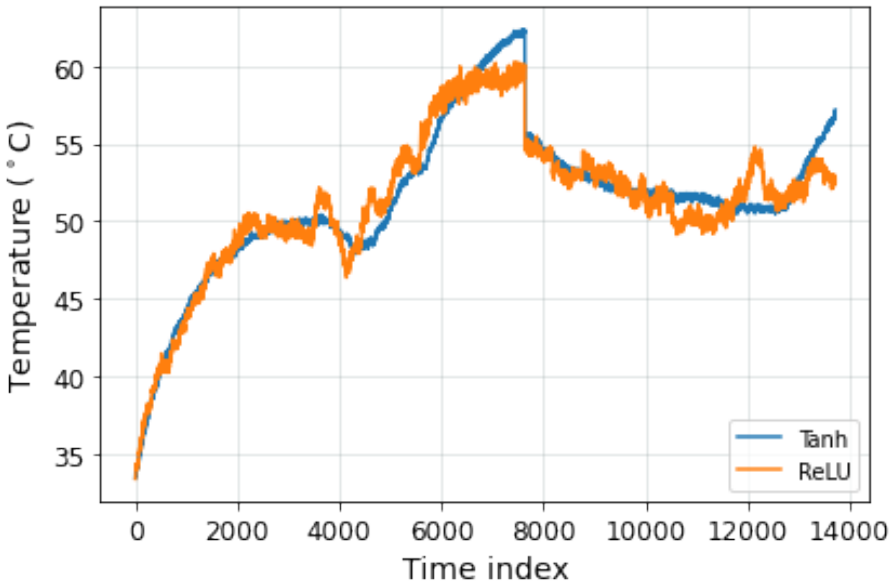
[23]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[24]:
```
plt.plot(all_data['Twsatin'])
plt.plot(data['Twsatin'])
plt.legend(['Tanh', 'ReLU'], loc='lower right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("Twsatin-3-6")
plt.show
```

```
Saving figure Twsatin-3-6
```

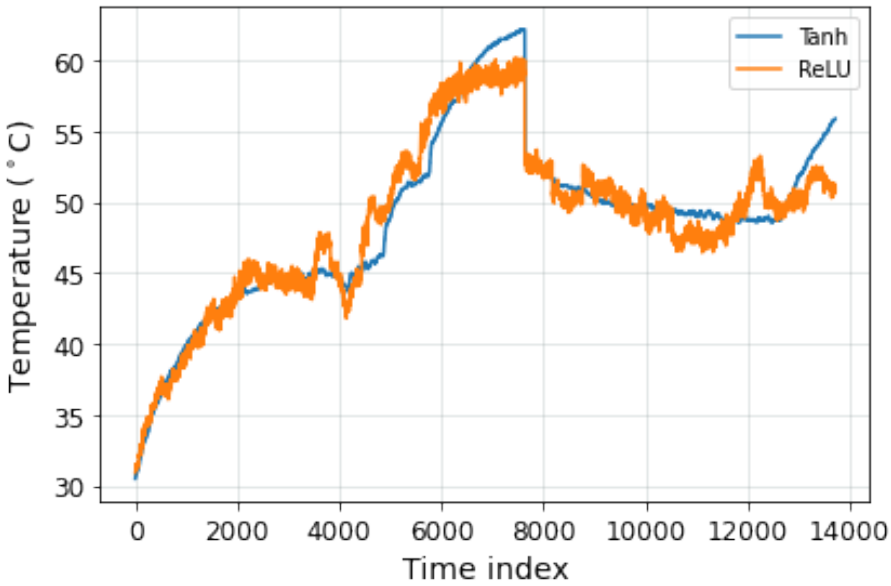
[24]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[25]:
```
plt.plot(all_data['TwIn'])
plt.plot(data['TwIn'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("TwIn-3-6")
plt.show
```

```
Saving figure TwIn-3-6
```

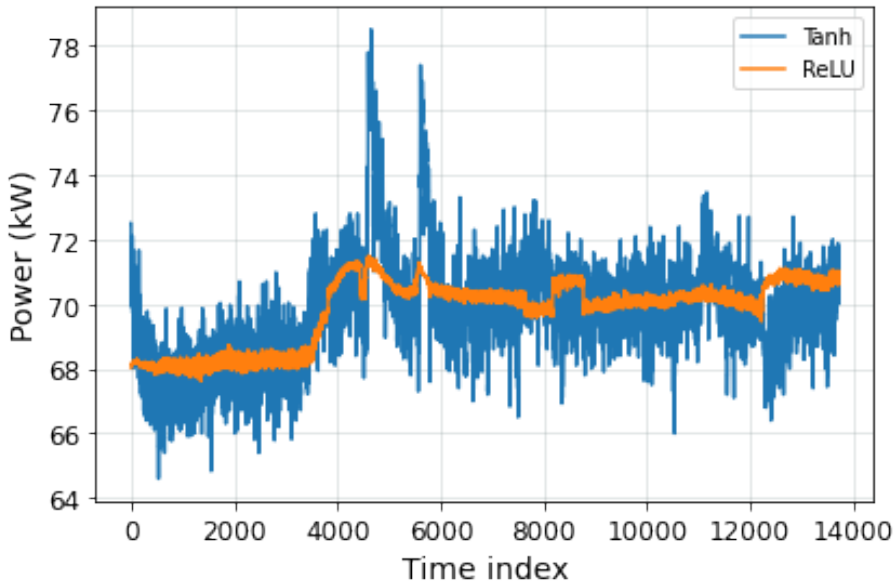
[25]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[26]:
```
plt.plot(all_data['Pgen'])
plt.plot(data['Pgen'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Power (kW)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("Pgen-3-6")
plt.show
```

```
Saving figure Pgen-3-6
```

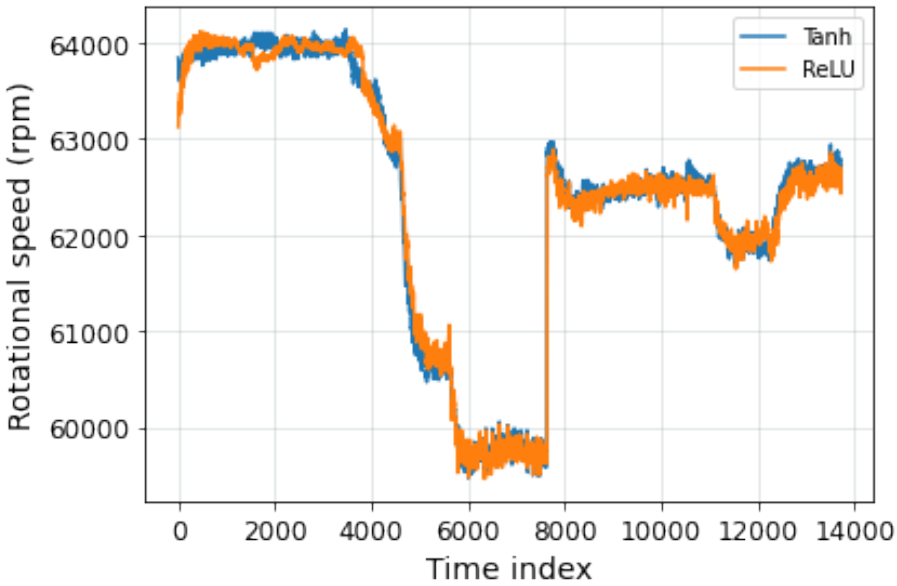
[26]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[27]:
```
plt.plot(all_data['N'])
plt.plot(data['N'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Rotational speed (rpm)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("N-3-6")
plt.show
```

```
Saving figure N-3-6
```

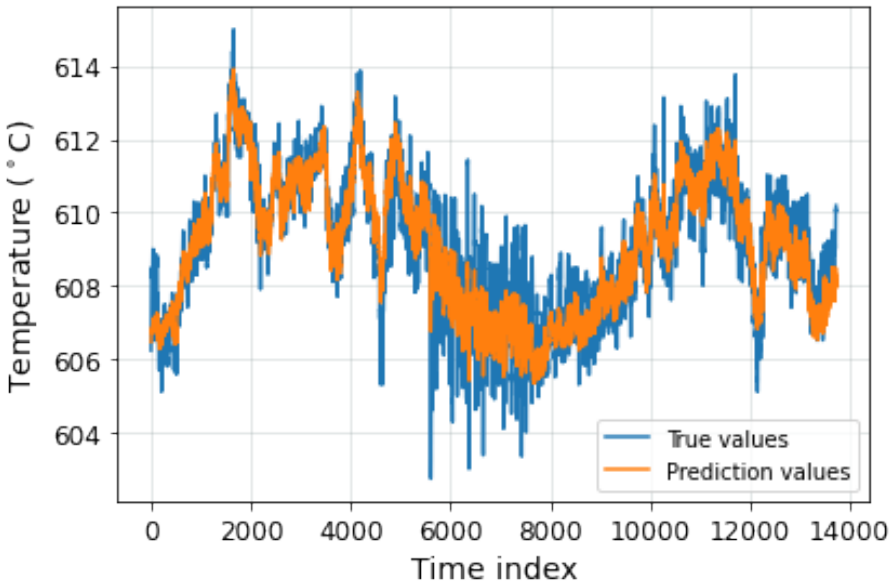
[27]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[28]:
```
plt.plot(all_data['TOT'])
plt.plot(data['TOT'])
plt.legend(['True values', 'Prediction values'], loc='lower right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("TOT-3-6")
plt.show
```

```
Saving figure TOT-3-6
```

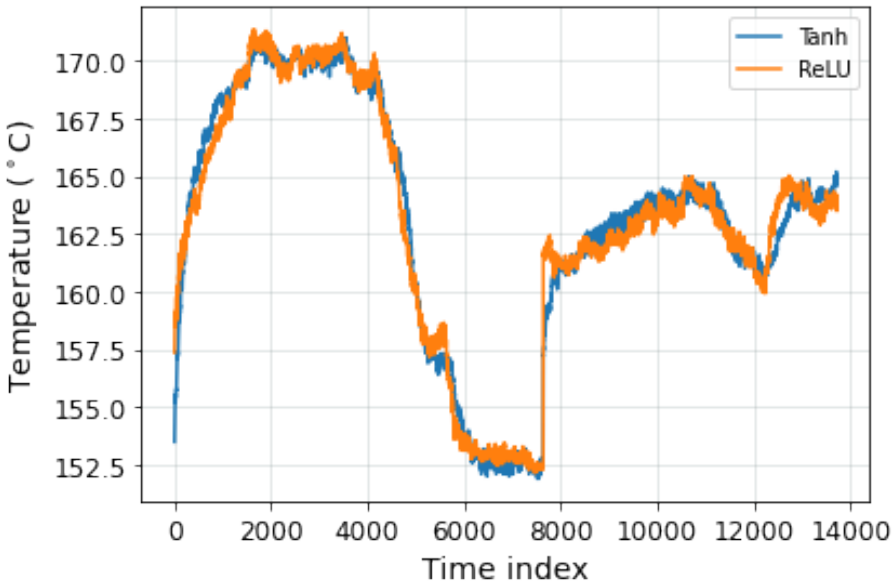
[28]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[29]:
```
plt.plot(all_data['COT'])
plt.plot(data['COT'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("COT-3-6")
plt.show
```

```
Saving figure COT-3-6
```

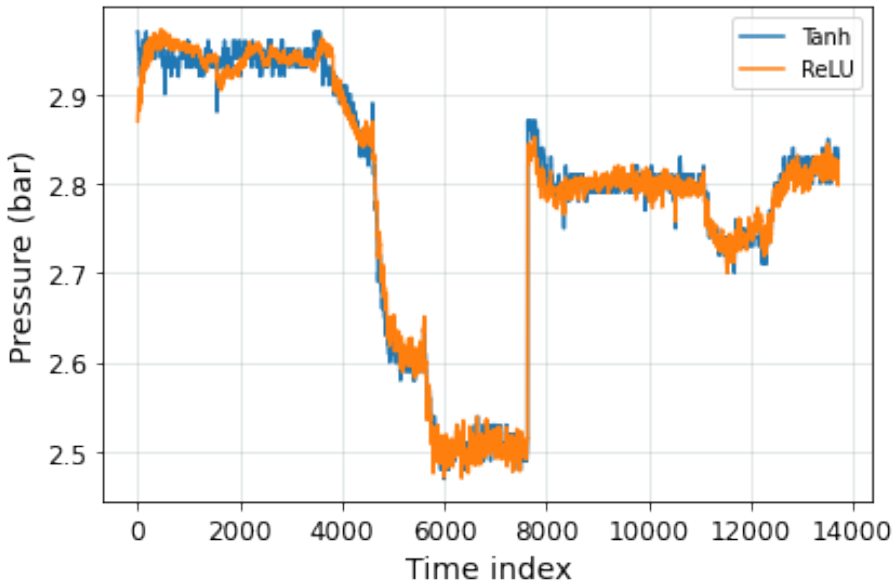
[29]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[30]:
```
plt.plot(all_data['COP'])
plt.plot(data['COP'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Pressure (bar)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("COP-3-6")
plt.show
```

```
Saving figure COP-3-6
```

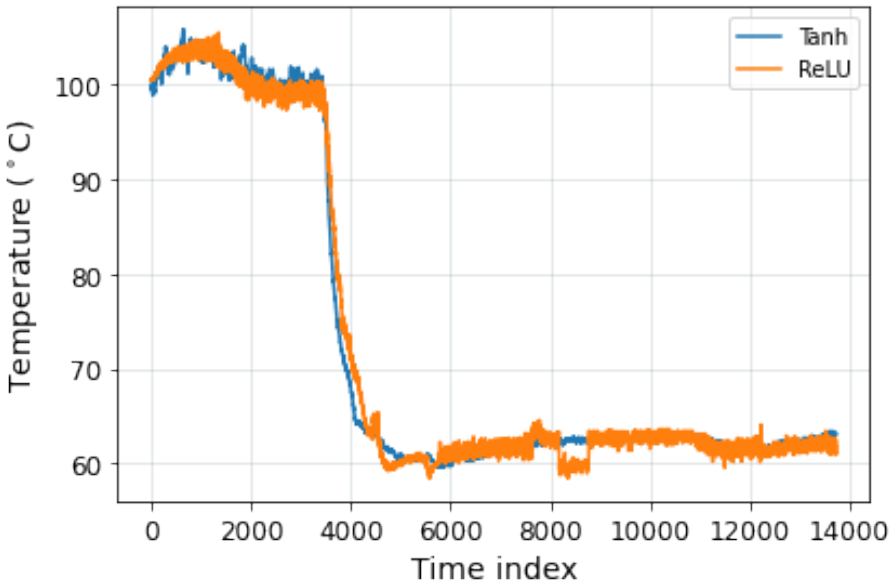
[30]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[31]:
```
plt.plot(all_data['RITa'])
plt.plot(data['RITa'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("RITa-3-6")
plt.show
```

```
Saving figure RITa-3-6
```

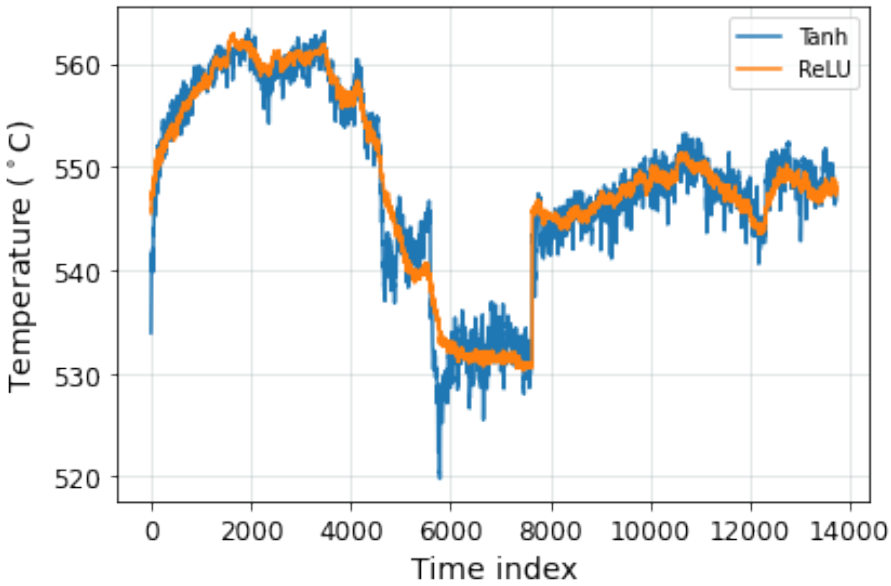
[31]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[32]:
```
plt.plot(all_data['CCIT1'])
plt.plot(data['CCIT1'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("CCIT1-3-6")
plt.show
```

```
Saving figure CCIT1-3-6
```

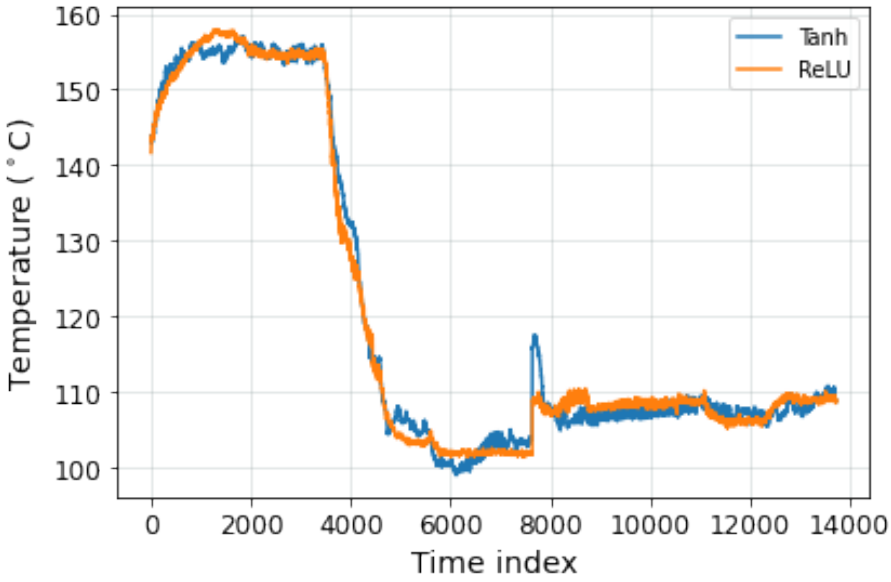
[32]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[33]:
```
plt.plot(all_data['EIT1'])
plt.plot(data['EIT1'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("EIT1-3-6")
plt.show
```

```
Saving figure EIT1-3-6
```

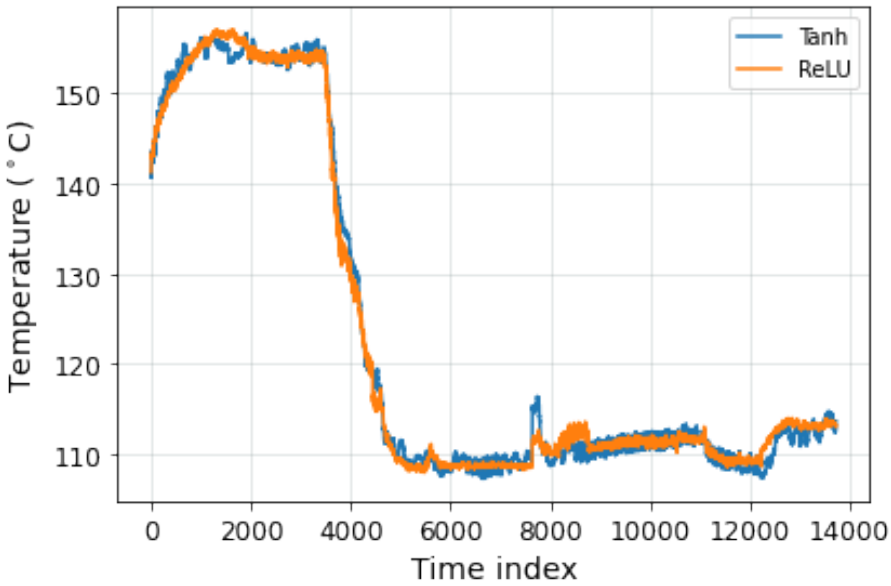
[33]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




[34]:
```
plt.plot(all_data['EIT2'])
plt.plot(data['EIT2'])
plt.legend(['Tanh', 'ReLU'], loc='upper right')
plt.xlabel('Time index')
plt.ylabel("Temperature ($^\circ$C)")
plt.grid(True, color="#93a1a1", alpha=0.3)
save_fig("EIT2-3-6")
plt.show
```

```
Saving figure EIT2-3-6
```

[34]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




### 2.1 To write over to xlsx file

load full data set

import pandas as pd $test_data = pd.read_csv('data/all_data_no_outliers.csv', index_col = 0) with pd.ExcelWriter("data/Predict.xlsx") as writer : test_data.to_excel(writer, sheet_name ='Before') new_pred.to_excel(writer, sheet_name ='After')$