Norwegian University
of Life Sciences

Master's Thesis 2019 30 ECTS
Faculty of Science and Technology

Predicting Fault Events in the Norwegian Electrical Power System using Deep Learning - A Sequential Approach

Kristian Wang Høiem
Miljøfysikk og Fornybar energi

THE NORWEGIAN UNIVERSITY OF LIFE SCIENCES

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS IN ENVIRONMENTAL PHYSICS & RENEWABLE ENERGY,
30 ECTS

Predicting Fault Events in the Norwegian Electrical Power System using Deep Learning

A Sequential Approach

By Kristian Wang Høiem

Supervisor: Heidi Samuelsen Nygård, Associate Professor at Faculty of Science and Technology NMBU

Co-supervisor: Christian Andre Andresen, SINTEF Energy Research

Co-supervisor: Bendik Nybakk Torsæter, SINTEF Energy Research

13 May 2019

Sammendrag

Etterspørselen etter energi er i stadig økning. Verdenssamfunnet samarbeider om en fremtid forsynt av ren energi fra fornybare energi kilder. Dette fører til nye krav for det eletriske kraftsystemet. For å løse overgangen må digitale teknologier implemeteres i systemet gjennom smartnett løsninger.

Denne masteroppgaven utforsket et sentralt punkt ved smartenett; nemlig predikering av feilhendelser i strømnettet. Disse spørsmålene ble forsøkt besvart: Er det mulig å predikere feilene før den oppstår? Hvor langt inn i fremtiden kan feilene bli predikert? Hvilke fysiske parametre er mest egnet til bruk som features ved predikering? Hvilen metode burde brukes til å predikere feil? Vil det være mulig å implementere dette i et sanntidsmonitoreringssystem?

Kunnskap fra elkraftdomenet og datavitenskapdomenet ble kombinert for å oppnå en kvalitativ evaluering av de utførte testene. Måledata av feiltypene avbrudd, spenningsdipp, og jordfeil, samt data hentet fra nominell drift av strømnettet ble brukt i prediksjonene. Datapunktene ble hentet fra det norske strømnettet med driftspenninger fra 15 kV to 300 kV. Evelueringen ble delt inn i flere tester. Hovedformålet var å bygge og sammenlikne resultater fra tre forskjellige tilbakekommende neurale nettverksaritekturer (RNN) trent på tidsseriedata hentet fra spenningskvalitetsmålinger (PQ). En sekvens-til-sekvens-Autoencoder ble foreslått for bruk ved signal-featureisolering. Forskjellige tester fra utforsing av rådata, til analyse av output data fra modellen ble utført. Resultatene viser at det er mulig å predikere feilene inntil sju minutter inn i fremtiden. Enda lengre predisjonshorisonter ble foreslått. Videre undersøkelser av harmoniskekomponenter relatert til signalanalyse og statistikk anbefales for bedre featureisolering. De mest lovende featurene var de harmoniskeomponentene til spenning og strøm. De forskjellige feiltypene kan ha en sammensettning av harmoniskekomponenter som gjør at hver feiltype har sin unike signatur. Det er foreslått å videreutvikle modellen med fokus på avviksdeteksjon. I kombinasjon med annet monitoreringsutstyr kan et feilhendelsesprediksjonssystem brukes som et verktøy i beslutningstaking.

Som en del av et KPN prosjekt wil denne masteroppgaven bidra til å danne grunnlaget for videre forskning på området som er beskrevet. For å plassere forskningen i et større perspektiv kan resultatene føre til økt forsyningssikkerhet, reduserte kostnader, og indirekte redusere miljøpåvirkningene ved å tilrettelegge en sikkrere integrering av bærekraftige energikilder.

Summary

The demand for energy is steadily increasing. The global community is working towards a society supplied by clean energy from renewable sources. This leads to new requirements for the electrical power system. To solve the transition, digital technologies need to be implemented in the system through smart grid solutions. This master's thesis explored one central aspect of smart grid; prediction of fault occurrences in the power grid. These questions were attempted to be answered: Is it possible to predict the faults before they happen? How long in advance can they be predicted, if the first question turns out positive? What kind of physical parameters are most suitable to use as features for prediction? What kind of method should be used to predict the faults? Will it be possible to implement this in a real-time monitoring system?

Knowledge from the electrical power system domain and the data science domain were combined to obtain a qualitative evaluation of the tests conducted. Measurement data of the fault types interruption, voltage dip, and earth fault, as well as data gathered from nominal operated power grid were be used in the predictions. The samples were collected from Norwegian power grids operated at voltage levels ranging from 15 kV to 300 kV. The assessment was be divided into multiple tests. The objective in focus was to build and compare results from three different recurrent neural network (RNN) architectures trained on time-series data acquired from power quality (PQ) measurements.

A sequence-to-sequence Autoencoder was proposed for use in signal feature extraction. Various tests were conducted from investigating the raw data, to analysing the output of the model. Results have shown a prediction horizon up to seven minutes is possible. It was proposed that even longer horizons may be plausible. Further investigation into the harmonic components was proposed, related to signal analysis and statistics, for better feature extraction. The most promising features were the harmonic components of voltage and current. The fault types may have various composition of harmonic components giving the different fault types an unique signature. Improvements to the model have been proposed, focusing on anomaly detection. In combination with other monitoring equipment, a fault event prediction system can be used as a tool in decision making.

As part of a competence building research program this thesis contributes to the foundation of further research on the area outlined. Placing the research in a broader view, the results may lead to increased security of power supply, reduced operation and maintenance (O & M) costs, and indirectly reducing the impact on the environment by enabling a safer integration of sustainable energy sources.

Preface

This master's thesis concludes my five-year graduate degree in Environmental physics and Renewable energy at the Norwegian University of Life Sciences (NMBU). These years have thought me a lot about the challenges the global community are facing, and strategies on how these challenges may be tackled. It became clear to me how dependent our society is on electricity. I decided to do my specialisation in digital electrical power systems when I got to know about the condition of the electrical power grid, and its challenges related to the increasing demand of electricity and the integration of renewable energy generation sources. I am glad I could write the thesis on a relevant topic, and I am very much looking forward to get started on my position as Energytrainee this autumn.

There are several I want to thank. First of all, I would like to thank SINTEF Energy Research for making this master's thesis possible, and for offering me a work space. My supervisors Heidi Samuelsen Nygård (NMBU) , Christian Andre Andresen (SINTEF Energy Research), and Bendik Nybakk Torsæter (SINTEF Energy Research), who believed in this thesis, took their time in supervising me and shared their knowledge. I am also grateful for being a part of the EarlyWarn* project; the discussions during the project meetings were highly valuable. Thanks to Helge Seljeseth and Volker Hoffmann for correspondence and providing, respectively, valuable insight into the power systems domain and the data science domain. I also want to thank people who have supported me during the master's degree, you know who you are. Last but not least, my family, who have supported me through this master's thesis with encouraging words and food.

Trondheim, 9 May 2019

Kristian Wang Høiem

*EarlyWarn is a KPN project (part of the ENERGIX programme) that is partly financed by the Research Council of Norway, Statnett, Haugaland Kraft Nett, NTE Nett, Lyse Elnett, Nettalliansen, Hydro Energi and NTNU. R&D partners in the projects are SINTEF Energy Research (project lead), SINTEF Digital and NTNU.

Table of Contents

List of Tables

List of Figures

List of Code

Abbreviations

a	=	phase a
AC	=	Alternating current
ASD	=	Adjustable speed drives
API	=	Application programming interface
ANN	=	Artificial neural network
AUC	=	Area under the curve
b	=	phase b
c	=	phase c
DC	=	Direct current
f	=	Frequency of a periodic waveform
FN	=	False negative
FP	=	False positive
FPR	=	False positive rate
GUI	=	Graphical user interface
GPU	=	Graphical processing unit
GRU	=	Gated recurrent unit
$i(t)$	=	Immediate current
I	=	Amplitude of current
I_a	=	Amplitude of current in phase a
I_{a0}	=	Zero sequence current in phase a
I_{a1}	=	Positive sequence current in phase a
I_{a2}	=	Negative sequence current in phase a
I_1	=	Fundamental current
I_n	=	Harmonic component of current
I_H	=	Total harmonic current
IT	=	Isolated terra
IHD	=	Individual harmonic distortion
ILE	=	Energy not supplied (ikke levert energi)
LSTM	=	Long short-term memory
MCC	=	Mathews correlation coefficient
MSE	=	Mean squared error
N_1	=	Number of coil turns on the primary side of a transformer
N_2	=	Number of coil turns on the secondary side of a transformer
n	=	Harmonic number
p(t)	=	Immediate power
P	=	Active power
PQ	=	Power quality
PMU	=	Phasor measurement unit
PQA	=	Power quality analyser
pu	=	Per unit

PWM	=	Pulse width modulation
Q	=	Reactive power
R	=	Ohmic resistance
RF	=	Random forest
RBF	=	Radial basis function
RNN	=	Recurrent neural network
ROC	=	Receiver operating characteristics
RMS	=	Root mean square
S	=	Apparent power
SVM	=	Support vector machine
THD	=	Total harmonic distortion
TN	=	Terra neutral
TN	=	True negative
TP	=	True positive
TPR	=	True positive rate
t-SNE	=	t-distributed stochastic neighbour embedding
U_1	=	Primary side voltage of a transformer
U_2	=	Secondary side voltage of a transformer
$v(t)$	=	Immediate voltage
V	=	Amplitude of voltage
V_a	=	Amplitude of current in phase a
V_{a0}	=	Zero sequence voltage in phase a
V_{a1}	=	Positive sequence voltage in phase a
V_{a2}	=	Negative sequence voltage in phase a
V_n	=	Harmonic component of voltage
$f(x;\theta)$	=	Function provided by a machine learning model
$f^*(x)$	=	Target function
$J(\theta)$	=	Loss function
k	=	Number of splits in cross validation
L	=	Per example loss
m	=	Number of samples
$x^{(i)}$	=	One sample of data
$x^{(i)}_{norm}$	=	One normalised sample of data
$x^{(i)}_{std}$	=	One standardised sample of data
X_{max}	=	Largest sample in a data set
X_{min}	=	Smallest sample in a data set
$y^{(i)}$	=	True target of the samples
θ	=	Phase angle between the voltage and the current
θ	=	Parameter adapted by the learning algorithm (not to be confused with the phase angle)
μ_X	=	Mean of the samples in a data set
σ_X	=	Standard deviation of the samples in a data set
ω	=	Angular velocity of a periodic waveform

Chapter 1

Introduction

The modern society makes people dependent on energy. Energy consumption has increased steadily trough time and exploded by the introduction of fossil energy carriers and again by the introduction of the steam machine. This has led to tremendous prosperity and increased standard of living. The flip side is now starting to emerge: Increased frequencies of extreme weather in a global perspective, and pollution on a local level [1]. Therefore, the global community is working together to reduce the climate change impact on the globe [2, 3]. The way energy is generated and consumed needs to change. The demand for energy does not seem to decrease in the future. It will rather increase. This increase must be covered by renewable energy sources. Renewable energy sources do also have to act as a replacement of fossil fuels, since these sources are being phased out [4]. The energy carrier delivered by most renewable energy sources is electrons. By present time only 20% of the global energy consumption is based on electricity. The remaining parts is partitioned as fuel in transportation or heat in industry and households. However, by 2050, this may increase to 45% due to electrification of the transportation sector and the industry by phasing in electrical vehicles and electrical arc furnaces. In certain areas the electrical proportion may be as high as 63% [5]. In general, by 2050, 80% of the energy produced is expected to come from renewable sources [6]. Focusing on the electrification of the power system, it is inevitable to not consider the state of the electrical power grid. It is an ageing grid designed to deliver 'linear' power and adapted to the development trends of the past 70-80 years [7, 8]. A high amount of adaptions and new solutions is necessary for the electrical power system to handle the implementation of renewable energy generation [6].

Several areas need to be developed and combined, stressing the focus on innovation [6, 9]. One such area is digitalisation. This area embraces wide and is essential regarding electrification and the development of the new power system [5, 6, 9, 10, 11, 12]. Automatic solutions could monitor the system and contribute in decision-making [13]. Several European countries, including Norway, are highly involved in research on these so called smart grids [7, 14]. Smart grid embraces areas such as cyber security, automatic load scheduling, and load- and fault prediction. New challenges will be introduced as a result of smart grids. Increased use of power electronics in the power grid will lead to degrada-

tion of the power quality, and increase the occurrence of harmonic components, if counter actions are not taken. Poor power quality may lead to wear and tear with destructive consequences on electrical components.

As mentioned, one of the objectives of the smart grid is to prevent faults from occurring in the power system. Preventing fault from happening in the power system increases the reliability of power supply and reduces the economical expenses related to damaged equipment and energy not supplied (ILE). This may be conducted by monitoring the power supply in real-time using sensors placed around the power grid. The measurements from the sensors may be fed into an algorithm trained on recognising patterns and signatures related to various faults. This has been enabled through the development within data processing- and communication technology. Some research have been conducted classifying fault events related to power system using deep learning [15, 16, 17, 18, 19]. However, little has been done on predicting or forecasting the fault events [20, 21]. With this in mind, some central questions. Is it possible to predict the faults before they happen? How long in advance can they be predicted, if the first question turns out positive? What kind of physical parameters are most suitable to use as features for prediction? What kind of method should be used to predict the faults? Will it be possible to implement this in a real-time monitoring system?

In this master's thesis the questions coming up will be explored combining knowledge from the electrical power system domain and the data science domain to obtain a qualitative evaluation of the tests conducted. Measurement data of the fault types interruption, voltage dip, and earth fault, as well as data gathered from nominal operated power grid will be used in the predictions. The samples are collected from Norwegian power grids operated at voltage levels ranging from 15 kV to 300 kV. The assessment will be divided into multiple tests. The objective in focus is to build and compare results from three different recurrent neural network (RNN) architectures trained on time-series data acquired from power quality (PQ) measurements.

Following chapters are divided into: background knowledge; outlining theory of which this thesis is based on, and method; outlining what has been done and how. The results and discussion present the test results and discuss their meaning, and proposing further work. Finally, a conclusion is made and recommendations of further work will be given.

Chapter 2

Background Knowledge

Various sources have been considered during this master's thesis. The most central sources are listed bellow, with some comments.
Research papers explored various deep learning architectures and were used as inspiration for this master's thesis:

- *Deep power: Deep learning architectures for power quality disturbances classification*, by Mohan *et al.* [16]
- *Data-Based Line Trip Fault Prediction in Power Systems Using LSTM Networks and SVM*, by Zhang *et al.* [19]
- *Railway Track Circuit Fault Diagnosis Using Recurrent Neural Networks*, by de-Bruin *et al.* [22]
- *Classification of Power Quality Disturbances via Deep Learning*, by Ma *et al.* [18]
- *Deep Learning Architecture for Voltage Stability Evaluation in Smart Grid based on Variational Autoencoders*, by Yang *et al.* [17]

The topics in the following papers were highly inspirational: The paper *Reducing the Dimensionality of Data with Neural Networks*, by Hinton *et al.* introduces the concept of Autoencoding [23] and the paper *Visualizing Data using t-SNE*, by van der Maaten *et al.* introduces t-SNE [24]. The paper *Unsupervised Learning of Invariances in Deep Networks*, by Park *et al.* presents a composite sequence-to-sequence model [25].

The books covering machine learning and deep learning are written by some of the most respected people in their field of research.

- *The Deep Learning Book*, by Goodfellow *et al.* [26]
- *Deep Learning With Python*, by Chollet [27]
- *Python Machine Learning*, by Raschka *et al.* [28]

- *Machine Learning Yearning: Technical Strategy for AI Engineers, In the Era of Deep Learning*, by Ng [29]
- *An Introduction to Statistical Learning*, by James *et al.* [30]

Books considered regarding the electrical power system:

- *Power Electronics - Converters, Applications, and Design*, by Mohan *et al.* [31]
- *Electrical Machines Drive and Power Systems*, by Wildi [32]
- *Modern Power System Analysis*, by Kothari *et al.* [33]
- *Power System Analysis & Design*, by Glover *et al.* [8]
- *Electrical Power Systems*, bym Wadhwa *et al.* [34]

Books considered regarding mainly power quality:

- *Power quality*, by Sanaran [35]
- *Electric Power Quality*, by Chattopadhyay *et al.* [36]

Following reports were considered regarding the future trends and development related to the power sector. They were chosen based on their impact and high credibility coming from respected sources.

- *Annual Report 2018*, by FME CINELDI [7]
- *1,5°C Hvordan Norge kan gjøre sin del av jobben*, by Energi Norge [11]
- *Digitization & Energy*, by IEA [12]
- *National and Regional Smart Grids initiatives in Europe*, by ETP [14]
- *Electrification with Renewables - Driving the transformation of energy services*, by IRENA [5]
- *Innovation landscape for a renewable-powered furure: Solutions to integrate variable renewables*, by IRENA [6]
- *Where does change start if the future is already decided?*, by EY [10]
- *Strategi 2018*, by Energi21 [9]

The following chapter contains the basic theory behind the main perspectives of the master's thesis. The first part will outline the concepts of the electrical power system from a Norwegian point of view. It will contain a brief overview of how the system is functioning today, before the system of tomorrow, such as smart grids, will be described. On a more technical note, the concept of power quality will be defined, and in more detail, some common fault events that may occur in the electrical power system. The second part will outline the concepts of machine learning, and more specific Deep Learning.

2.1 The Electrical Power System

The electrical power system consists of the following main parts:

- Power plants generating and converting mechanical energy to electrical energy. The main objective is to respond to the demand.
- Transformers, transforming the voltage level up or down depending on delivery or consumption, respectively. They are often found in different sizes in substations.
- Power lines that connect the electrical grid.
- Consumers (households, commercial or industry) which make up the load, and represents the demand.

The Norwegian electrical power grid is shown in **Fig. 2.1**. Whats so unique about the Norwegian power system is the excessive use of hydro-electric plants, that is contributing to an energy mix containing 98% renewable energy [37].

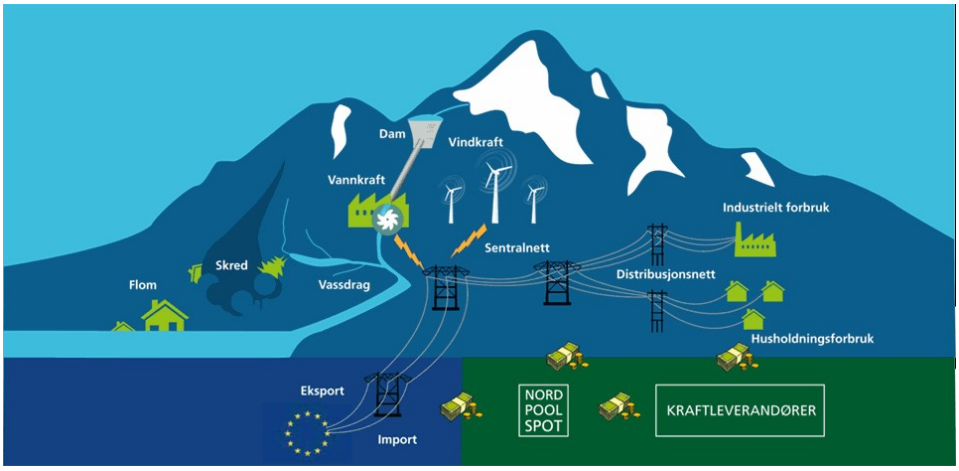


Figure 2.1: General layout of the Norwegian electrical power system [38].

2.1.1 The Power Grid

The Norwegian power grid can be divided into three main systems: Transmission system, regional system and distribution system. The transmission systems main objectives are to transmit electrical energy over long distances, such as between regions or nearby countries. This is usually a robust system as there are few interconnections and relatively easy to monitor. The voltage levels of the transmission system is somewhere between 132 kV - 420 kV. It is owned and maintained by the Norwegian transmission system operator (TSO), Statnett. The regional system is the next step after the transmission system. In this system the electricity is either transferred within the region, or delivered to industry or the distribution system. Before reaching the distribution system the voltage level is transformed down in a substation from its usual voltage levels of 33 kV - 132 kV. The distribution system consists of voltage levels between 230 V and 33kV, and is the system most people interact with. Due to its complex structure and kilometres of cables and lines it is more prone to failure. This is due to its complex structure and kilometres of cables and lines. Both the regional system and the distribution system are owned and maintained by local distribution system operators (DSO).

What differentiates the Norwegian power grid from the rest of Europe is the high usage of IT low voltage grids. This type of grid configuration is insulated from earth, meaning the neutral of the transformer is not grounded to earth. The most common type of circuit in Europe is the TN network, which has the neutral of the transformer grounded to earth. New low voltage projects in Norway will also use this form of grounding. That being said, this thesis will only focus on the high voltage grid.

The power generation in the traditional power grid is centralised. Meaning the flow of energy is unidirectional, going from the power plant through the transmission grid and to the consumer. However, in the future more power production will be distributed throughout the grid as a result of increased availability of energy production from renewable sources. This may lead to energy flow in the opposite of the originally intended and additional stress on an ageing power grid.

2.1.2 Smart Grid

Smart grid is the designation of the electrical power system of the future, with the purpose of delivering reliable energy and resiliency against disturbances [7]. There are several definitions of a smart grid. Glover *et al.* [8] defines it as:

> *"... uses technology to improve reliability, security and efficiency (both economic and energy) of the electric system from large generation, through the delivery system to electricity consumers and a growing number of distributed generation and storage resources."*.

This vision implies great changes to the current power system and huge investments into new equipment [12].

2.1.3 Digitalisation & Data

The IEA report on digitalisation and energy [12] describes digitalisation as:

> *"... the increasing application of digital technologies (i.e. ICT) across the economy, including energy, to achieve desired outcomes such as improved safety, efficiency and productivity. The trend toward greater digitalization is enabled by advances in data, analytics and connectivity: increasing volumes of data thanks to the declining costs of sensors and data storage, rapid progress in advanced analytics and computing capabilities, and greater connectivity with faster and cheaper data transmission."*

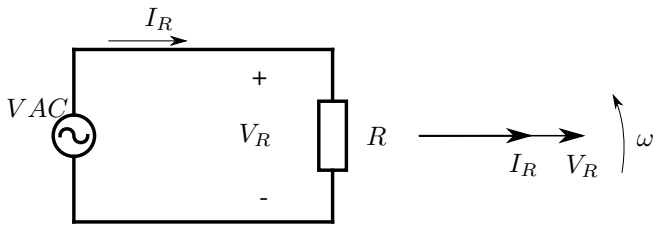
The new power system will rely on sensors and other components to gather and process data for use in operations [7]. There is a large amount of opportunities when it comes to digitalisation.

2.2 Fundamentals

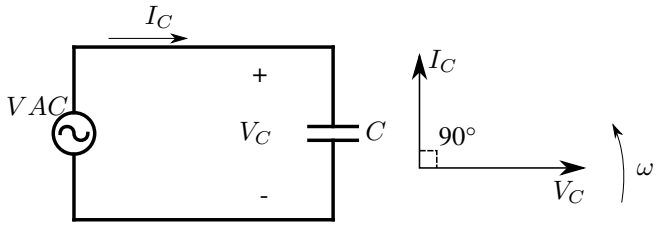
2.2.1 Electric Power

The core basics of electronic transfer of energy is an electrical potential difference between two points where electrons are free to move along a path. The path could be a circuit, and the difference in potential could be due to a voltage drop introduced by a load connected between the negative and positive poles of the applied voltage source. In direct current (DC) circuits only active power is drawn. Introducing an inductor or a capacitor into the circuit would either do nothing or cut the current flow if connected in series, respectively. In alternating current (AC) circuits the characteristics of the inductive and capacitive components, known as the reactive load, need to be taken into consideration. The resistive and reactive loads compose the inductance of the system, which is a complex quantity. In a circuit with purely resistive load, the voltage and current are in phase with each other, as seen in **Fig. 2.2a**. An inductive load on the other hand will draw lagging current, so that in a phasor diagram the current phasor would be 90° after the phasor of the voltage, as seen in **Fig. 2.2c**. This phenomenon is due to Lenz's law, which states that a current induced in a conductor by a changing magnetic field will generate a magnetic field opposite to the one creating it [39].

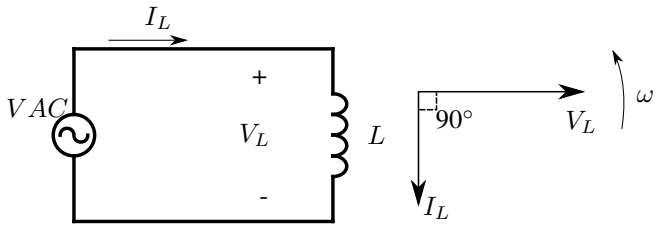
Simple AC Circuits



(a) Simple resistive AC circuit with AC voltage source VAC, current I_R, voltage V_R, and resistance R. The phasor diagram to the right show the RMS current being in phase with the RMS voltage.



(b) Simple capacitive AC circuit with AC voltage source VAC, current I_C, voltage V_C, and capacitance C. The phasor diagram to the right show the RMS current leading the RMS voltage by 90°.



(c) Simple inductive AC circuit with AC voltage source VAC, current I_L, voltage V_L, and inductance L. The phasor diagram to the right show the RMS current lagging the RMS voltage by 90°.

Figure 2.2: Simple circuits and phasor diagrams related to pure resistance, capacitance, and inductance, showing the relationship between voltage and current. ω indicates the direction of rotation.

The capacitive load will also react differently compared to a resistive load. In a AC circuit it will draw leading current, and will in the same phasor diagram mentioned earlier, lead the voltage by 90°, as seen in **Fig. 2.2b**. This is due to the fact that the capacitor is a charging component and the voltage across the capacitor is dependent on the charge level of the component. It can be further visualised as follows; when the capacitor is fully discharged, the electrons "sees" the capacitor as a short circuit, and a full voltage drop appear across the component. As electrons start to condense on one of the conductors inside the capacitor, a potential starts to build until maximum possible voltage from the voltage source is applied. At this point, the current flow will then be zero with no voltage drop across the component.

In the most basic forms, voltage and current can be expressed as sinusoidal functions of time, t.

$$v(t) = Vsin(\omega t) \tag{2.1}$$
$$i(t) = Isin(\omega t \pm \theta) \tag{2.2}$$

where $\omega = 2$ is the angular velocity of the periodic waveform, θ is the phase angle between the voltage and the current, and V and I are amplitude of the voltage and the current, respectively. See **Fig. 2.5** reference of the sinusoidal shape. In a purely resistive circuit, the product of the immediate voltage, $v(t)$, and the immediate current, $i(t)$ yield the immediate power, $p(t)$. The power transferred directly from a source is known as the apparent power, S, and is a complex quantity. A DC circuit or an AC circuit with purely resistive loads draws active power. Active power, P, is the real component of the apparent power that performs useful work, as well as contributing to copper loss, or ohmic loss. Copper loss is defined by I^2R, where R is the resistance and I is the current through the resistance.

An imaginary component is introduced to the apparent power in AC circuits with reactive loads, namely reactive power. The relationship is shown in **Eq.** (2.3) and visually in **Fig. 2.3**. Reactive power, Q, is the portion of the apparent power interacting with the inductive and capacitive units in, e.g. an electrical power system, where the positive or negative sign of the quantity depends on if the unit delivers or draws reactive power from the system, respectively. The inductive unit could be a transformer or an induction motor, where the reactive power is stored in the magnetic field. The capacitive unit might be power cables or other appliances in the power system drawing leading currents, such as capacitor banks. Especially capacitor banks are known to deliver reactive power to the system.

$$S = \sqrt{P^2 + Q^2} \tag{2.3}$$

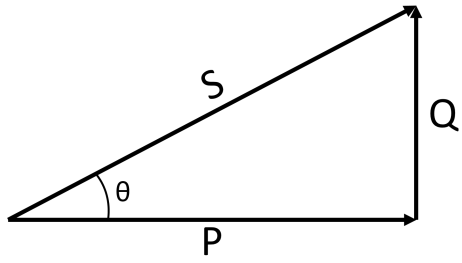


Figure 2.3: Power triangle. The trigonometric relationship between Apparent (S), active (P), and reactive (Q) power represented by phasors.

Reactive power is in most situation unwanted as it contributes to flow of unusable current that may degrade the transmission line. The power factor is a useful metric to examine the portion of the apparent power contributing to useful work. Its definition is seen in **Eq.** (2.4).

$$cos\,\theta = \frac{P}{S} \tag{2.4}$$

where θ is the phase angle between the voltage and the current.

The electrical power system usually consists of three phases. There are several reasons for this, firstly the root mean squared (RMS) value of active power transfer becomes constant, meaning a three phase power supply may deliver continuous power [32]. Secondly, the three phases contain information about rotation, which may be applied to an electric motor. Thirdly, additional phases increases the capacity of transfer of power [32]. **Fig. 2.4** illustrates the phase relationship between phase a, b, and c of a balanced system in a phasor diagram. Note that the phasors are 120° relative to each other and that the components are equal in magnitude.

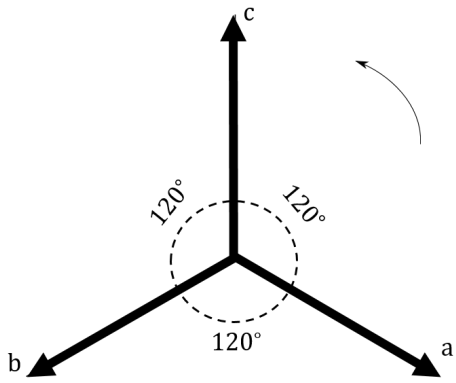


Figure 2.4: Phasor diagram of a balanced three phase power system. Phases a, b, and c are shifted 120° relative to each other and their physical parameters are equal in magnitude.

2.2.2 Transformer

In the electrical power system transformers play an important part. The task is to either reduce or increase the operation voltage, depending on the power delivered for consumption or transmitted over large areas, respectively. The functionality of the transformer is enabled by induction; two separate coils are wound around a ferromagnetic core. The ratio of which the change in voltages is decided by the wound turns of the coils, and is given by **Eq.** (2.5)

$$\frac{U_1}{U_2} = \frac{N_2}{N_1} \tag{2.5}$$

U_1 and U_2 are the respective voltage on the primary and secondary side of the transformer. N_1 and N_2 are the number of coil turns of the primary and secondary side. If the voltage applied to the transformer reaches a certain amount past the nominal voltage, the core of the transformer reaches saturation. This happens when the applied magnetic field is not able to increase the magnetic flux density in the core any more, due to the property of the core material.

2.2.3 Power Quality

Power Quality (PQ) is a term used to describe the condition of the energy transferred in an electric power system. The objective is to maintain a near sinusoidal waveform, as seen in **Fig. 2.5**, of the rated voltage and current [36]. As more delicate electrical devices are manufactured, the delivery of power has become more strict since sudden variations may destroy the equipment [36]. **Fig. 2.6** displays an example of the development of an interruption in a three phase power system.

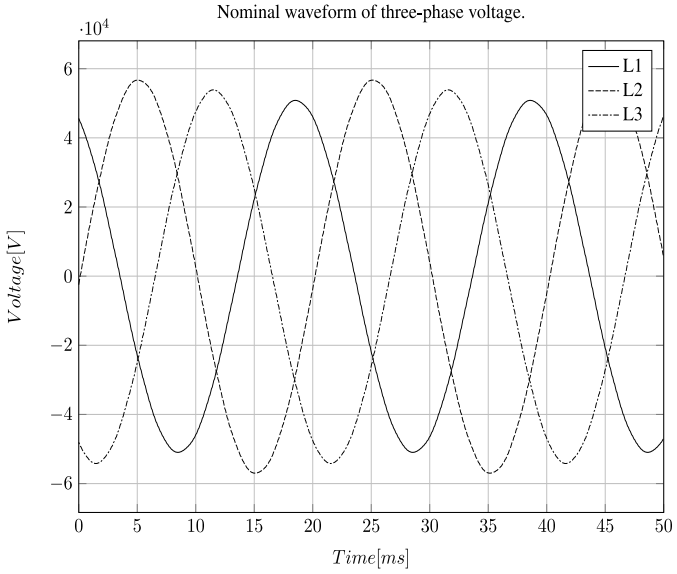
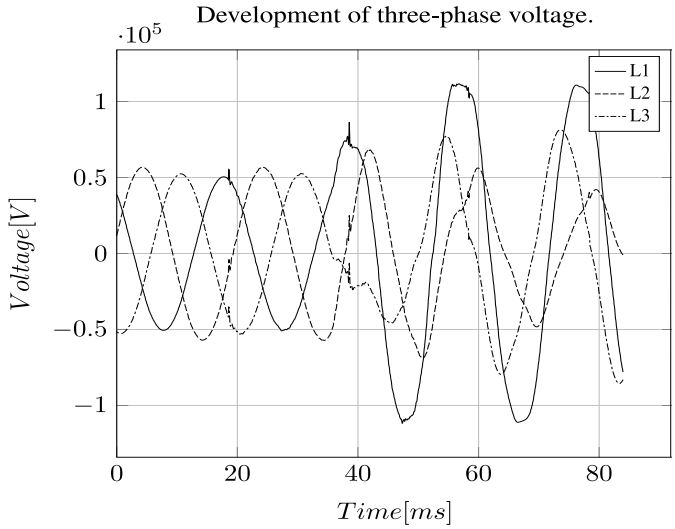
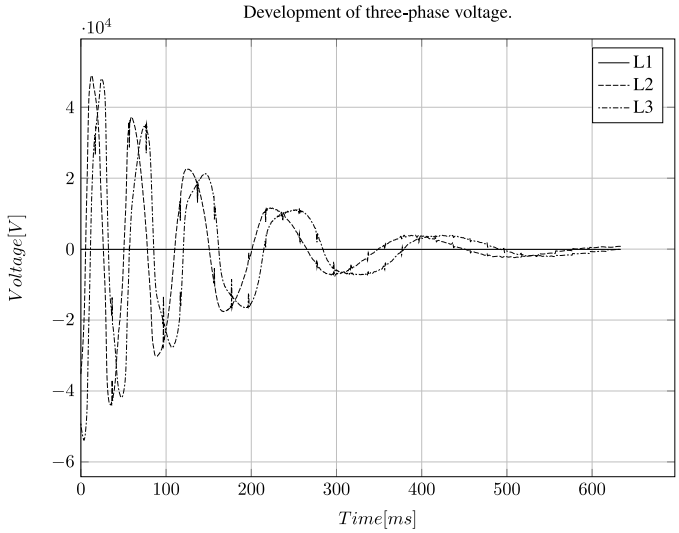


Figure 2.5: Plot of the nominal three-phase voltage in a power system. Data points are gathered from Elspec Investigator.



(a)



(b)

Figure 2.6: Plot of the development of an interruption in a three-phase power system. Data points are gathered from Elspec Investigator. a): First visual sign of fault. b): Voltage reduces to zero.

Phasor Measurement Unit (PMU) is also a way to monitor the condition of the power system, but will not be taken into consideration in this thesis because they only measures every 20 milliseconds while power quality analysers (PQA) are able to measure at least 1024 samples per cycle [20]. Typical terms within PQ will be further outlined in this section.

Power Quality Measurements

To get reliable measurements, it is important to choose an instrument that is suited for that application, as stressed by Sanaran [35]. One leading company in this area is Elspec, who provides PQ monitoring equipment for measurements and analysis [40].

Definition of disturbances and faults:

Interruption

In general an interruption is a reduction in power supply. It is defined by Norwegian legislation that the supply voltage to the customer is under 0.05 per unit (pu), or phrased differently as under 5% of the agreed supply voltage, where long term and short term interruptions have duration over and under 3 minutes, respectively [41].Other definitions of interruption is a reduction in power supply to less than 0.1 pu [36]. Causes may be faulty equipment, protection gear activated, or operation gone wrong. Interruptions may lead to ILE, which stands for 'not delivered energy' and can result in large economic expenses.

Over voltages

Over voltages are by Norwegian legislation defined as rapid increase in RMS voltage to over 1.1 pu, lasting from 10 milliseconds to 1 minute [41]. Causes may be due to faulty isolation, ferro resonace, or induced voltages due to lightning [36]. This can lead to overload on the isolation, reduced voltage stability, and even demand for more reactive power [36].

Under voltages

Under voltages are by Norwegian legislation defined as rapid reduction in RMS voltage to under 0.9 pu but over 0.05 pu, lasting from 10 milliseconds to 1 minute [41]. Causes may be incapability to deliver enough power to the loads, due to high demands or low delivery, high demand for reactive effect, or other faults in the power system.

Voltage dip (Sag)

In literature voltage dip is defined as variation in RMS voltage between 0.1 and 0.9 pu, with a duration between 10 milliseconds and 10 minutes [36]. Causes may be sudden consumption of power, startup of a large induction motor, or a line-to-earth fault.

Swell

Swell is by literature defined as variation in RMS voltage between 1.1 and 1.8 pu, with a duration between 10 milliseconds and 1 minute [36]. Causes may be shutdown of large loads, charging of capacitor banks, increased voltage in healthy phases during a line-to-earth fault in an isolated grid.

2.2.4 Grounding

Grounding is defined as a conductive connection established between a body and earth or a large conductive element functioning as earth [35]. In an electrical power system the neutral of the transformers are usually grounded. This is conducted due to the fact that the performance of the state of the neutral will have influence on the performance of the power system under various conditions [34]. There are several advantages of grounding the neutral, *e.g.* the phase voltages are constrained to the phase-earth voltage, and over voltages due to lightening discharges to earth. There are different practices of grounding an electrical system. The four typical grounding practices are isolated neutral, solid grounding, resistance grounding and reactance grounding. Isolated and resistance earthed grids operates at nominal voltages under 24 kV. Reactance grids operates between 20 - 150 kV. Solid grounded grids have a nominal operating voltage of over 100 kV [42].

Isolated neutral is where the neutral of the transformer in the power system is not connected to earth. The advantages of this approach of grounding is the possibility to maintain power delivery even with fault on one of the lines, and a reduction of interference on communication lines due to lack of zero sequence currents. Solid grounding is the most common grounding method. The neutral is connected directly to earth. Resistance grounding is basically adding a resistance between the neutral and the ground. This is done to reduce short circuit currents due to earth fault to ensure that the currents becomes large enough such that the circuit breaker trips. Reactance grounding is used to counteract the short circuit currents by adding a coil, known as a Peterson coil, between the neutral of the transformer and the ground. Some parts of the old net are also upgraded and fitted with solid grounding [43].

Short circuit fault

The conduction lines in the power system may be exposed to faults, such as short circuit of lines in between or to ground. The typical faults are line-to-ground, line-to-line, double line-to-ground, and three phase short circuit. The first three are unbalanced faults, which introduces negative and zero sequence components [35].

2.2.5 Symmetrical components

When analysing a three phase power system, each phase (a, b and c) are often represented by a phasor with magnitude and direction. These phase phasors will in a balanced system be equal in magnitude, and be displaced by 120°, as shown in **Fig. 2.7a**. It is called the positive sequence. However, if the three phase system becomes unbalanced, additional sequences may occur. These sequences are called negative and zero sequence.

Fig. 2.7b and **Fig. 2.7c** illustrate the phasor diagram of the negative and the zero sequences, respectively. They will be the symmetrical components together with the positive sequence constructing the original phase signal as seen in **Eq.** (2.6) and **Eq.** (2.7). The same applies to phase a and b. **Fig. 2.7d** illustrates the relationship of the unbalanced phases. The sequences do not interact directly with each other since they are uncoupled [34]. During faults introducing zero sequence components, zero sequence currents tend to add up in the neutral of the system.

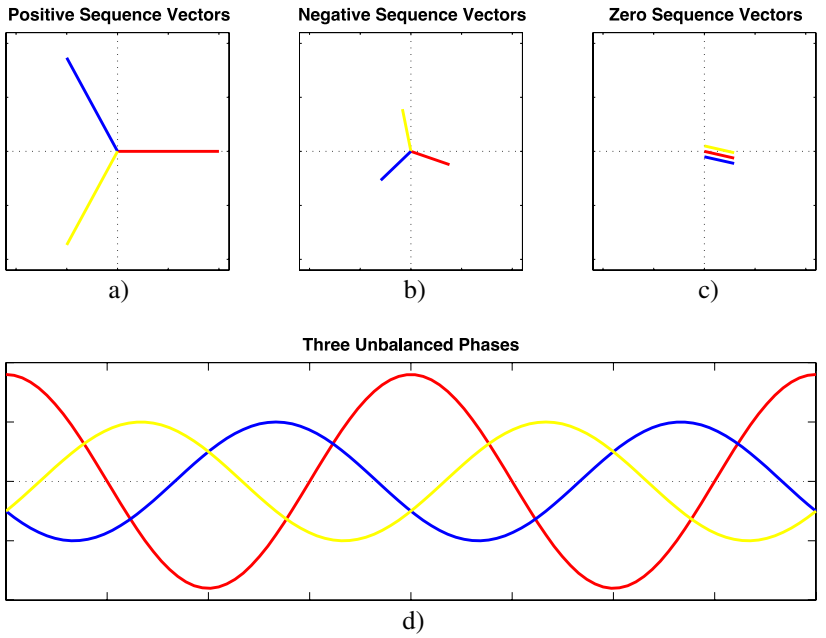


Figure 2.7: Diagram showing the case in which there are three unbalanced phases, and the necessary symmetrical components that will create the resulting three-phase system. Red is phase a, yellow is phase b, and blue is phase c. Illustration by Kashyap.valiveti [44].

$$I_a = I_{a0} + I_{a1} + I_{a2} \tag{2.6}$$
$$V_a = V_{a0} + V_{a1} + V_{a2} \tag{2.7}$$

Positive sequence components will have rotation and sequence equal to the phase signal in balanced condition, following a counterclockwise direction. The negative sequence components will have a counterclockwise rotation with a clockwise sequence. Whereas the zero sequence components are equal in magnitude, with no phase displacement [36].

2.2.6 Harmonics

For most power systems, the waveform of the voltage or the current is in some degree distorted. The waveform does not appear purely sinusoidal. The true waveform is a combination of multiple waveforms superimposed on each other. As the fundamental being the rated frequency of the system, and the other being decomposed harmonic components of the distorted waveform. These harmonics are in general unwanted, due to their interference with different applications in the power system. The harmonic power cannot be utilised as work, only dissipated as heat in the AC circuit [32].

To visualise the distorting effect of a harmonic, as inspired by Wildi [32], consider two pistons on top of each other going up and down in a smooth sinusoidal manner, one double as fast as the other. Focusing on the end of the top piston as a function of time, its motion will result in a flat topped wave, or distorted wave. **Fig. 2.8** illustrates the effect of harmonic components on a sinusoidal wave, by shows the first four partial sums of the Fourier series for a square wave.

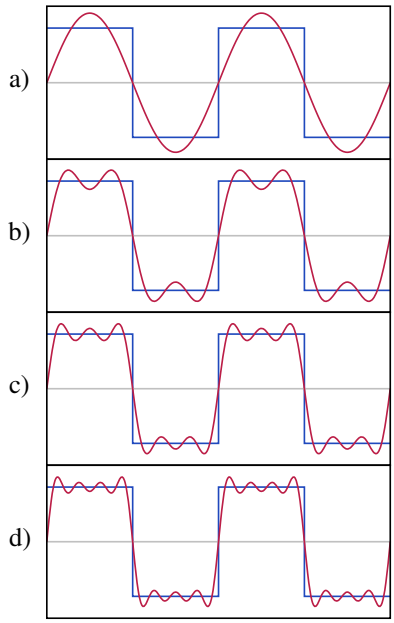


Figure 2.8: The first four partial sums of the Fourier series for a square wave. a) show the square wave in blue and its fundamental in red. To create a square wave from periodic sinusoidal waves odd harmonics of the fundamental need to be added together with the fundamental. b) shows the fundamental combined with its third harmonic. c) shows the fundamental in combination with both the third and the fifth harmonic. d) shows a nice approximation of the square wave as a sum of the fundamental, third, fifth, and the seventh harmonics. In general, the presence of harmonic components in a waveform is an indication of a distorted non sinusoidal waveform.

As mentioned, periodic non sinusoidal waveforms can be decomposed into a fundamental component and harmonic components. This can be achieved by utilising the

Fourier transform [35], which yields

$$\begin{aligned} v(t) = V_0 + V_1 sin(\omega t) + V_2 sin(2\omega t) + V_3 sin(3\omega t) \\ + \ldots + V_n sin(n\omega t) + V_{n+1} sin((n+1)\omega t) + \ldots \end{aligned} \tag{2.8}$$

As seen from **Eq.** (2.8) the Fourier expression is an infinite series, where V_0 represents the constant DC component of the waveform and $V_1, V_2, V_3, \ldots$ corresponds to the harmonic components representing the amplitude of the terms.

Analytically, a periodic waveform of fundamental frequency $\omega = 2\pi f$ can be expressed as [31]

$$f(t) = F_0 + \sum_{h=1}^{\infty} f_h(t) = \frac{1}{2}a_0 + \sum_{h=1}^{\infty} \{a_h cos(h\omega t) + b_h sin(h\omega t)\}$$

where $F_0 = \frac{1}{2}a_0$ is the average value.

$$a_h = \frac{1}{\pi} \int_0^{2\pi} f(t) cos(h\omega t)\, d(\omega t) \quad h = 0, \cdots, \infty$$

$$b_h = \frac{1}{\pi} \int_0^{2\pi} f(t) sin(h\omega t)\, d(\omega t) \quad h = 0, \cdots, \infty$$

$$F_0 = \frac{1}{2}a_0 = \frac{1}{2\pi} \int_0^{2\pi} f(t)\, d(\omega t) = \frac{1}{T} \int_0^{T} f(t)\, dt$$

a_h is the harmonic component coefficient corresponding to even symmetric quantity of the original signal. b_h is the harmonics component coefficient corresponding to odd symmetric quantity of the original signal.

The harmonic components can, as indicated, be classified as odd, even, inter, and sub harmonics. Where odd and even harmonic frequencies are odd and even multiples of the fundamental frequency, respectively. These are the main parts in combination with the fundamental constructing the non sinusoidal waveform. The inter harmonics can also be present in non sinusoidal waveforms. However, they consist of harmonic frequencies higher than the fundamental and are not integer multiples. The sub harmonic on the other hand are made up of harmonic frequencies below the fundamental frequency.

Considering a three phase system, each phase will have its harmonic components with relationships similar to the fundamental voltage and current waveform phasors. This denotes that in a balanced system, the harmonic voltages of equal harmonic number would be 120° apart with equal magnitude. Due to symmetry of the fundamental waveform, even harmonics will in most cases be absent due to their asymmetrical shape. Harmonics with multiples of three will triple due to constructive interference. The odd harmonics will not be affected by the balanced three phase, because they will follow the symmetry of the fundamental. The angle between the fundamental voltage and the fundamental current would be the displacement power factor angle. The fundamental and the harmonics are uncoupled. Reactive and apparent power are not defined for harmonic voltages and currents [32].

The concept of symmetrical components may be introduced to the analysis of harmonics. In general harmonic components in a three phase system can be categorised as either positive , negative , or zero sequence. Where the 1st, the fundamental, is positive sequence, 2nd is negative sequence, 3rd is zero sequence, 4th is positive sequence, 5th is negative sequence, 6th is zero sequence, and so on. Appendix C outlines the proof of central harmonic sequences. **Table 2.1** lists some harmonic numbers and their corresponding sequence.

Table 2.1: Harmonic numbers and their corresponding sequence.

Sequence	**Harmonic number**
Positive	1, 4, 7, 10, 13, 16, 19
Negative	2, 5, 8, 11, 14, 17, 20
Zero	3, 6, 9, 12, 15, 18, 21

It is observed that the triplen harmonics are zero sequence components, indicating that the displacement angle between the phasors is zero. This may result in triplen harmonic currents adding up in the neutral conductor in a transformer or a bus bar [35]. The positive sequence harmonics will follow the rotation of the fundamental and have the same sequence as the fundamental currents and voltages. This will cause torque in the same direction as the fundamental component on the rotor when applied on the stator of a rotating machine, with a frequency corresponding to the harmonic frequency [36]. On the other hand, the negative sequence harmonics will apply a torque in the opposite direction of the shaft rotation, counter acting the fundamental magnetic field. In a balanced three phase system, both the positive sequence currents and the negative sequence currents will cancel in the neutral, while the zero sequence currents will add [32].

Calculating the individual harmonic distortion (IHD), and the total harmonic distortion (THD) is a common approach to analysing the harmonic content of a waveform [31, 35]. IHD is the rate between the RMS value of individual harmonics and the RMS value of the fundamental [41] as in **Eq.** (2.9), considering current. The same is applicable to harmonic components of voltage. The fundamental is sometimes referred to as the first harmonic.

$$IHD_{I_n} = I_n/I_1 \tag{2.9}$$

where I_1 is the fundamental component, and I_n is the n^{th} harmonic component after the fundamental.

THD is used to describe how much a non sinusoidal wave deviates from a perfect sinusoidal wave [35]. It is the ratio between the RMS value of all the harmonics and the RMS value of the fundamental [41], and is given by **Eq.** (2.10)

$$THD_I\% = (I_H/I_1) \times 100\% \tag{2.10}$$

, where $I_H = \sqrt{I_2^2 + I_3^2 + I_4^2 + I_5^2 + \ldots}$.

It is also possible to express the THD using the IHDs as in **Eq.** (2.11).

$$THD = \sqrt{IHD_2^2 + IHD_3^2 + IHD_4^2 + IHD_5^2 + \dots} \tag{2.11}$$

As nicely stated by Snaran [35]:

> *"The individual harmonic distortion indicates the contribution of each harmonic frequency to the distorted waveform, and the total harmonic distortion describes the net deviation due to all the harmonics. The total harmonic distortion, while conveying no information on the harmonic makeup, is used to describe the degree of pollution of the power system as far as harmonics are concerned."*

There are several sources of harmonic distortion. Examples are nonlinear loads, magnetic saturation in the core of a transformer, or power electronics [32, 35]. The nonlinear loads may absorb reactive power, where the majority produces odd harmonics [35]. If the loads draw uneven current between the positive and negative halves of a cycle even harmonics may occur. Nonlinear load could be adjustable speed drives (ASD) applying pulse width modulation (PWM), fluorescent lights, rectifier banks, and arc furnaces. There are several ways a transformer can produce harmonics. During excitation, the characteristics of the magnetising material are nonlinear. This is the main source of zero sequence triple harmonics [36]. Over excitation is another source of harmonic components. Voltage values exceeding the rated value are applied to the transformer, resulting in saturation of the core, generating odd harmonics [36]. The main source of harmonic components in the power system is semiconductor based devises, such as found in power electronics. This source can generate all the different types of harmonics.

The presence of harmonic components can affect the power system in various ways. In a transformer, harmonic voltages may cause additional losses due to hysteresis and eddy current, as well as copper loss. Copper loss is given by I^2R. This increases stress on the insulation and excessive heating. The same applies to electrical AC motors. Capacitor banks used for power factor correction may act as a sink for harmonic currents, and cause overload and collapse of the bank. In severe cases resonance between the capacitor bank and the rest of the power system may occur, leading to over voltages and high currents resulting in increase loss and overheating of the capacitors [35, 36].

In transmission lines, harmonic currents will cause additional copper loss in the conductor, reducing the capacity of transferring power [36]. Skin effect is a known phenomena in AC systems, where concentration of current tend to be high near the surface of the conductor. This, in addition to the proximity effect, create higher effective AC resistance, since they depend on frequency. With the increase in frequency the possibility of audible noise increases [33].

2.3 Machine Learning

Machine learning is a field embracing several disciplines, such as informatics, data science, mathematics, and statistics. Machine learning is used in classification problems and regression. In machine learning, a model of algorithms are said to learn from data [26]. Mitchell [45] stated the definition of a learning algorithm to be

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*

It has in recent years exploded in popularity due to the increased amount of data gathered from sensors and other sources, and the advancement in computer technology and processing capability. The availability of open source machine learning frameworks, such as TensorFlow, Keras, scikit-Learn, and Torch, to name a few, have brought machine learning to the layman.

Baseline

Developing machine learning algorithms tend to result in complex models. The performance of the model is compared to a baseline, to evaluate the benefits of adding more complexity to the model. Examples of baselines are a naive classifier, such as coin toss, a very simple artificial neural network (ANN), or basic statistics.

State-of-the-art models

Two benchmark models have been established in machine learning; Support Vector Machines (SVM) and Random Forest (RF).

The SVM tries to separate data points by fitting a hyperplane with the help of support vectors. These support vectors are used to maximise the margin, or distance between the support vectors and the hyperplane. The popularity of the SVM may be due to its applicability of kernels. This kernel trick transforms the data points into higher dimensional space, where a non-linear problem can become a linear problem. After separation, the data points is then transformed back to the original dimension [26]. The most common used kernel is the Radial Basis Function (RBF) kernel, also known as Gaussian kernel. It is not by default included in the Keras API, however Darecophoenixx [46] have implemented a python class compatible with the Keras API layers.

RF uses tree based decisions to navigate to the right class, and have a high performance on complex problems. The high performance is mainly a result of its structure. The structure is a variant of ensemble learning, where the prediction is made by a majority vote among multiple decision trees. A decision tree chooses its path by answering a series conditions until it reaches its final decision.

2.3.1 Pre processing

One of the simplest and most effective pre processing techniques for optimisation problems is to scale the data putting the features on the same scale [28]. This can be achieved

by min-max normalisation or standardisation, and are represented by **Eq.** (2.12) and **Eq.** (2.13), respectively.

$$x_{norm}^{(i)} = \frac{x^{(i)} - X_{min}}{X_{max} - X_{min}} \tag{2.12}$$

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_X}{\sigma_X} \tag{2.13}$$

$x^{(i)}$ is one sample, X_{min} is the smallest sample, and X_{max} is the largest sample, μ_X is the mean of the sample to be standardised and σ_X is its standard deviation.

2.3.2 Classification

There are some challenges that may occur when dealing with samples containing more than one class. For instance, having a data set of samples containing more samples from one class results in an imbalanced data set. This is seen as problematic as the majority class may dictate the decisions of the model, such as picking only the majority class when classifying a problem. There are several strategies for keeping the model from only choosing the majority class . One way is to under sample the majority class, such that the number of samples of the majority class equals the number of minority classes, resulting in a balanced distribution of all classes. Another technique is to over sample the minority class, which results in duplicating existing samples to yield a balanced data set [28].

Binary

In general binary classification the classifier decides if a sample belongs to one given class (positive) or not (negative).

Multiclass

It is a multiclass classification problem if data set contains multiple classes. A multiclass classification problem is conducted by a One-vs-All approach where the problem is split into multiple binary classification problems, seeing one class as the positive and the rest as negative.

2.3.3 k-Fold Cross Validation

k-fold cross validation is a technique to get a robust measure on the performance of the model. It splits the training set into k splits, and the model trains on k-1 splits and validates on the remaining. This will be performed k times and the average of the evaluation measurements gives a more representative description of the performance of the model.

2.4 Deep Learning

Deep learning is a branch within machine learning. It has increased in popularity rapidly this decade mainly due to advancements in computer technology. Powerful graphical processing units (GPU) have led to breakthroughs in image classification, here referring to AlexNet's classification accuracy on images from ImageNet, and further improvements in speech recognition [26]. The main idea of deep learning is to train a network on a significant large amount of data, and let the hidden layers in the network extract and learn the patterns and variations within the data. This is achieved by having a tuple of two elements; input data going into the network, and the expected output of the network, known as the target.

The following sections will outline the deep learning architectures that are intended to be studied in this thesis.

2.4.1 Long Short-Term Memory (LSTM)

Regular ANNs are not able to handle the temporal dependencies in a time serie. The reason for this is that they cannot operate sequences. In sequences the first element directly influences the next and so on in one direction. This challenge has been partly solved in an architecture referred to as recurrent neural networks (RNN). In these networks each time step output a hidden state sent to the next time step combined with the input at that time step.

However, the basic RNN model has a weakness when it comes to dependencies over long sequences, due to vanishing gradients when backpropagating. The introduction of the Long Short-Term Memory (LSTM) architecture eliminated the vanishing gradients problem [47]. In the LSTM, the memory state is propagated through the cell without activation, using a gated unit approach. This means that during differentiation the memory state would not be affected. The output of the LSTM gates are given by **Eqs.** (2.14)-(2.18), where i_t is the output of the input gate, f_t is the output of the forget gate, c_t is the cell state of the current unit, c_{t-1} is the cell state of the previous unit, o_t is the output of the output unit, and h_t is the hidden state, or activity output, which will be passed to the next unit or layer. The LSTM has several possible configurations, where each cell in the sequence could output directly or be shifted. The number of outputs could also be defined. See Raschka and Mirjalili [28] for more details. **Fig. 2.9** shows the structure and the active paths of a LSTM unit.

$$i_t = \sigma(W_{xi}X_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.14}$$
$$f_t = \sigma(W_{xf}X_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.15}$$
$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}X_t + W_{hc}h_{t-1} + W_{hc}h_{t-1} + b_c) \tag{2.16}$$
$$o_t = \sigma(W_{xo}X_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.17}$$
$$h_t = o_t tanh(c_t) \tag{2.18}$$

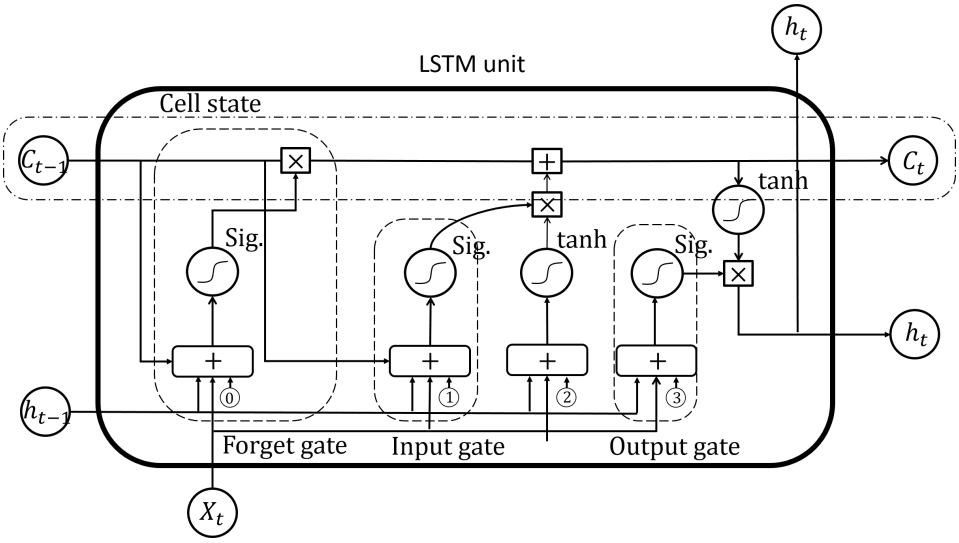


Figure 2.9: Figure of a LSTM unit, where X_t is the input vector, C_{t-1} is the memory from previous block, h_{t-1} is the output of the previous block, C_t is the memory from current block, h_t is the output from current block. Circles with S shape is denoted activation functions. Summation and multiplication blocks are element-vice. Numbers are respective biases. The figure is based on [48].

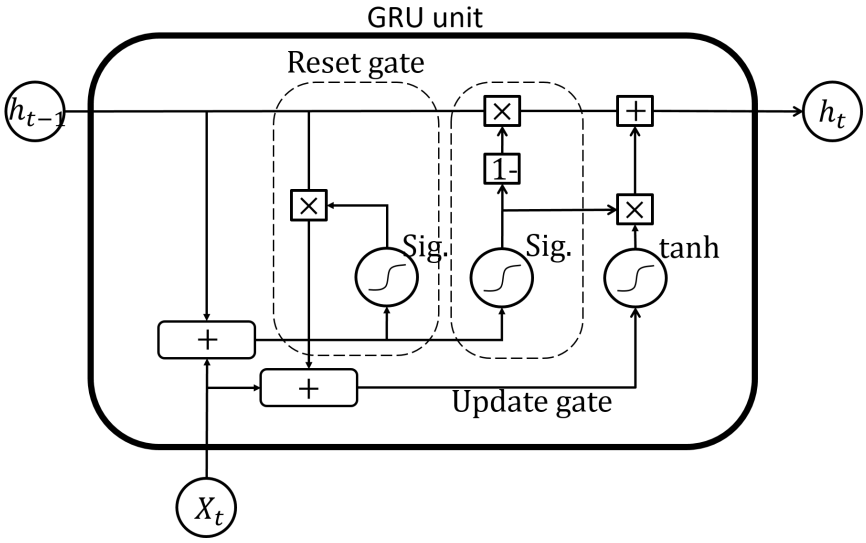


Figure 2.10: Figure of a GRU unit, where X_t is the input vector, h_{t-1} is the hidden state output of the previous block, h_t is the hidden state output from current block. Circles with S shape is denoted activation functions. Summation and multiplication blocks are element-vice. The figure are based on [48].

2.4.2 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit (GRU) architecture is a RNN variation introduced by Cho *et al.* [49] in 2014 and further explored by Chung *et al.* [50]. It is said to perform better compared to the LSTM architecture on problems with relatively small data sets [27]. The GRU is similar to the LSTM, but has half the amount of parameters. It uses the hidden state rather than an isolated memory cell to transfer memory across units. This leaves the GRU architecture to allocate less memory compared to the LSTM architecture. **Fig. 2.10** shows the structure and the active paths of a GRU unit.

2.4.3 Autoencoder

The concept of the Autoencoder was first introduced by Hinton *et al.* [23]. The main idea is to reduce the representation of a set of high dimensional data points into low dimensional codes. The code can then be used to reconstruct the data points back to its original state. The Autoencoder is an ANN, that can be thought of containing two parts; one encoder and one decoder. **Fig. 2.11** illustrates the layers of an Autoencoder. The task of the encoder is to compress the input data, such that the dimension of the input data is reduced at the output. This output will be the representation of the input data in lower dimensions, and is the previously mentioned code. The decoder on the other hand will use the code to reconstruct the original input data. This will be achieved through training, where the output of the decoder is compared to the actual input data. The whole network will be updated through backpropagation in a process called self supervised training. After the network has been trained, the encoder may be isolated and used as a data compressor or feature extractor.

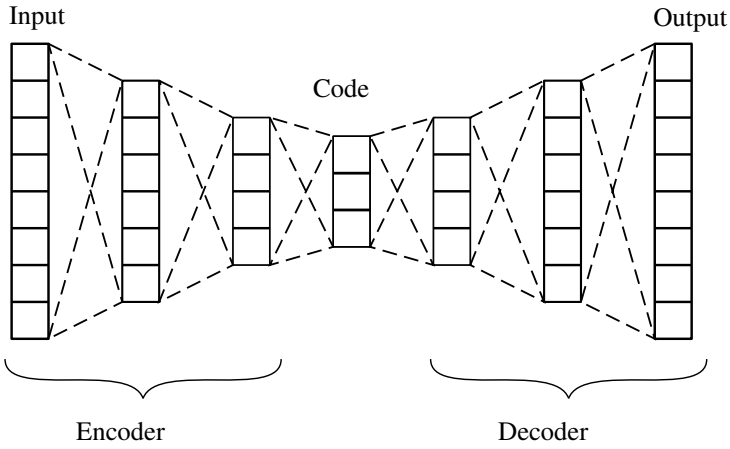


Figure 2.11: Illustration of an Autoencoder.

Sequence-to-Sequence Autoencoder

The sequence-to-sequence (s2s) Autoencoder builds on the same principles as the regular Autoencoder [51]. The encoder and decoder parts will in this case be two RNN type architectures. However, the representation will be the hidden states of the last sequence of the encoder. These hidden states are used to initialise the hidden states of the decoder RNN. Examples of use cases for the decoder are reconstruction of the input sequence of the encoder, and language translation in sentiment analysis and natural language processing (NLP) [49][52]. The decoder part does not need to be provided with any input data, only the hidden states are necessary. The output is as mentioned before. When the model is trained, the encoder may be isolated, and the hidden states representing the code of the Autoencoder, can be used in analysis or as feature elements fed into a classifier model.

2.5 Algorithm Evaluation Methods

2.5.1 Metric

A metric is a way of measuring the performance of a model to evaluate the suitability of the model for a specific task, being, regression, prediction or classification.

Confusion matrix

Both in binary and multiclass classifications confusion matrices serve as an appropriate and easy tool to analyse the performance of the model. In binary classification, as mentioned in chapter 2.3.2, the outcome could either be true or false. This means that if a classification of a sample, which in reality should be true is classified as false, it would be recognised as a false negative (FN) classification. If it was the other way around, there would be a false positive (FP) classification. Should the classifications of the samples be correct for either true or false, the classifications of the samples would be recognised as true positive (TP) or true negative (TN), respectively.

Given a data set of n samples with their true labels and their predicted labels, the accumulation of the TP, FP, FN, and TN would be displayed in a 2x2 grid confusion matrix. The columns of the matrix would represent the predicted labels and the rows represent the true labels. Consider **Table 2.2** for reference. The values in the confusion matrix may be used to calculate other ways of measuring the performance of a model, such as the as the receiver operating characteristics (ROC) and its area under the curve (AUC).

Table 2.2: Layout of a confusion matrix.

		Predicted label	
		Positive	Negative
True label	Positive	TP	FP
	Negative	FN	TN

Receiver Operating Characteristic (ROC)

The Receiver Operating Characteristic (ROC) is a clear way of visualising and analyse the performance of the model. It plots the true positive rate (TPR) against the false positive rate (FPR), and the characteristics of the lines are given by changing the threshold for when to classify a sample as positive. For a threshold of 1 all the samples are classified as negative and the TPR and FPR becomes zero, which are given by **Eq.** (2.19) and **Eq.** (2.20), respectively.

$$TPR = \frac{TP}{TP + FN} \tag{2.19}$$

$$FPR = \frac{FP}{FP + TN} \tag{2.20}$$

Decreasing the threshold, more samples will be classified as positive and there will be an increase in TPR and FPR. With a threshold of 0 all samples are classified as positive and TPR and FPR will become 1. As seen in **Fig. 2.12**, the straight line indicates that the model predicts random, meaning the model are not able to separate the different samples from each other. A curved line is an indication that the model is able to distinguish the positive samples from the negative samples. It is preferable to have a characteristic where the curve has a steep increase from the beginning and then flattens out.

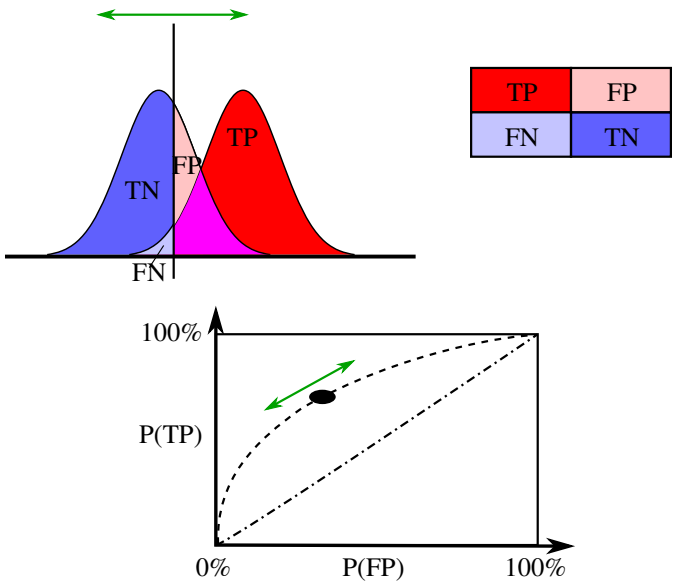


Figure 2.12: Defining mechanisms of the ROC curve. Illustration by Sharpr [53]

The area under the curve (AUC) is an extension to the ROC plot, giving a value between 0 and 1. Interpreting the value, an AUC of 1 means that the model is a perfect classifier, classifying all the samples correctly, an AUC of 0.5 means that the model is just as good as random guess, and an AUC under 0.5 is worse than random guess.

If the problem considered is a multiclass problem it is possible to average the scoring metric via One-versus-All classification [28] either by macro or micro averaging, given by **Eq.** (2.21) and **Eq.** (2.22), respectively. Micro averaging would weight each predictions equally, while macro averaging would take the mean of all precisions weighting all classes equal. The macro averaging approach would result in the most frequent class labels dominating the evaluation of the performance.

$$PRE_{micro} = \frac{TP_1 + \cdots + TP_k}{TP_1 + \cdots + TP_k + FP_1 + \cdots + FP_k} \tag{2.21}$$

$$PRE_{macro} = \frac{PRE_1 + \cdots + PRE_k}{k} \tag{2.22}$$

Matthew's correlation coefficients

Another metric that uses the elements from the correlation matrix is Matthew's correlation coefficients (MCC). It can obtain any value from -1 to 1, where 1 indicates a model predicting the expected label, -1 a model predicting the complete opposite than expected, and 0 no better than random guess. Compared to the ROC, the MCC also include the true negatives. This means that if more samples classified as negative are added, the MCC would increase. The MCC can be calculated from equation **Eq.** (2.23).

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.23}$$

2.5.2 Loss function

The task of training a model is conducted by providing a feedback on the performance. This feedback is used to update the model parameters. A set of functions are used to measure the deviation between the output values and the target. This deviation is called the loss, and the function is called a loss function. The main objective of the loss function is to minimise the loss during training. A decrease in loss indicates that the model is learning the variation within the data set used for training. Common loss functions are outlined below.

Mean Squared Error (MSE)

The mean squared error (MSE) is a basic loss function that works well on regression problems [27]. It is defined as follows

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(f^*(x^{(i)} - f(x^{(i)};\theta))^2 \tag{2.24}$$

where $f^*(x)$ is the target function to be learned, and $f(x;\theta)$ is the function provided by the model. The parameters θ will be adapted by the learning algorithm to make f as similar as possible to f^*.

Softmax

A common last layer activation is the softmax function. It provides a probability of each class instead of choosing one. The softmax is optimised by minimising the log-likelihood function **Eq.** (2.25), also known from information theory as a variation of cross entropy.

$$L_i = -log\left(\frac{e^{\theta_j^T x_i}}{\sum_{l=1}^{C} e^{\theta_l^T x_i}}\right) \tag{2.25}$$

2.5.3 Optimiser

In deep learning the task of optimisation refers to the minimisation or maximisation of the loss function mentioned in chapter 2.4.2 [26]. The optimisation is generally gradient based, meaning the derivative of the loss function is used for minimisation to try to reach the global minimum. The gradient to be computed is on the shape **Eq.** (2.26)

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\nabla_\theta L(x^{(i)}, y^{(i)}, \theta) \tag{2.26}$$

where m is the number of samples, L is the per example loss, $x^{(i)}$ is the sample data, $y^{(i)}$ is the true target of the sample, and θ is the optimisation variable. The most basic optimiser algorithm is the stochastic gradient descent (SGD). It only uses some of the samples for estimating the gradient RMSProp is the ”go-to optimisation method” in deep learning [26], and is a variant of SGD. An optimiser with high reputation is the Adam optimiser. It is similar to RMSProp but with momentum, and is empirically tested more effective than other optimisation algorithms [54].

2.5.4 t-SNE

t-distributed Stochastic Neighbour Embedding (t-SNE) is an effective technique to reduce the dimensions of a data set with high dimensional data onto lower dimensions. The closeness of two samples in the higher dimension will be kept through the transformation. However, samples or points that are far away in lower dimensions is not an indication of faraway relationship in higher dimensions. This technique minimises the Kullback-Leibler divergence on the high dimensional point distribution and a proposed low dimensional point distribution to find a suitable projection of the original points onto the lower dimension.

Chapter 3

Method

This thesis consists of analysis of results obtained from multiple tests related to sequential deep learning architectures. Supplementary illustrations are found in appendix A. Source code are placed in appendix B. The full process is outlined in **Fig. 3.1**, illustrating the steps taken. The main points of the steps are described in more detail in the sections of this chapter.

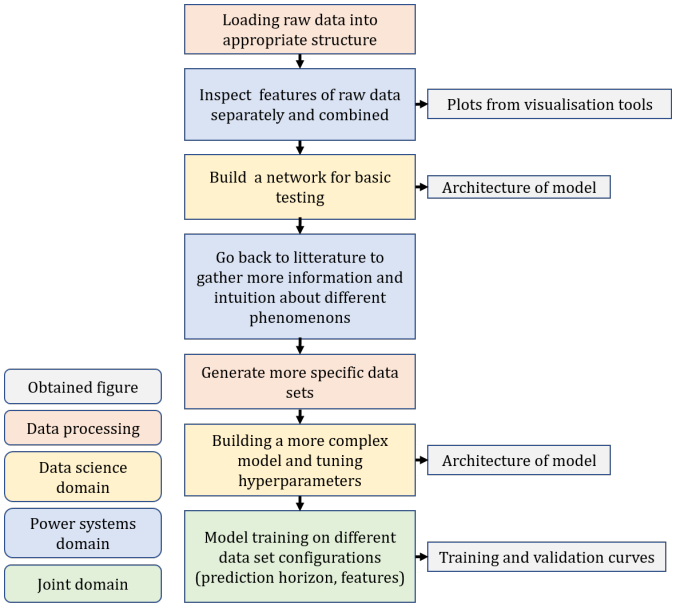


Figure 3.1: Outline of the test process. Boxes with text on the left hand side describe the colour coding.

3.1 Data Collection

The data used for training and testing is real, historical data obtained from the Norwegian electrical power system. The fault events have been detected using the software *'Automatisk Hendelsesanalyse'* (AHA) developed by SINTEF Energy Research [55]. The fault event types include interruption, voltage dip, rapid voltage changes (RVC), and earth fault. The distribution of faults have been explored using the Dynamic data set generator (DDG) GUI, also developed by SINTEF Energy Research. This software has the enables the selection of parameters to use, such as resolution, duration, voltage or current represented in RMS or waveform, and phase or line quantities.

Table 3.1: Time parameters used to generate the initial data sets.

Resolution	500 milliseconds
Duration	10 minutes
Buffer	2 minutes
Transient	10 minutes

The time parameters used to generate the initial data sets are listed in **Table 3.1**. Duration is the time period from the beginning of the fault event and back in time, buffer includes more data points after the beginning of the fault event. Transient defines the time after the fault where nominal operation has been reached after a fault clearance. Consider **Fig. 3.2** for illustration. The duration, buffer and resolution give the number of data points per feature. Using **Eq.** (3.1) the number of data points and subsequently rows become 1440.

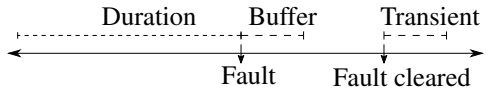


Figure 3.2: This figure illustrates the time parameters used to generate the initial data set from DDG.

$$number\ of\ data\ points = \frac{duration\ in\ seconds + buffer\ in\ seconds}{resolution\ in\ milliseconds} \times 1000 \quad (3.1)$$

The physical parameters used to generate the data sets were:

- cycle-by-cycle RMS line voltage
- cycle-by-cycle RMS phase voltage
- cycle-by-cycle RMS phase current
- cycle-by-cycle RMS IDH of line voltage
- cycle-by-cycle RMS IDH of phase voltage
- cycle-by-cycle RMS IDH of phase current
- cycle-by-cycle active power
- cycle-by-cycle reactive power

where the voltages and the currents represents all three phases, and harmonics up to the 19th are included. This yield a total of 189 raw data features. The structure of the data sets were on the form presented in **Table 3.2**.

Table 3.2: Example of structure of individual sample data set used to generate the full data set.

fault_detection	True
fault_type	Avbrudd
fault_time	2018-01-28 12:28:51
start_time	2018-01-28 12:28:50
end_time	2018-01-28 12:28:51
duration_sec	1.0
duration_days	1.157e-05
resolution_ms	1
Time_buffer_sec	0
Time_transien_sec	0
N_points	1000
node	*<system operator>*

Time [s]	rms_V1_AVG	rms_V2_AVG	rms_V3_AVG
.	.	.	.
.	.	.	.
.	.	.	.

The data samples in the data set were extracted and concatenated into a multidimensional Numpy array on the form (sample, time step, feature) and stored locally in a hdf-file for easy access.

3.1.1 Analysis of raw data

An interactive visualising tool was developed using the plotly library [56] in python for inspection of the data sets. The tool may be used to plot the harmonic components related to an event in an EEG inspired plot for easy investigation of the individual components. **Fig. 5.4** in appendix A displays the interface. See functions on lines 56, 185 and 196 in **Code 5.2** in appendix B, and appendix A for visual examples.

The balanced data set contains 8414 samples from different system operators (SOs). There was a relatively high variation among the distribution of the samples. This was mostly due to the variation of accessible measuring nodes.

3.2 Pre-processing

Little pre-processing has been applied to the data set since the harmonic components are already scaled to a value between zero and one, with regards to the fundamental of the original signal. Standardising the active and reactive power as well as the currents and voltages were performed so that only the variation was kept. Filtering for smoothing the time series was also conducted.

3.3 Tests

Multiple exploratory tests were conducted, and results from these are listed in chapter 4 with supplementary results in appendix A. The tests conducted were:

- Pre-training versus no pre-training of the composite model
- Testing of physical parameter features
- Metric development regarding prediction horizon
- Comparison of model architectures (sequence-to-sequence composite, LSTM, and GRU)
- Analysis of state outputs.

Metric used to evaluate the tests were accuracy and Matthews correlation coefficient (MCC) for prediction, and mean square error loss (MSE) for the reconstruction. ROC-plots with AUC were used extensively for further evaluation of the performance. They can be interpreted as follows: When the curve starts ascend equally linear to the axes, the probability distribution of positive samples and negative samples are blended, *i.e.* random selection occur. As a result of this interpretation, a steep upward line is the ultimate result. 5-fold cross validation were performed during all tests, except the full test when the whole training data set were used. The main tests of this thesis will be conducted using the composite model architecture.

3.3.1 Models

The machine learning models were developed using the Keras API [57] with TensorFlow [58] as backend. Consider **Code 5.4** and **5.5** for model code.

The generic LSTM model function is constructed in a such way that layers and cells can easily be defined by the user, and number of input sequences and the length of the input vector is defined by the sample. The output size is defined by the number of categories in the training set.

As mentioned earlier, this thesis presents a new architecture for time series prediction; the sequence-to-sequence (s2s) Autoencoder. It consists of an encoder part and a decoder part. Firstly, the s2s Autoencoder is trained on encoding the input sequence and then reconstructing it with the decoder, as seen in **Fig. 3.3**. The training set consists of both faults and non-faults equally distributed. The data set is said to be balanced. The true output data will be the reverse of the input data sequence, as proposed by Serivastava *et al.* [51].

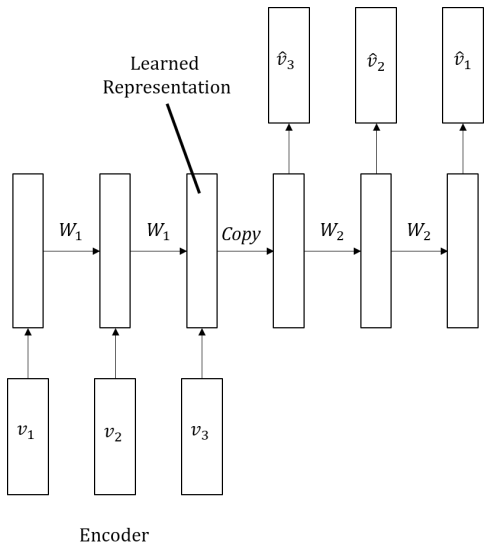


Figure 3.3: Sequence-to-sequence Autoencoder model. Inspired by Serivastava *et al.* [51].

Secondly, after training for several epochs the encoder is isolated and the output of the encoder, the code, is fed into a classifier for prediction. The encoder layer is set to non-trainable and the classifier is trained to predict faults. Thirdly, the encoder and classifier are recombined with the decoder, so that the encoder output is fed to both the classifier and the decoder. A sketch of the model setup is seen in **Fig. 3.4**. The loss functions of the classifier and the decoder are combined, so that the reconstruction is preserved which might keep the model from predicting only one of the possible class categories. The two loss functions can be weighted differently depending on the importance of their loss contribution.

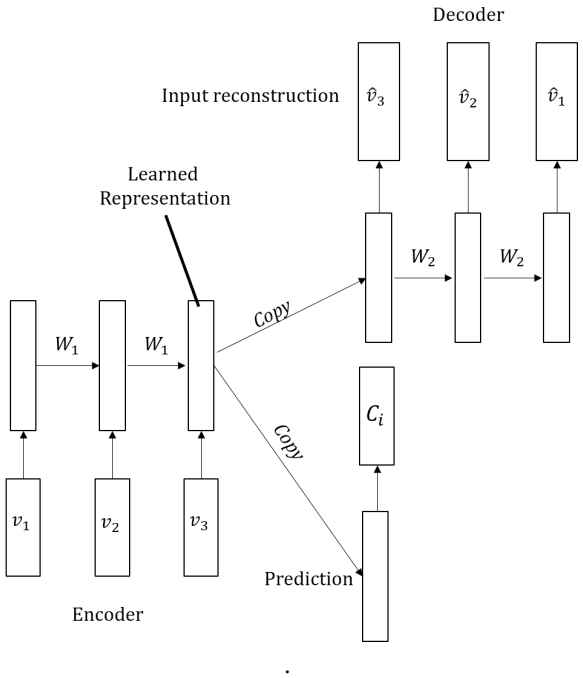


Figure 3.4: Composite sequence-to-sequence Autoencoder model with prediction branch. Inspired by Serivastava *et al.* [51]

The initial idea was to use an SVM as a classifier. However, a comparison of the SVM classifier and a generic softmax with cross entropy loss showed a slightly better result in performance for the softmax classifier. As a result, the softmax was chosen as the last layer activation function in the classifier part.

3.3.2 Input Data

The input data is selected to have 120 time steps (a total of 60 seconds), and the prediction horizon is adjusted using a sliding window over the complete data set. The sliding window technique used is seen in **Code 5.3**, line 51 in appendix B. The harmonic components up to 15 were chosen because when inspecting several samples, the harmonic components over 15 did not seem to pass the threshold of the Elspec measurement equipment. This is due to a user selected threshold of for instance 0.1% is filtering out noise returning a value of zero. This also reduces the amount of features that need to be processed, resulting in lower memory- and time consumption. **Fig. 3.5** illustrates how the input vector is fed into the model. Lines 141-191 in **Code 5.3** outline the process of generating the input data set.

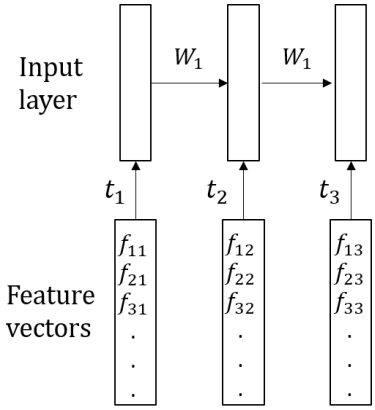


Figure 3.5: The figure shows how the input vectors were constructed using each element in the time series sample as the respective element in the vector.

3.3.3 Training, validation and testing

Train-test split is set to be 90/10 of the prepared data set. During training, validation is set to be 10% of the input set. **Fig. 3.6** and **3.7** show the strategies and steps taken for testing the model.

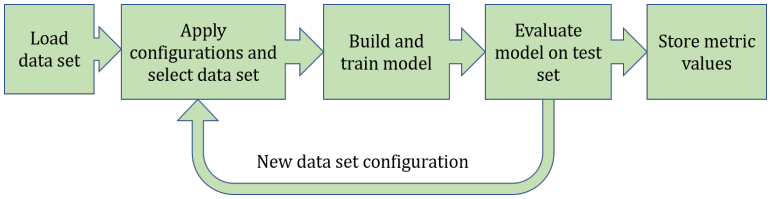


Figure 3.6: General strategy for model testing.

Since various configurations of architectures have different characteristics [27], it is important to tune the hyperparameters of the model so that the model performs at its best. Possible tuneable hyperparameters is learning rate, number of units, and regularisation parameter.

The tests were conducted using 5-fold cross validation, for checking the robustness of the model. The results from these tests might be biased since the data set used for testing were not fully isolated from the test data, meaning decisions made during training may be based on data intended for testing. This is data which should have been completely unseen, and may prevent the model to generalise well.

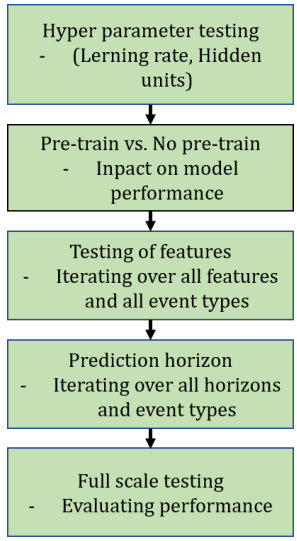


Figure 3.7: Illustration of steps and testing strategy followed when conducting the testing of the model.

Chapter 4

Results and Discussion

In this chapter an analysis of the results will be presented, and further work and approaches discussed. The chapter is divided into the respective tests. Tests were conducted on the fault types interruptions, voltage dips, and earth faults. Only results from one fault type are presented in the text, additional results are found in appendix A.

4.1 Raw data analysis

The first approach was to analyse the raw data. Samples containing fault events and nominal operation data were randomly selected and visualised by plotting the time series. Examples of the tool used for visualisation are displayed in **Fig. 4.1**. Consider **Fig. 5.1**, **5.2** **5.3** and **5.4** in the appendix A for more examples. These visualisations were performed to form intuition about the conditions before and after a fault event. This information was used to determine the quality of the data set and plausible candidates for features among the physical parameters.

Some samples during the inspection were discovered to contain incorrect classifications. This applies to the fault type of interruptions, where the occurrence of the fault had already taken place, leaving the non-nominal values before the fault. Another discovered flaw in the data set was samples containing NaN values. The reason for this might be due to no properly logging of data by the measurement unit. Despite this, none of theses samples were removed from the data set.

There were some traits that repeated themselves. It was observed in some samples that the active- and reactive power, and consequently the current, were reduced to zero minutes before an interruption occurred. Possible explanations for this behaviour are activation of protection equipment cutting the power delivery, or it could be a result of a scheduled interruption.

Overall, the samples contained a significant amount of noise and fluctuations. As an attempt to reduce the noise level, the Butterworth filtering technique was applied to each feature, as exemplified in **Fig. 4.2**.

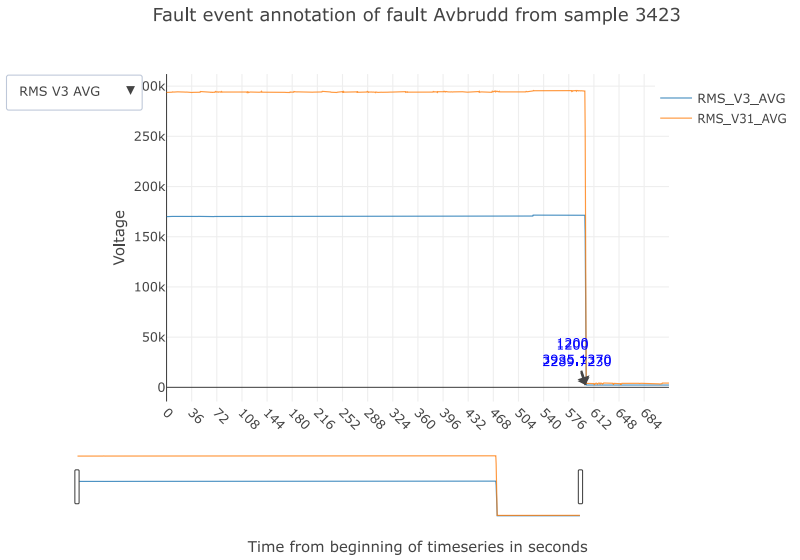


Figure 4.1: Example from the visualisation tool. Main plot is the portion in focus. Subplot is an overview of the whole sample. The plot shows a sample from one phase and one line voltage development before an interruption.

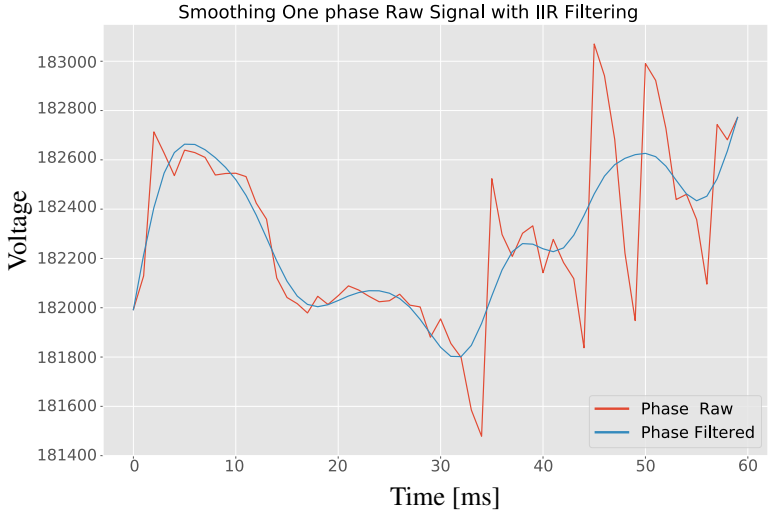
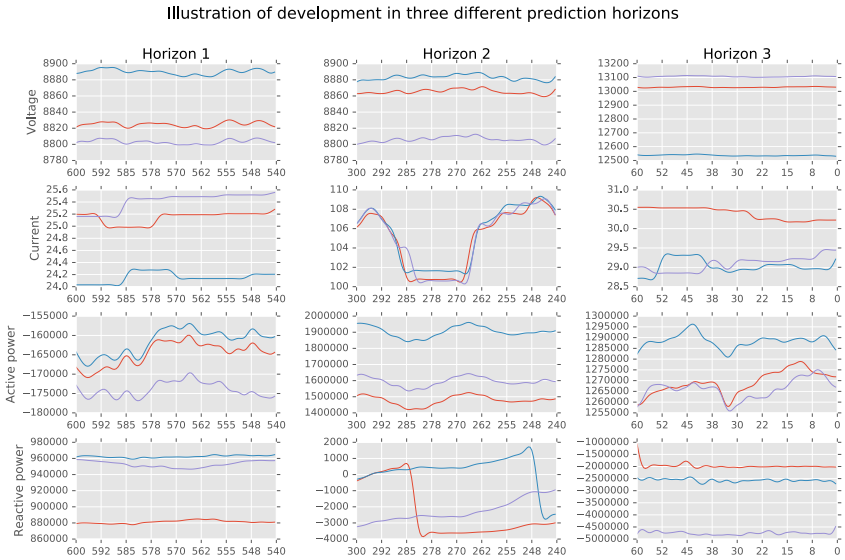
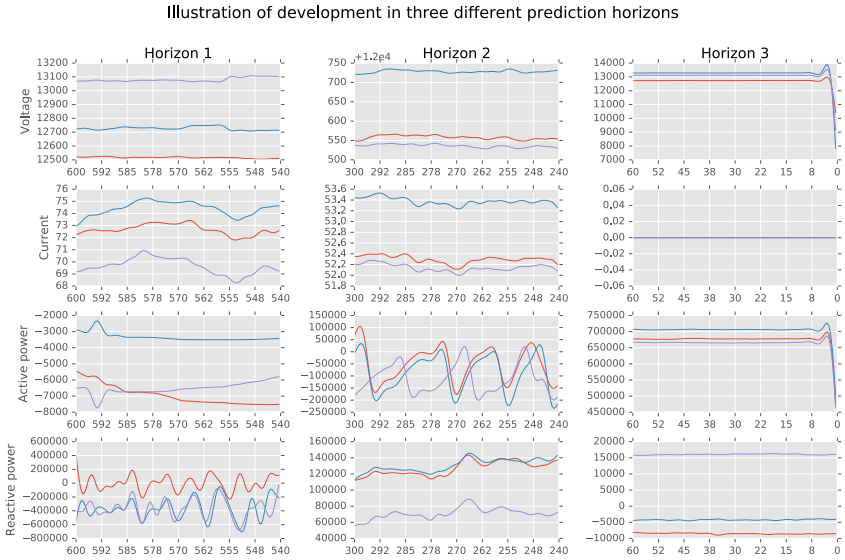


Figure 4.2: Comparison of raw signal and filtered signal where Butterworth filtering has been applied. The y-axis describes voltage and the x-axis denotes time in seconds.

The data set used as input data was a generated sub-set of the initial data set. In the generation process, the event types and features were selected. The little amount of pre-processing, such as filtering and standardisation, were applied to each feature column in the sub-data set after the generation. This was done due to the fact that adjusting the prediction horizon or length of the time series, the distribution of values within each feature will change, and would be more realistic when it comes to real world application. The generation of the sub-set was determined by a sliding window approach. The size of the window was set to cover 60 seconds. With a resolution of 500 milliseconds of the initial data set the time series sequence in the sub-set contained 120 steps. The sliding window approach was chosen as a measure to investigate the impact of the prediction horizon on the performance of the model. The input samples were also inspected to compare the nominal data to the fault event data, as seen in **Fig. 4.3a** and **4.3b**. The sub-sets of the features are plotted using three different prediction horizons. Horizon 3 include the fault event, which can be spotted in **Fig. 4.3b**. Horizons 2 and 1 have prediction horizons 240 and 540 seconds, respectively. At a glance, there are no immediate difference between the fault event and the nominal operation samples except at time of the fault event. Despite this, the harmonic components of the current and voltage, as exemplified in **Fig. 5.4** show an increase in the RMS value several seconds before the fault. This behaviour, and the known presence of harmonic components related to various conditions in the electrical power system, makes the harmonic components a reasonable feature candidate. The results of the feature test are presented further below.



(a)



(b)

Figure 4.3: Illustration of the development of a non fault, a), and an interruption sample, b), in the same spacing as some of the prediction horizons. The plot displays the voltage, current, active power and reactive power of the same sample in descending order. There are three phases in each plot. The x-axis states the time from the fault event in seconds. Horizon 3 include the fault event. Horizons 2 and 1 have prediction horizons 240 and 540 seconds, respectively.

4.2 Model Architecture

The architecture of the model used to conduct the main tests are shown in **Fig. 4.4**. It is a modification of the sequence-to-sequence natural language processing (NLP) architecture in [59] combined with the idea in [51]. The idea is to send the output of the encoder part to a prediction part and a reconstruction part making a composite model. The selection of hidden layers were inspired by the approach in [22] since this was a study diagnosing railway track circuit faults using RNNs. For reference, **Fig. 5.10** and **5.9** in appendix A show the isolated reconstruction and classifier parts of the model.

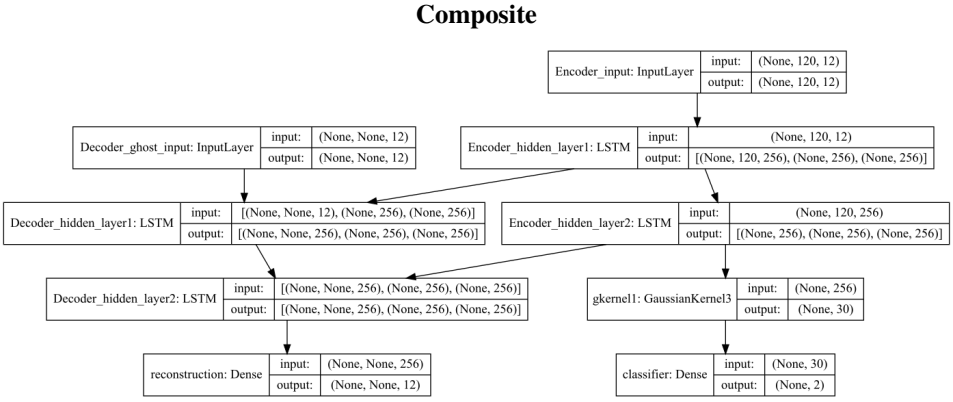
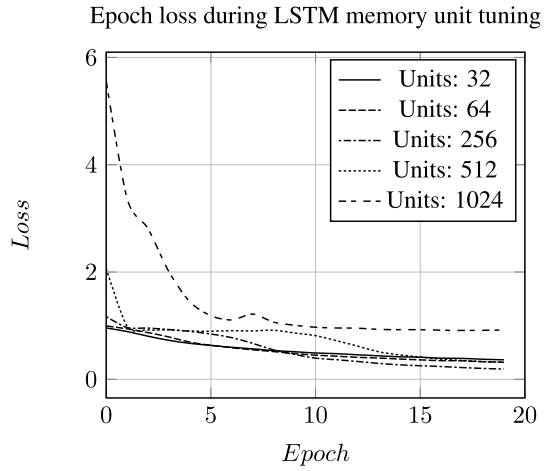


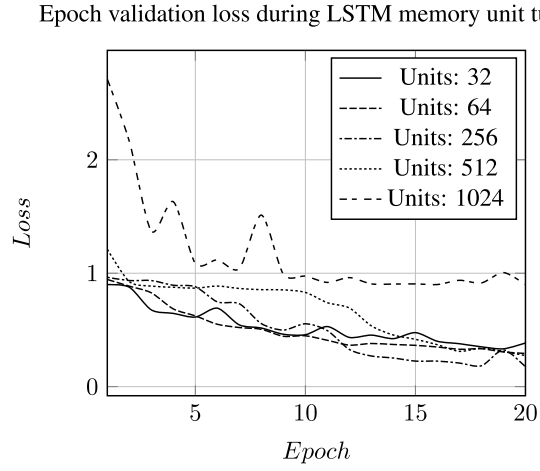
Figure 4.4: Structural architecture of the whole composite model, where both the reconstruction part and the classifier part is connected.

The following step after constructing the model was to choose suitable hyperparameters. In this case, the hyperparameters searched for were learning rate and hidden units, also known as memory cells. In this step, a training epochs of 20 were used to narrow down the alternative values before conducting a grid search using 100 epochs to find the best combination. First the hyperparameters of the reconstruction part were found. The tuning was conducted on 3773 samples of nominal operation samples containing 60 time step points and three features of line voltage, and validated on 420 samples.

Inspecting **Fig. 4.5**, the number of units which yield the best loss over the training and validation period is 256 units. The learning rate used for training is obtained from considering **Fig. 4.6**. Learning rate of 0.001 seems to be converging while learning rate 0.01 still declines.



(a)



(b)

Figure 4.5: The development curve of six hidden unit configurations over 20 epochs. 2048 units curve is not included in the plot because it deviated a lot from the others, however, it tended to converge to the same as 1024 units curve. a) Training loss. b) Validation loss

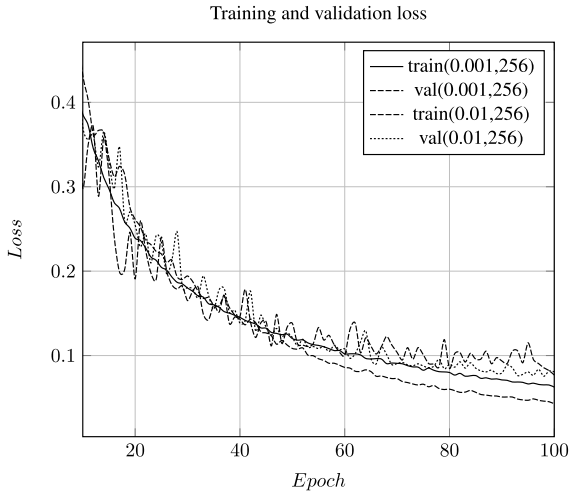


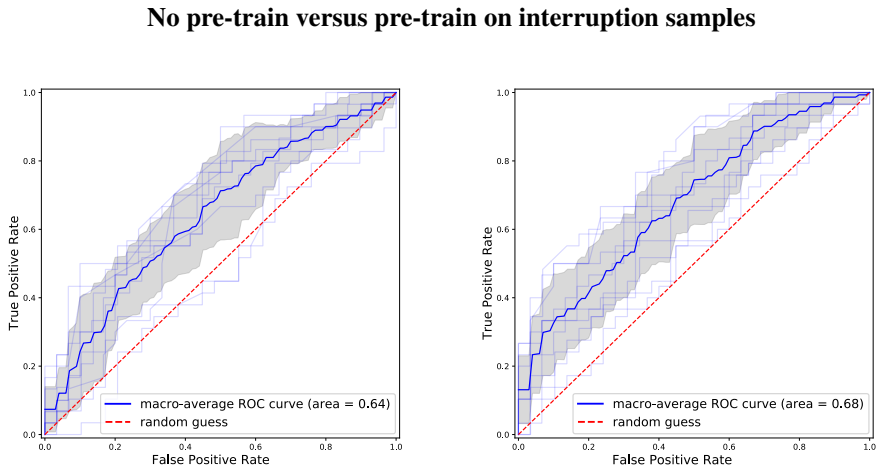
Figure 4.6: The training and validation loss during grid search for the hyperparameters learning rate and memory unit. Parameters tested for learning rate was 0.001 and 0.1, for memory units it was 32, 64, 256, 512. Of the two learning rate runs 256 and 512 memory units performed the best. In the plot it is observable that the two curves with learning rate 0.01 starts to deviate. This is a sign of overfitting. The curves with learning rate 0.001 appears to converge at the same rate. A learning rate of 0.001 will be chosen for the LSTM part of the model.

When finding the optimal learning rate for the classifier, the s2s model was first trained using the previous obtained best parameters. A balanced data set of 296 of none faults and interruptions were used. The model was trained on 266 samples and validated on 30. After a selection run of 20 epochs the interesting learning rates appeared to be 0.01 and 0.1. New training run of 100 epochs was then conducted. The loss functions were equally weighted in the contribution to the total loss used in the optimising process. Loss function used for the classifier was cross entropy loss with adam optimiser and softmax at the end layer. For reconstruction MSE was used as loss function and rmsprop for optimisation, with no activation due to better reconstruction.

The model tended to overfit and converge to a training prediction accuracy of 96% and a validation accuracy of 47% with a reconstruction loss of the training declining. Validation reconstruction loss increased while converging at the same rate as the prediction accuracy.The overfitting may be solved by adding regularisation of the weights in the hidden layers, or force the outputs to be sparse by adding a penalty term on the activation of the units in the network [26]. This was applied without any noticeable improvements in performance. It was decided to reduce the number of epochs to be between 5 and 50 during the tests, to reduce overfitting, and due to time considerations.

4.3 Pre-training Testing

Finding the best hyperparameters, the next step was to investigate the impact of pre-training the different parts of the model. The results from this test show that there is an improvement in performance when pre-training a model, as shown in **Fig. 4.7**. More examples may be found in **Fig. 5.5** and **5.6** in appendix A. Particularly voltage dip prediction performance increased as seen in **Table 4.1** where the standard deviation is reduced by half. The parameter setup for this test is found in **Table 4.2**. Pre-training each part of the model before fine tuning the full model are in general conducted as a form of initialising the weights before training. Other research has shown good results from pre-trained models as in a study by Qi *et al.* [60]. This is observed both in **Fig. 4.7**, **5.5**, and **5.6**, where the AUC is consequently higher for pre-trained models.



(a) no pre-train **(b)** pre-train

Figure 4.7: a): Results from 5-fold cross validation and no pre-training. b): Results from 5-fold cross validation. The model was trained on a balanced data set of nominal operation and interruption samples. The opaque blue lines are the ROC curves for each class, the solid blue is the macro average of all the curves, and the dashed line corresponds to random guess. The gray area represents the standard deviation of the calculation of the average. The area under the curve (AUC) is also computed and displayed in the plot.

Table 4.1: Results from cross validation test.

	Interruption		Voltage dip		Earth fault	
	No pre-train	**Pre-train**	**No pre-train**	**Pre-train**	**No pre-train**	**Pre-train**
Classification accuracy	57.38% +/- 6.79	62.13% +/- 5.6%	60.87% +/- 2.99	63.26% +/- 1.45%	64.41% +/- 2.13%	64.84% +/- 1.56%
Classification MCC	0.15 +/- 0.14	0.24 +/- 0.11	0.22 +/-0.06	0.27 +/-0.03	0.29 +/-0.04	0.30 +/- 0.03
Reconstruction loss	0.57 +/- 0.02	0.59 +/- 0.02	0.51 +/-0.03	0.53 +/- 0.01	0.57 +/- 0.1	0.55 +/- 0.01
Train/Test	236/36	236/36	3070/768	3070/768	2220/556	2220/556
Batch size	32	32	64	64	64	64
Epochs	(10/10/30)	50	(10/10/10)	30	(10/10/10)	30

Table 4.2: Parameter setup for cross validation test

Number of cross validations	5
Optimizer reconstruction	rmsprop
Optimizer classificator	adam
Optimizer composite	adam
Activation classificator	softmax
Loss classificator	categorical cross entropy
Features	harmonics V12 V23
Loss weights	(1, 1)
Latent dimentions	256
Random seed	42
Preprosessering	smoothing, standardising
Data set	balanced
Prediction horizon	10 sec.
Length timeseries	60 sec.

4.4 Feature Testing

The features used in the previous tests were randomly chosen. However, different features may not share the same characteristics or information. Therefore it was interesting to test the various physical features and phases to see their influence on the performance of the model, and to see if the features of the harmonic components were as good as foreseen. From the result presented in **Fig. 4.8** and **Table 4.3**, the harmonic components of the current appear to contain information that enables the model to perform better in predicting events. They are also relatively easy to reconstruct, as seen from the same tables, which indicates a possible sparse data set. Other features that appear to contain useful information is the harmonic line voltages. Training only on RMS voltages, currents or power did not yield a particularly good performance. Due to the lack of sparsity of these features, the reconstruction part of the model should be trained for several epochs prior to full training to be able to learn some of the complex structures of the features. An identical test was conducted on voltage dips and earth faults with similar results.
See **Fig. 5.11**, **5.12**, **5.1**, and **5.2** for reference. The features that were selected to conduct the remaining tests were harmonic components of the three phase currents and the harmonic components of the three line voltages.

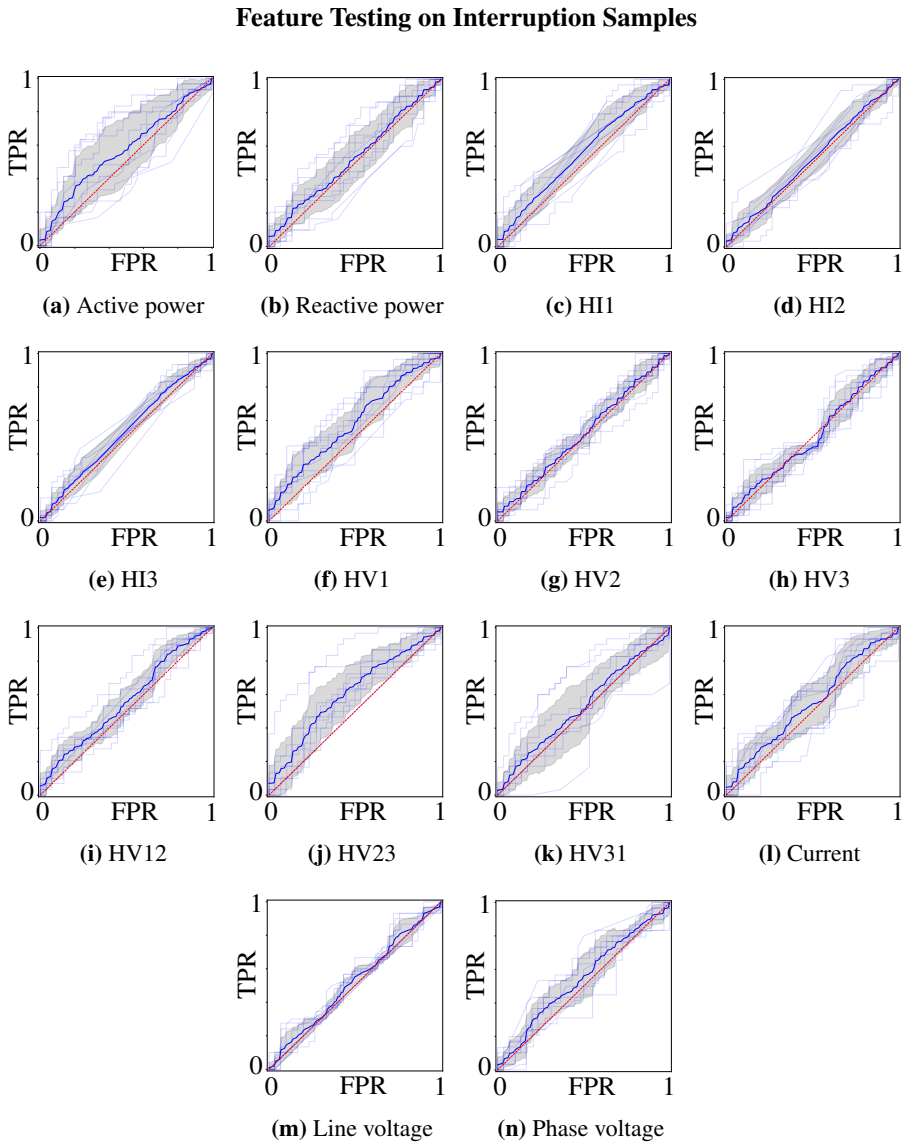


Figure 4.8: ROC-plots over the model trained on the corresponding feature a-n. The training set contained non-faults and interruptions. See **Table. 4.3** for metric results for each feature. TPR and FPR denote true positive rate and false positive rate, respectively. HIn refers to the harmonic components of phase current n. HVn refers to the harmonic components of phase voltage n. HVnm refers to the harmonic components of the line voltages in a three-phase system.

Table 4.3: Results from training on individual features classifying non-faults and Interruptions. Each fold was trained for 10 epochs, no pre-training of the model, with batch size of 64 and train/test samples of 236/36. The harmonic features contains six harmonic components; 2nd, 3rd, 5th, 7th, 9th, 11th and 13th, the rest consists of three columns, one for each phase. See **Fig. 4.8** for ROC-plots of each feature. HIn refers to the harmonic components of phase current n. HVn refers to the harmonic components of phase voltage n. HVnm refers to the harmonic components of the line voltages in a three-phase system.

	Interruption		
Features	**Classification Accuracy**	**Classification MCC**	**Reconstruction Loss**
Active power	52.99% (+/- 6.87%)	0.06 (+/- 0.14)	0.93 (+/- 0.03)
Reactive power	49.37% (+/- 3.49%)	-0.01 (+/- 0.07)	0.93 (+/- 0.02)
HI1	57.11% (+/- 3.90%)	0.14 (+/- 0.08)	0.22 (+/- 0.03)
HI2	48.33% (+/- 1.83%)	-0.03 (+/- 0.04)	0.23 (+/- 0.04)
HI3	53.37% (+/- 2.81%)	0.07 (+/- 0.06)	0.24 (+/- 0.02)
HV1	53.71% (+/- 6.10%)	0.07 (+/- 0.12)	0.58 (+/- 0.03)
HV2	51.36% (+/- 1.66%)	0.03 (+/- 0.03)	0.54 (+/- 0.02)
HV3	50.64% (+/- 4.92%)	0.01 (+/- 0.10)	0.49 (+/- 0.02)
HV12	51.66% (+/- 4.23%)	0.03 (+/- 0.08)	0.59 (+/- 0.02)
HV23	60.41% (+/- 7.39%)	0.21 (+/- 0.15)	0.59 (+/- 0.02)
HV31	55.06% (+/- 6.33%)	0.10 (+/- 0.13)	0.60 (+/- 0.03)
Current	52.39% (+/- 3.83%)	0.05 (+/- 0.08)	0.91 (+/- 0.04)
Line voltage	49.33% (+/- 1.33%)	-0.01 (+/- 0.03)	0.93 (+/- 0.02)
Phase voltage	49.66% (+/- 0.69%)	*-0.01 (+/- 0.01)*	0.96 (+/- 0.05)

4.5 Model Evaluations

One important objective of this thesis was to investigate the prediction horizons impact on the performance. The test was conducted on all fault types using the best features previously found, obtaining individual metric values for each fault type. Each run were 5-fold cross validated, and the metric result was the average of the five validations. This was done to get a more robust result. **Fig. 4.9** and **4.10** show the developments of the ROC-curves of each fault event related to the various prediction horizons. The aim of most predictive models is to beat the base line chosen for the specific problem. In this case, by evaluating the results in **Fig. 4.11**, the model is outperforming the base line within a prediction horizon close to 30 seconds for all fault types. The same figure also shows an interesting development after 240 seconds, at 7 minutes. Here the metrics increase in the positive direction. This may indicate some characteristics in the harmonic content which appear four minutes before the fault event. Since this was discovered after the tests were concluded, further investigation will be conducted in later research.

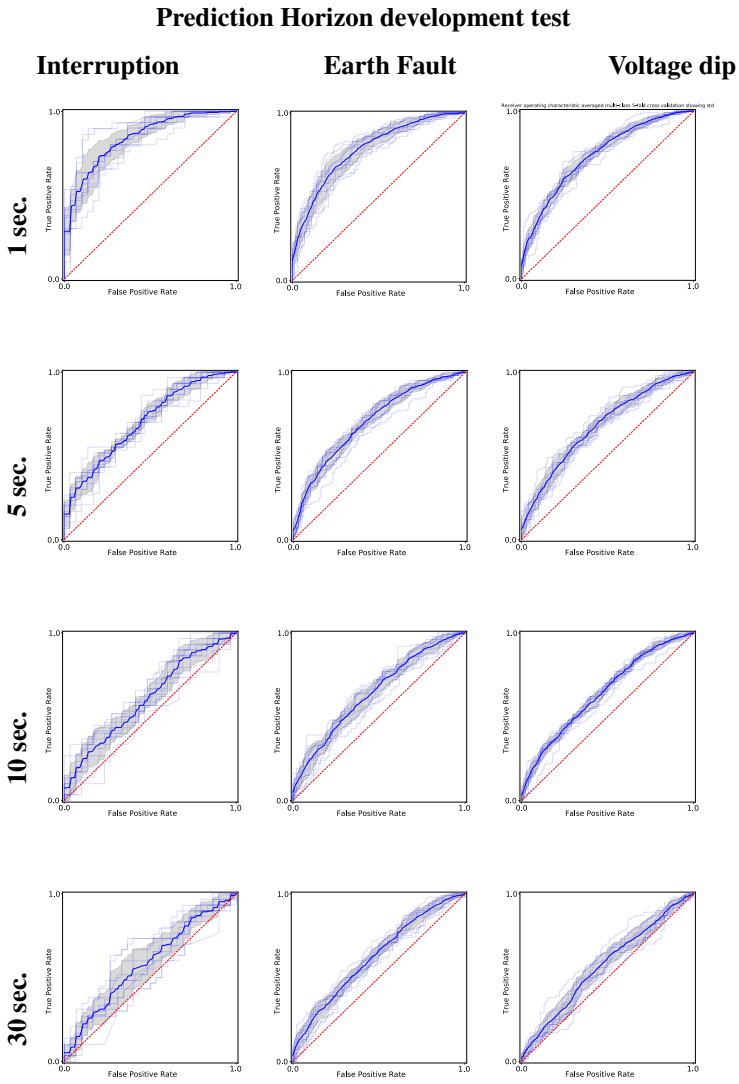


Figure 4.9: ROC-plot over model training on different fault event types and prediction horizons. The columns represents the fault event types, and the rows represents the prediction horizons. Vertical axis and horizontal axis represents the true positive rate (TPR) and the false positive rate (FPR), respectively, with axis going ranging from 0 to 1.

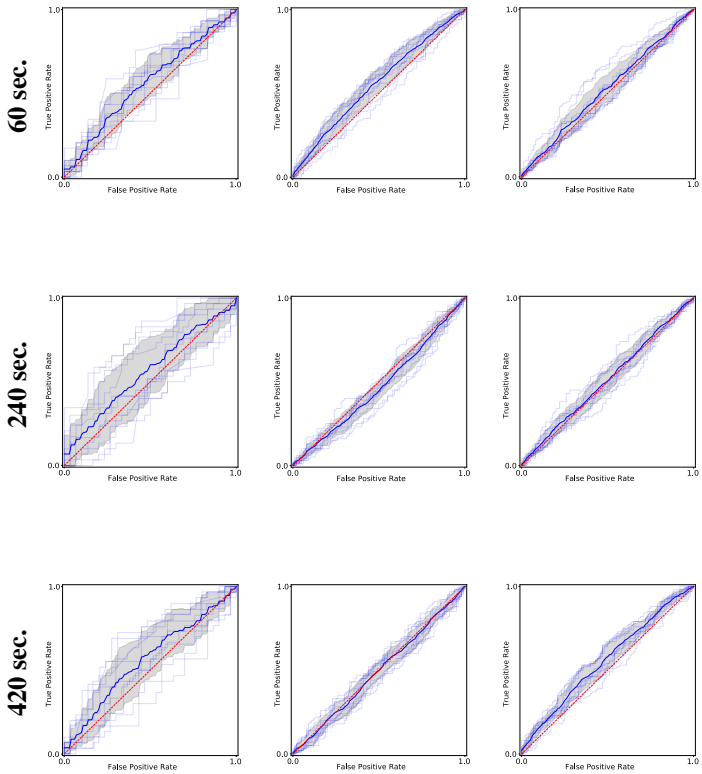


Figure 4.10: ROC-plot over model training on different fault event types and prediction horizons. The columns represents the fault event types, and the rows represents the prediction horizons. Vertical axis and horizontal axis represents the true positive rate (TPR) and the false positive rate (FPR), respectively, with axis going ranging from 0 to 1.

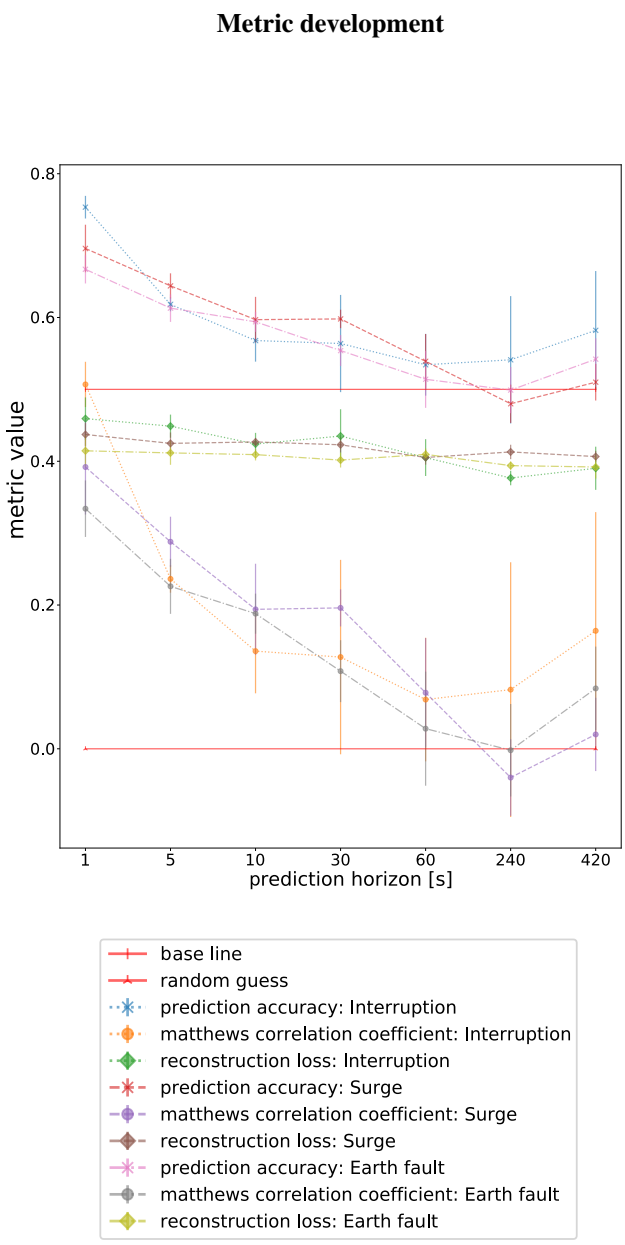
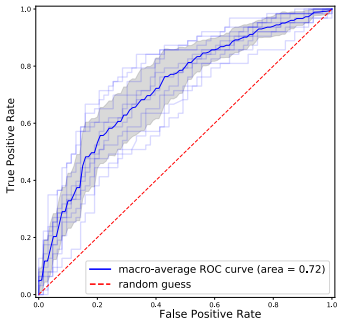


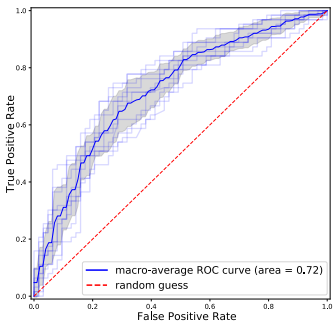
Figure 4.11: Development of evaluation metrics with respect to prediction horizon in seconds from fault event. The x-axis describes the horizon in seconds. Each fault event type were tested against each horizon using 2,3,5,7,9,11,13 harmonic from phase current, phase voltage, and line voltage. Pre-training was conduced with epochs 20,10,15. The metrics are the average from a 5-fold cross validation and are plotted together with the standard deviation. The base line corresponds to the naive classifier of just selecting one class and are related to the accuracy measure. Random guess corresponds to the Matthew's correlation coefficients implying a random classifier. Keep in mind that the x-axis is not linear.

Until this point, only the composite model has been tested. To understand its level of performance, other machine learning algorithms known for preserving the temporal dependencies were used for comparison. The results presented in **Fig. 4.12**, **4.13**, and **4.14** show some differences in performance both when predicting two classes of fault non-fault, and multiple classes. In the majority of the cases, both on validation and unseen data, the GRU architecture performed better compared to the LSTM. This is in accordance with the results from [50] where various RNNs were compared. It performed better both in predicting the faults, and time consumption vice. The composite model did perform slightly better on the more complex multiclass problem, tested on unseen data as shown in **Fig. 4.15** and **4.16**.

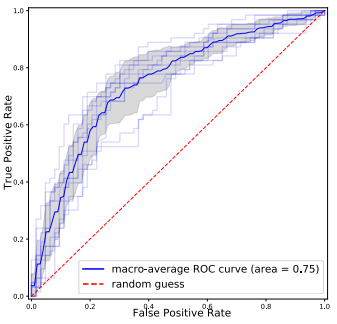
Comparison of Model Performance on Predicting Voltage dips During Cross-Validation



(a)



(b)

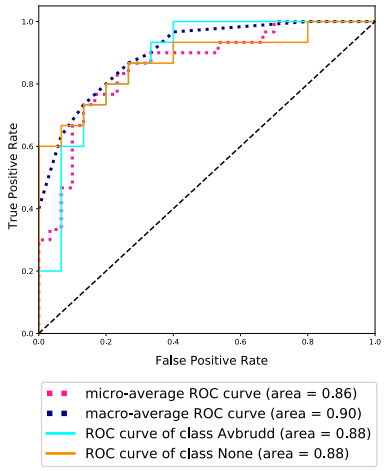


(c)

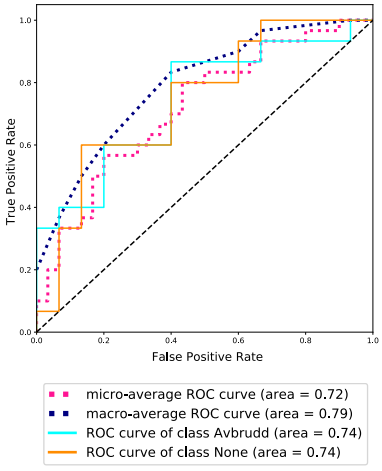
Figure 4.12: a): Composite model performance. b): LSTM model performance. c): GRU model performance.

The main results from testing the various models are as follows. Firstly, the models predict in most cases better than random. The models are able to confirm the existence of differences between fault types. This may be explored in more detail in further research. Secondly, tests have shown that the harmonic components of currents and voltages contain enough information by itself to be used as features. Feature engineering on other entities of physical parameters, such as voltage, may also increase the performance.

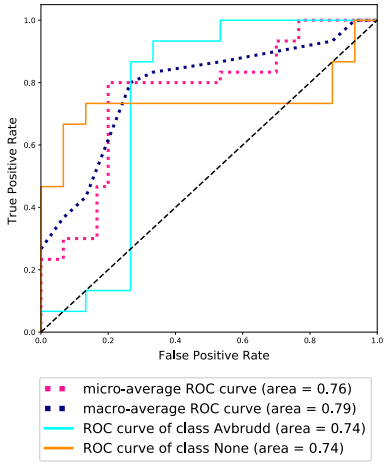
Comparison of Model Performance on Predicting Interruptions on Unseen Data



(a) Composite



(b) LSTM



(c) GRU

Figure 4.13: For this experiment each model was trained on the full training set, no validation performed under training, for 10 epochs and 50 units. They were then tested on a unseen test set. The data sets used contained non-fault events and Interruptions. Results are shown in the figure. ROC-plots displaying AUC for each class and the weighted ROC-AUC.

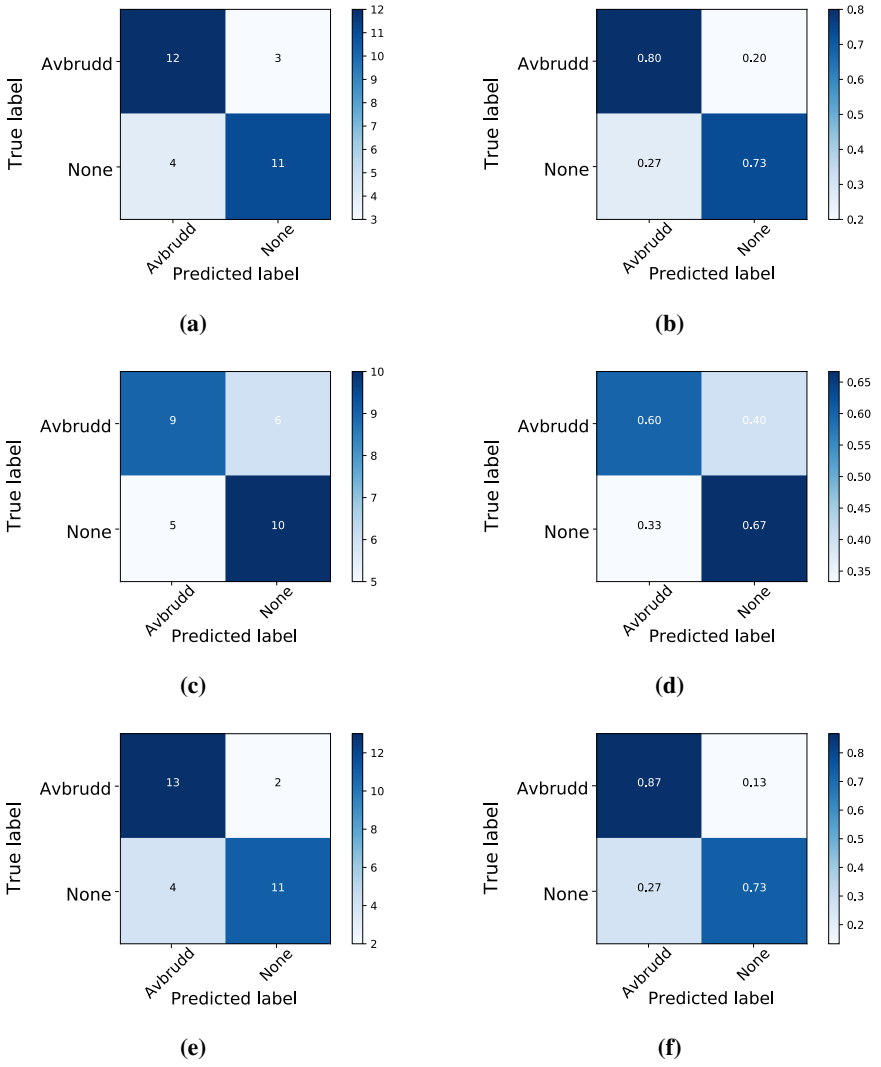
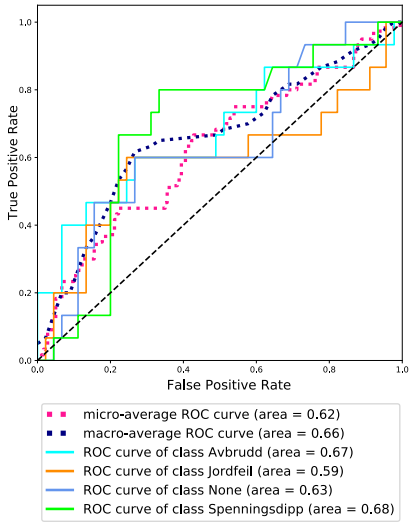
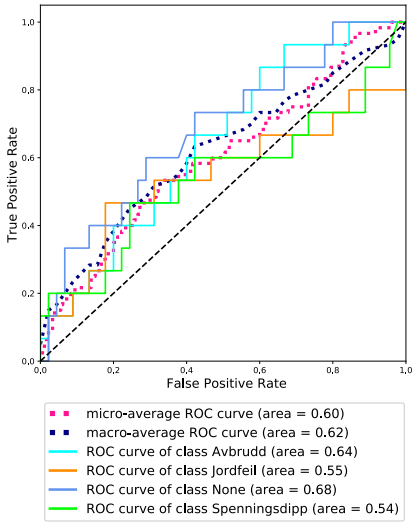


Figure 4.14: Predictions on unseen data of non-fault events and Interruptions. a) & b): Confusion matrix and Normalised confusion matrix over predicted events by composite model. c) & d): Confusion matrix and Normalised confusion matrix over predicted events by LSTM model. d) & e): Confusion matrix and Normalised confusion matrix over predicted events by GRU model

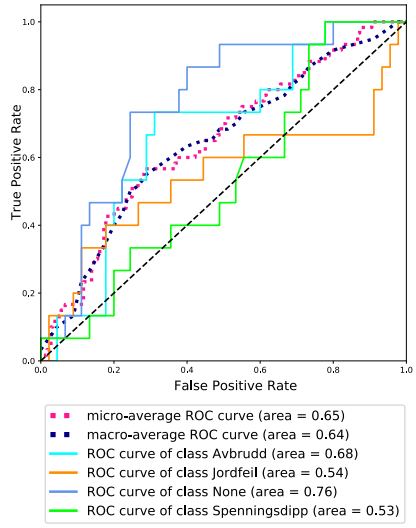
Comparison of Model Performance on Multiclass Prediction on unseen data



(a) Composite



(b) LSTM



(c) GRU

Figure 4.15: For this experiment each model was trained on the full training set, no validation performed under training, for 10 epochs and 50 units. They were then tested on a unseen test set. The data sets used contained all available event types; non-fault, interruption, surge, and earth fault. ROC-plots displaying AUC for each class and the weighted ROC-AUC.

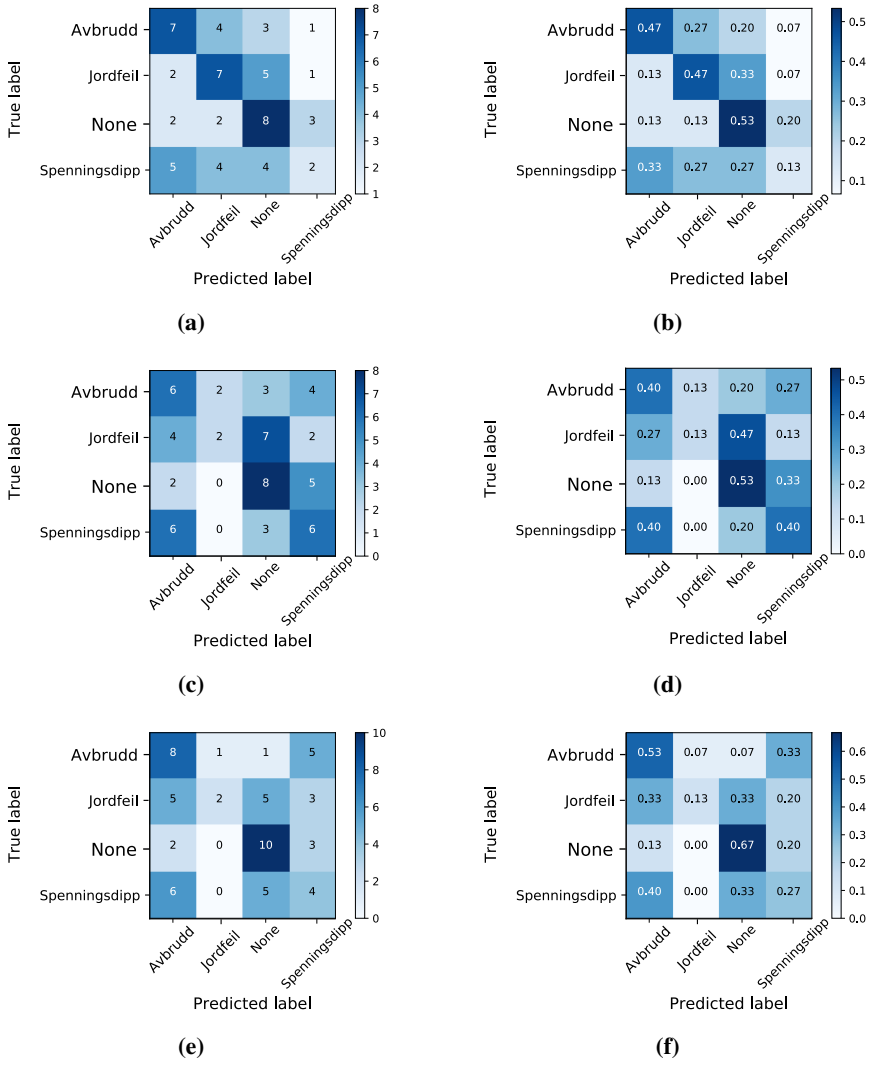


Figure 4.16: Predictions on unseen data containing all available event types; non-fault, interruption, surge, and earth fault. Results are shown in the figure. a) & b): Confusion matrix and Normalised confusion matrix over predicted events by composite model. c) & d): Confusion matrix and Normalised confusion matrix over predicted events by LSTM model. d) & e): Confusion matrix and Normalised confusion matrix over predicted events by GRU model.

4.6 Output analysis

"What does the model learn?", is a common question in machine learning, especially regarding networks with hidden layers. Researches use several techniques to inspect the models. Some approaches may be to visualise the weights and states in the network by plotting their raw or processed values. In this study, the activity output state and memory cell state of the last outputs were analysed by t-SNE dimension reduction [24] using the Barnes-Hut-SNE method [61] implemented in the Scikit-learn library [62]. The results are presented in **Fig. 4.17**, **4.18**, and **4.19**. Analysing the results in **Fig. 4.17** and **4.18**, it is clear that the model is able to distinguish between the samples. This is an expected result.

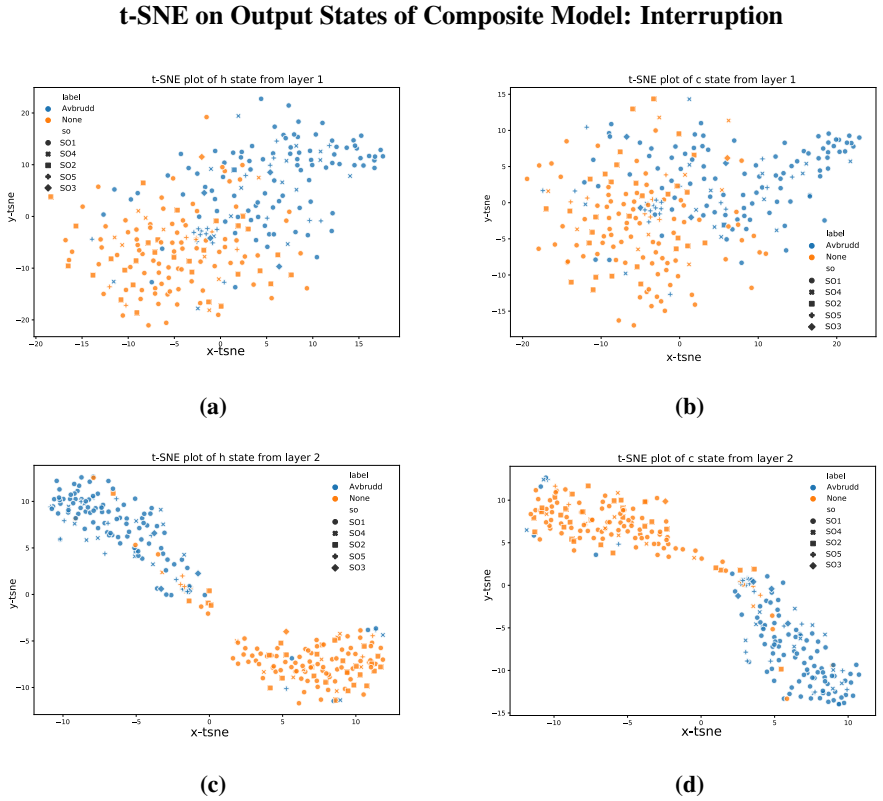


Figure 4.17: During this analysis the composite model was trained on non fault and interruption events. 'Avbrudd' and 'None' refers to interruption and nominal operation samples, respectively. 'SO' refers to system operator. a): t-SNE representation of the activity state of the last output of the first layer. b): t-SNE representation of the memory cell state of the last output of the first layer. c): t-SNE representation of the activity state of the last output of the second layer. d): t-SNE representation of the memory cell state of the last output of the second layer.

One curious notion is the clustering of the system operator marked by the plus sign in the middle of the top plots. This indicates that the signature of the operational state of the system operators is unique for their location or system, and must be taken into consideration when using data from various sources. This is in accordance with how the power system is designed. The power system is divided into sections, with different operating voltages, grounding schemes, and composition of equipment affecting the characteristics of the system.

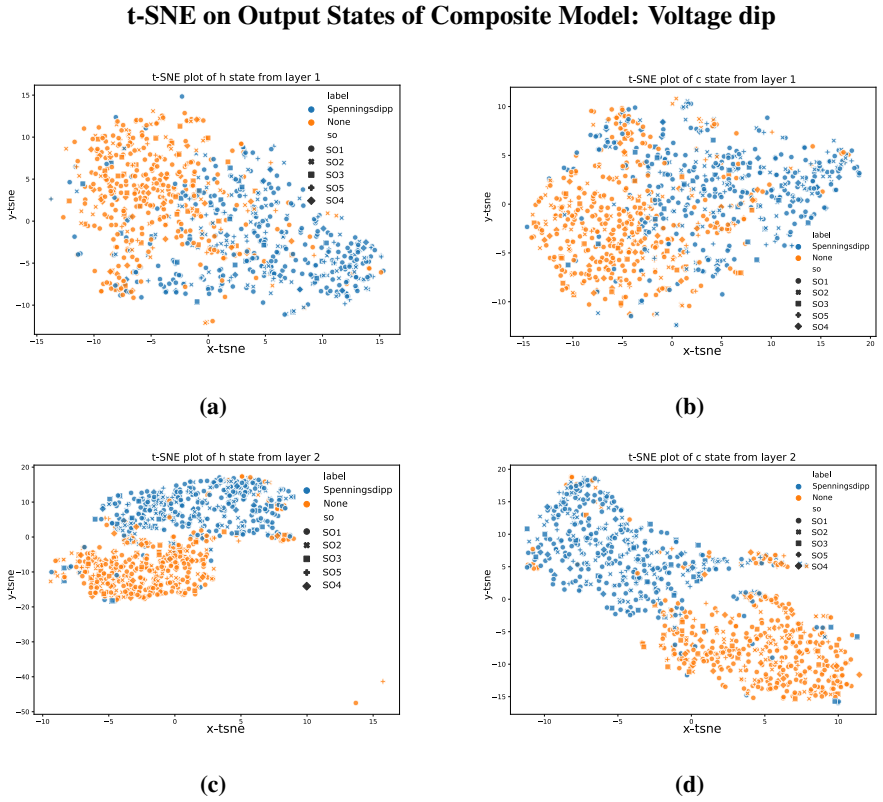


Figure 4.18: Non fault and voltage dip samples was used in this training of the composite model. 'Avbrudd' and 'None' refers to interruption and nominal operation samples, respectively. 'SO' refers to system operator. a): t-SNE representation of the activity state of the last output of the first layer. b): t-SNE representation of the memory cell state of the last output of the first layer. c): t-SNE representation of the activity state of the last output of the second layer. d): t-SNE representation of the memory cell state of the last output of the second layer.It is clear that the composite model has learned to separate the two event classes. However, there are still some mixing of samples, meaning those samples are close in the original dimension. In the case of interruption this may be an indication of a scheduled outage.

Fig. 4.19 show the model states on seen and unseen data. These plots may be interpreted in the direction of what kind of event class the model are less and most likely to learn. As seen in the bottom plots, most non-fault samples are clustered together, while earth faults and interruptions seem to mix. When it comes to the unseen data, no valuable information can be taken away from this analysis.

t-SNE on Output States of Composite model: Unseen and seen data, Multiclass

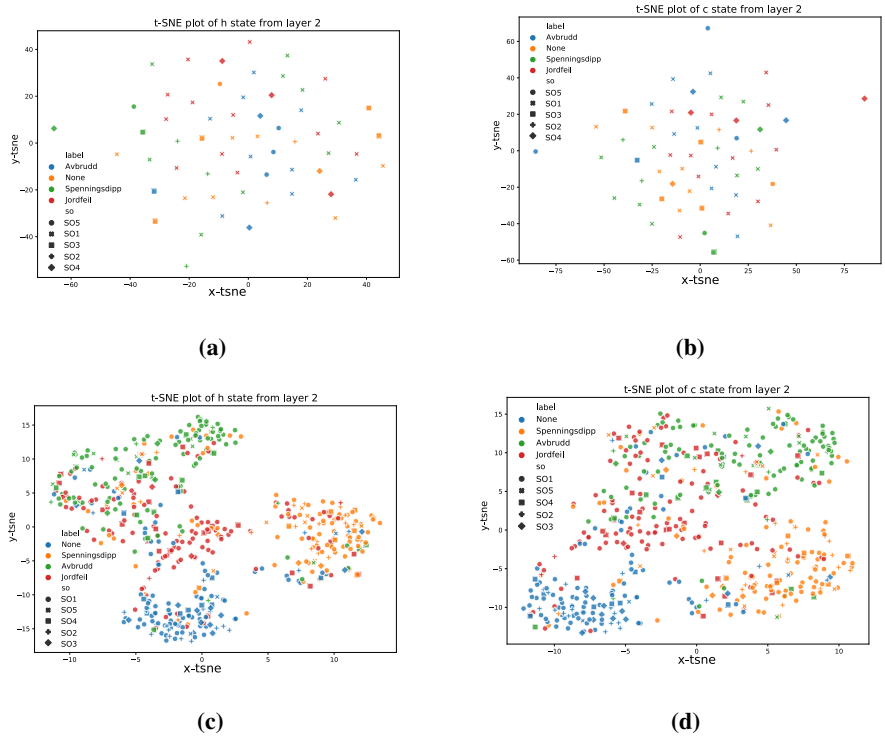


Figure 4.19: a) and b): Activity output and memory cell state of the second layer when evaluated on unseen test data. c) and d): Activity output and memory cell state of the second layer when evaluated on seen train data. 'Avbrudd', 'Spenningsdipp', 'Jordfeil', and 'None' refers to interruption, voltage dip, earth fault, and nominal operation samples, respectively. 'SO' refers to system operator. These plots may be interpreted in the direction of which kind of event class the model are less and most likely to learn. As seen in the bottom plots, most non fault samples are clustered together, while earth faults and interruptions seems to mix. When it comes to the test data, no valuable information is taken away from this analysis.

4.7 General Discussion

As part of a competence building research program this thesis contributes to the foundation of further research in the area outlined. Placing the research in a broader view, the results may lead to increased security of power supply, reduced operation and maintenance (O & M) costs. This will indirectly reduce the impact on the environment by enabling a safer integration of sustainable energy sources.

Currently there are no systems in operation enabling to predict faults in the power grid. Most of the research have been focusing on PMU data. Previous research conducted on PQ data is merely of classifying faults that already have occurred, and is tested on synthetic data [16, 17]. It is believed that PQ data contains more information about the different fault types compared to PMU data. This is because faults in the power system, as investigated in this thesis, contain an unique set of harmonic components, as may be used to distinguish a fault from a nominal operating line. Results from testing the various features confirm the uniqueness of the harmonic components. With certainty, they can be used to predict faults in the power system. PMU may have its strength in monitoring overall system stability. However, PMU was not in the scope of this thesis.

The main objective of predicting a fault in development is to support the system operation. The robustness of the model is crucial in the sense of predicting true positives. False negative prediction might be preferable to false positive prediction, because a false positive can result in unnecessary shutdown of the system. However, a false negative might give the impression of nominal operation, when in fact a fault is about to occur. In these cases, combinations of monitoring systems would be preferable to deploy. If the majority of the monitoring units report anomalies from nominal operation the system operator may use this as a tool to perform corrective maintenance work before escalation of the fault. These systems can be realised through work already being conducted on digitalisation in the energy sector, and easily integrated in a secured network. The benefits of having a system that is able to predict faults in the manner of minutes, or even hours in advance, are many. This can enable more effective use of the power grid by increasing the capacity of power supply due to better monitoring. It can reduce economical expenses by alerting the operator of the power grid about possible faults. Actions can then be taken to isolate the affected equipment, which then can be repaired instead of being replaced. That in turn will reduce ILE expenses.

The supply of energy have to be transparent. In our society, electrical power is considered a matter of course. Too drastic changes may cause discontent among the consumers. However, security of energy supply is also an important aspect that needs to be addressed when developing the electrical power system of the future. By transparency means that the consumer does not receive an extra responsibility in every day life when it comes to when to or not to use electricity.

When talking about the development of the new power system, it is inevitable not to talk about the socioeconomic aspect of it. Are the investments socioeconomically justifiable? A self monitoring system will contribute to reduced operation and maintenance (O&M) costs, and increase the overall health of the system. This will in turn reduce the need for reinvestments and enable smart asset management.

On the other hand, the self monitoring system will guide the energy transfer from renewable energy sources, or any energy production sources in general, to be more efficient.

This will lead to a reduced impact on the environment.

This thesis use techniques from the data sciences to analyse PQ data, and tries to break new ground by applying unconventional methods to approach the problems in question. Harmonic components of the physical parameters are investigated, sources of it are identified, and associated with specific fault types. Since all the harmonic components sums to the original signal, the inclusion of the original signal is redundant if the model is more sensitive to the relationship between the harmonic components. To give a conclusion about this redundancy, tests targeted this problem need to be conducted.

No known research is addressing the composite model proposed in this master's thesis. The closest are RNN architectures trained on labelled targets in conjunction with an SVM, or a regular Autoencoder, not considering the temporal dependency of the time series. As mentioned, the model is inspired by structures form sequential problems, such as video frame prediction and sentiment analysis, trying to extract the hidden pattern of the data points, and represent the sequences of signal vectors into a compressed set variables. It is believed that this approach is able to remove noise and capture the most important feature of the input sequences. To improve the performance of the reconstruction part, and subsequently giving a better representation of the sequence, more thorough testing and modifications of the model must be conducted. One specific improvement may be to use the output of one sequence of the decoder as the input of the next sequence, helping with the reconstruction. If this model seems to work well, it can be applied to any problem or situation where time series data are obtainable, such as flow of fluids, weather, and electrical signals in general. In these instances the model can be used for anomaly detection or system state monitoring. An interesting extension of the model would be to add a one dimensional convolutional neural network (CNN) before the input of the encoder or the other architectures, as tested in [16], with good results. In that study, a hybrid architecture was proposed to be applied on raw PQ signal. The architecture consisted of a multilevel CNN for capturing low level spatial dependencies combined with a LSTM layer, capturing the temporal dependencies and creating a more abstract feature map. The benefit of adding this kind of layer is to extract the most important information from multiple time steps, functioning as a feature extractor. This may help the LSTM layers to learn more abstract patterns.

Testing the composite model, the states at the output sequence were analysed to see what the model had learned. It was particularly interesting to see how well the model performed on the multiclass problem. With this in mind, having a good reconstruction model, it would be interesting to analyse the activity state vectors when the model has only been trained for reconstruction. This to see if feeding various fault events into the model will be well separated when doing dimension reduction such as t-SNA or principal component analysis (PCA).

The voltage in the power system is considered to be fixed. However, the current varies mainly with the loads in the system. This can affect the voltage level in the system if the variation is too drastic. This is analogous to water pressure in a pipe decreasing when opening a valve to an empty chamber. There are direct relationships between the physical parameters. This needs to be explored in further detail.

In this thesis, 148 samples containing interruption, 1388 samples containing earth fault, and 1919 samples containing voltage dip, were used. Of all the tests conducted,

interruptions were the easiest to predict, with earth faults second and voltage dips third, considering the ROC-plots in **Fig. 4.9**. This result concurs with the results from [21]. In the referred study, the spectral information of cycle-by-cycle voltage measurements were used with time series ranging from 40 to 1280 seconds testing prediction horizons from 0 to 40 seconds.

Pre processing the data is an effective way of getting the most information out of the data in hand. In this thesis smoothing and standardising were performed. However, many pre processing techniques in the form of feature extraction are available. Pre processing should be applied to counteract the problem some models have not to generalise well due to the size of the training data. This is a typical problem in deep learning. Usually data sets containing several thousand samples are used to classify events [15, 19, 22].

As stated earlier, there are some samples in the data set which were not classified correctly. This has only been confirmed related to the interruption samples. However, it may apply to the other fault classes as well. The reason for this statement has to do with the difficulty of predicting voltage dips. This class dominates the fault event statistics in form of number of events. Regardless of this, it is the most difficult event to predict compared to the other classes. Similarities between the signatures of the voltage dip and the nominal operation could be an explanation of this difficulty, since a voltage dip can be caused by a heavy load being connected to the grid with no pre warning. An earth fault could come in various forms, it could be single phase to ground earth fault, double phase earth fault, or triple phase earth fault, isolated or to ground. Each fault will act differently on the power system, in form of short circuit currents and system impedance.

In general, there are different causes for a fault to occur, and some of these causes do not have a development that can be detected by the PQ data measurements. Examples of this are weather conditions, environmental factors, such as animals and vegetation, accidental power line ruptures by vehicle encounters or maintenance. The situations mentioned are also among the main causes of interruptions and disturbances in the Norwegian power grid [63]. By combining data from these external factors, a more solid evaluation may be given by the system. The factors not directly imposed on the power grid prior to the fault event, can in this case be used in combination with internal measurements to predict the state of the power system.

A possible approach related to data set preparation, is focusing on few measurement nodes, such as only data from one specific system operator when training the model. The theory of faults in electrical power systems states that some fault events have specific signatures, such as harmonic components occurring prior or during a fault [35]. In this study, only the combination of all harmonics were used as features. It could be interesting to expand the study. Would the symmetrical component sequences of current and voltage result in a more distinct separation between nominal operation and a given fault event?

One approach may be to train one single model for each fault class and combine the probabilities of the predictions to determine the most likely class. This will minimise the complexity the model has to deal with. Due to relatively small size of fault events, conducting deep learning on these problems may be inferior compared to other approaches, such as clustering and other state-of-the-art statistical methods. To address this problem, the sequence-to-sequence Autoencoder may be trained solely on the infinite stream of nominal data, as proposed earlier. When fed non nominal data, the output data and the

states of the model may be analysed for anomalies and used to determine the state of the system.

4.8 Further Work

In this section the central points from the discussion related to further work will be summarised.

Improvements to the composite model is proposed, and its applicability in other areas is suggested. The sequence-to-sequence Autoencoder is highly suitable to be used in problems where data for labelling is difficult to obtain or rarely occurs. This is because the concept of the method is to use self-supervised learning, a version of unsupervised learning, not using labelled data. Fault event prediction may be viewed as an anomaly detection problem, because a fault event deviates from the nominal operation.

A more focused analysis of what the models learn and how well they distinguish between classes, is recommended. This is best done by analysing the weight and state vectors of the model. In this study, only the state vectors were analysed. However, the weights could contain valuable information about the nature of the input data. Basal analysis, such as inputting a simple signal to see if there is a clear pattern correlating the signal and learned weights, is proposed.

An important part of this study has been to investigate suitable features among the physical parameters. It is suggested to do more specific testing of the physical parameters. Because the results from this study show an indication that different physical parameters explains the signature of the fault type. An abstraction to this is to investigate which harmonic component best explains a given fault type.

Further investigation on the prediction horizon is proposed. In this thesis the metrics made an improvement in the positive direction at the seven minute mark. For this it is recommended to use longer time series because it is believed that more information is present in a longer development span.

In general, the data used for prediction has to be pre processed before fed to the model. Various methods to extract statistical information from raw data or spectral representation should be explored in more detail.

The long term goal is to provide a real time monitoring system for prediction of fault events. To include a work in progress model into the system for testing would yield great insight in how the model reacts on continuous data flow. It would also provide valuable information on how the model functions in a practical point of view, and reveal any shortcomings the model might have.

Chapter 5

Conclusion

In this master's thesis, deep learning techniques originally from video and language (NLP) research were tested on time series measurement data from the Norwegian electrical power system to predict the occurrence of a fault event. A sequence-to-sequence Autoencoder was proposed for use in signal feature extraction. Various tests were conducted from investigating the raw data, to analysing the output of the model. Results have shown a prediction horizon up to seven minutes is possible. It was proposed that even longer horizons may be plausible. Further investigation into the harmonic components was proposed, related to signal analysis and statistics, for better feature extraction.

The most promising features were the harmonic components of voltage and current. The fault types may have various composition of harmonic components giving the different fault types an unique signature.

This thesis has been focusing on sequential recurrent neural networks, capturing the temporal dependencies in the time series data. Improvements to the model have been proposed, focusing on anomaly detection.

In combination with other monitoring equipment, a fault event prediction system can be used as a tool in decision making.

As part of a competence building research program this thesis contributes to the foundation of further research on the area outlined. Multiple research topics were proposed as suitable extensions to the research conducted. Placing the research in a broader view, the results may lead to increased security of power supply, reduced operation and maintenance (O & M) costs, and indirectly reducing the impact on the environment by enabling a safer integration of sustainable energy sources.

Bibliography

[1] Met. Det blir varmere. https://www.met.no/vaer-og-klima/hvordan-blir-vaeret-i-framtiden, Date Accessed: 2019-03-27, 2017.

[2] The Paris Agreement — UNFCCC, 2018.

[3] United Nations Framework Convention on Climate Change. Paris Agreement - Status of Ratification. UNFCCC. In *United Nations Framework Convention on Climate Change*, 2018.

[4] ENTSO-E. Who Is ENTSO-E? https://www.entsoe.eu/about/inside-entsoe/objectives/, Date accessed: 2019-03-28, 2016.

[5] IRENA. ELECTRIFICATION WITH RENEWABLES Driving the transformation of energy services. Technical report, International Renewable Energy Agency, Abu Dhabi, 2019.

[6] IRENA. Innovation landscape for a renewable-powered furure: Solutions to integrate variable renewables. Technical report, International Renewable Energy Agency, Abu Dhabi, 2019.

[7] CINELDI. Annual Report 2018 CINeLDI. Technical report, Trondheim, 2018.

[8] J Duncan Glover, Thomas J Overbye, and Mulukutla S Sarma. *Power System Analysis & Design*. Cengage Learning, Boston, 6 edition, 2017.

[9] Energi21. Strategi 2018. Technical report, Energi21, Oslo, 2018.

[10] Serge Colle, Paul Micallef, Anthony Legg, and Andrew Horstead. Where does change start if the future is already decided ? Technical report, EY, 2019.

[11] Energi Norge. 1 , 5 ° C – Hvordan Norge kan gjøre sin del av jobben. Technical report, 2019.

[12] IEA Publications. Digitization & Energy. Technical report, 2017.

[13] European Network. Where the digital transformation of the european electricity system starts:. 2017.

[14] Nikos Hatziargyriou. National and Regional Smart Grids initiatives in Europe. *European Technology and Innovation Platform (ETIP)*, 2(Second Edition):52, 2016.

[15] Ebrahim Balouji, Irene Y.H. Gu, Math H.J. Bollen, Azam Bagheri, and Mahmood Nazari. A LSTM-based deep learning method with application to voltage dip classification. *Proceedings of International Conference on Harmonics and Quality of Power, ICHQP*, 2018-May:1–5, 2018.

[16] Neethu Mohan, K. P. Soman, and R. Vinayakumar. Deep power: Deep learning architectures for power quality disturbances classification. *Proceedings of 2017 IEEE International Conference on Technological Advancements in Power and Energy: Exploring Energy Solutions for an Intelligent Power Grid, TAP Energy 2017*, pages 1–6, 2018.

[17] Haosen Yang, Robert C. Qiu, Xin Shi, and Xing He. Deep Learning Architecture for Voltage Stability Evaluation in Smart Grid based on Variational Autoencoders. pages 1–9, 2018.

[18] Jian Ma, Jun Zhang, Luxin Xiao, Kexu Chen, and Jianhua Wu. Classification of Power Quality Disturbances via Deep Learning. *IETE Technical Review (Institution of Electronics and Telecommunication Engineers, India)*, 34(4):408–415, 2017.

[19] Senlin Zhang, Yixing Wang, Meiqin Liu, and Zhejing Bao. Data-Based Line Trip Fault Prediction in Power Systems Using LSTM Networks and SVM. *IEEE Access*, 6:7675–7686, 2017.

[20] Christian Andre Andresen, Bendik Nybakk Torsaeter, Hallvar Haugdal, and Kjetil Uhlen. Fault Detection and Prediction in Smart Grids. In *9th IEEE International Workshop on Applied Measurements for Power Systems, AMPS 2018 - Proceedings*, 2018.

[21] Volker Hoffmann, Katarzyna Michalowsa, Christian Andre Andresen, and Bendik Nybakk Torsaeter. Incipient Fault Prediction in Power Quality Monitoring. In *25 th International Conference on Electricity Distribution APPLICATION OF DYNAMIC TRANSFORMER RATINGS TO INCREASE THE 25 th International Conference on Electricity Distribution Madrid , 3-6 June 2019*, number June, pages 3–6, 2019.

[22] Tim de Bruin, Kim Verbert, and Robert Babuska. Railway Track Circuit Fault Diagnosis Using Recurrent Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 28(3):523 – 533, 2017.

[23] Hinton G.E and Salakhutdinov R.R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(July):504–507, 2006.

[24] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE Laurens. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[25] Jaehyun Park. Unsupervised Learning of Invariances in Deep Networks. 2016.

[26] Aaron Courville Ian Goodfellow, Yoshua Bengio. The Deep Learning Book. *MIT Press*, 521(7553):785, 2017.

[27] Francois Chollet. *Deep Learning With Python*. Manning, New York, 2018.

[28] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning*. Packt Publishing, Birmingham, 2 edition, 2017.

[29] Andrew Ng. *Machine Learning Yearning: Technical Strategy for AI Engineers, In the Era of Deep Learning*. deeplearning.ai, 2018.

[30] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer, New Yrok, 2013.

[31] Ned Mohan, Tore M. Undeland, and William P. Robbins. *Power Electronics - Converters, Applications, and Design*. Wiley, 3 edition, 2003.

[32] Theodore Wildi. *Electrical Machines Drive and Power Systems*. Pearson Education, New Jersey, 6 edition, 2006.

[33] D. P. Kothari and I. J. Nagrath. *Modern Power System Analysis*. Tata McGraw Hill, New Delhi, 3 edition, 2009.

[34] C L Wadhwa. *Electrical Power Systems*. New Academic Science Limited, Kent, UK, 2012.

[35] C. Sanaran. *Power quality*. CRC Press, New York, 2002.

[36] Surajit Chattopadhyay, Madhuchhanda Mitra, and Samarjit Sengupta. *Electric Power Quality*. New Yrok, 1 edition, 2011.

[37] NVE. Nasjonal varedeklarasjon 2017. https://www.nve.no/reguleringsmyndigheten-for-energi-rme-marked-og-monopol/varedeklarasjon/nasjonal-varedeklarasjon-2017/, Date Accessed: 2019-05-01, 2017.

[38] NVE. Om kraftmarkedet og det norske kraftsystemet - NVE. https://www.nve.no/stromkunde/om-kraftmarkedet-og-det-norske-kraftsystemet/, Date Accessed: 2019-05-08, 2018.

[39] Paul A. Tipler and Gene Mosca. *Physics for scientists and engineers*. W. H. Freeman, New York, 6 edition, 2008.

[40] Power quality logger & Power quality solutions - Elspec. https://www.elspec-ltd.com/, Date Accessed: 2019-04-30.

[41] Leveringskvalitetsforskriften. (FOR-2004-11-30-1557) Forskrift om leveringskvalitet i kraftsystemet, 2004.

[42] Metoder og bruksområder. Technical report, EFI Sintefgruppen, Tech. Rep.

[43] Morten Særen and Martin Giset. Systemjording. Technical Report august, 2017.

[44] Kashyap.valiveti. Unbalanced symmetrical components. https://en.wikipedia.org/wiki/File:Unbalanced_symmetrical_components.pdf. Public domain, 2014.

[45] Tom Mitchell. *Machine Learning*. 1997.

[46] Darecophoenixx. GaussianKernel layer keras extension. https://github.com/darecophoenixx/wordroid.sblo.jp/tree/master/lib/keras_ex/gkernel, Date Accessed: 2019-04-23.

[47] Sepp Hochreiter and Jj Urgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[48] Shi Yan. Understanding LSTM and its diagrams – ML Review – Medium. https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714, Date Accessed: 2019-04-26, 2016.

[49] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. 9 2014.

[50] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. pages 1–9, 2014.

[51] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised Learning of Video Representations using LSTMs. 2 2015.

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. pages 1–9, 2014.

[53] Sharpr. Reciever Operating Characteristics [Image]. https://commons.wikimedia.org/wiki/File:ROC_curves.svg. CC BY-SA 3.0 - https://commons.wikimedia.org/w/index.php?curid=44059691".

[54] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 12 2014.

[55] Henrik Kirkeby. AUTOMATISK HENDELSESANALYSE. Technical report, Informasjonsteknologi og elektroteknikk - det digitale energiskiftet: NEF teknisk møte, Trondheim, 2017.

[56] Plotly. Python Graphing Library, Plotly. https://plot.ly/python/, Date Accessed: 2019-04-18.

[57] Home - Keras Documentation. https://keras.io/, Date Accessed: 2019-04-18, 2018.

[58] TensorFlow. https://www.tensorflow.org/, Date Accessed: 2019-04-18.

[59] François Chollet. Lstm seq2seq - Keras Documentation. https://keras.io/examples/lstm_seq2seq/, Date Accessed: 2019-04-23.

[60] Yumei Qi, Changqing Shen, Dong Wang, Juanjuan Shi, Xingxing Jiang, and Zhongkui Zhu. Stacked Sparse Autoencoder-Based Deep Network for Fault Diagnosis of Rotating Machinery. *IEEE Access*, 5:15066–15079, 2017.

[61] Laurens van der Maaten. Barnes-Hut-SNE. pages 1–11, 2013.

[62] scikit-learn: machine learning in Python — scikit-learn 0.20.3 documentation. https://scikit-learn.org/stable/, Date Accessed: 2019-04-29.

[63] Årsstatistikk 2017. Technical report, Statnett SF, 2017.

Appendix A: Supplementary Results

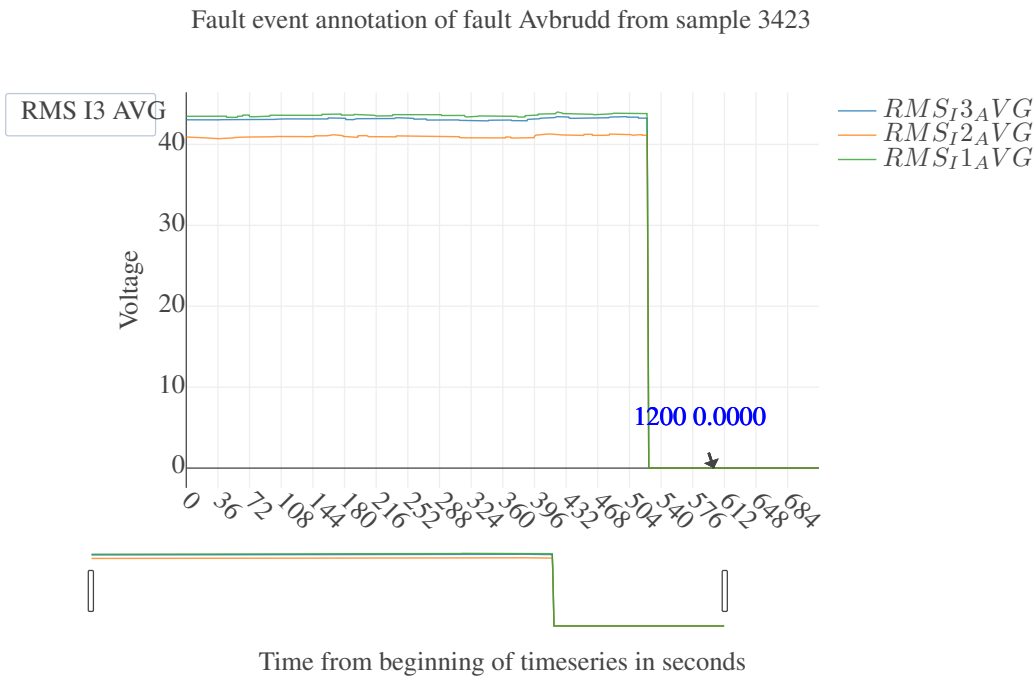


Figure 5.1: Sample from one phase phase current development before an interruption.

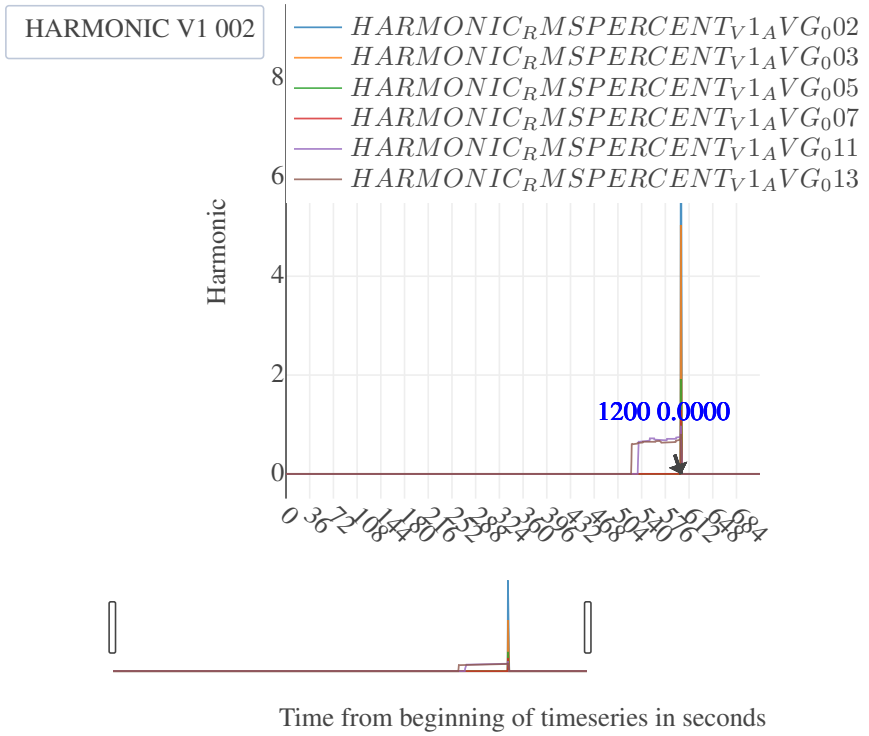


Figure 5.2: Development of harmonic phase voltage before an interruption.

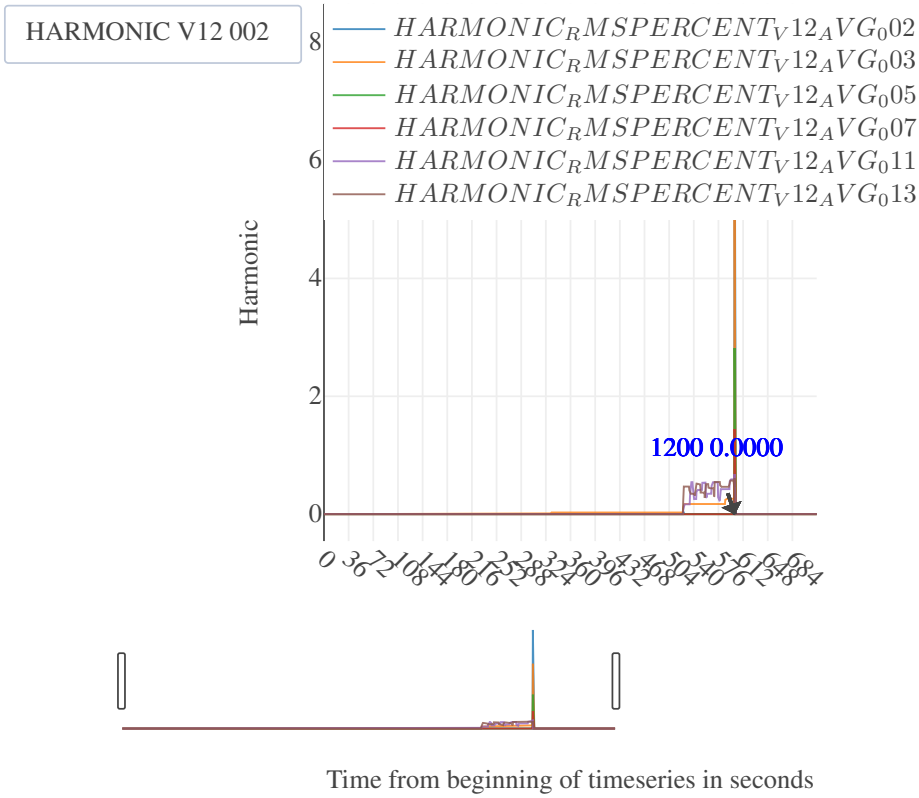


Figure 5.3: Development of harmonic line voltage before an interruption.

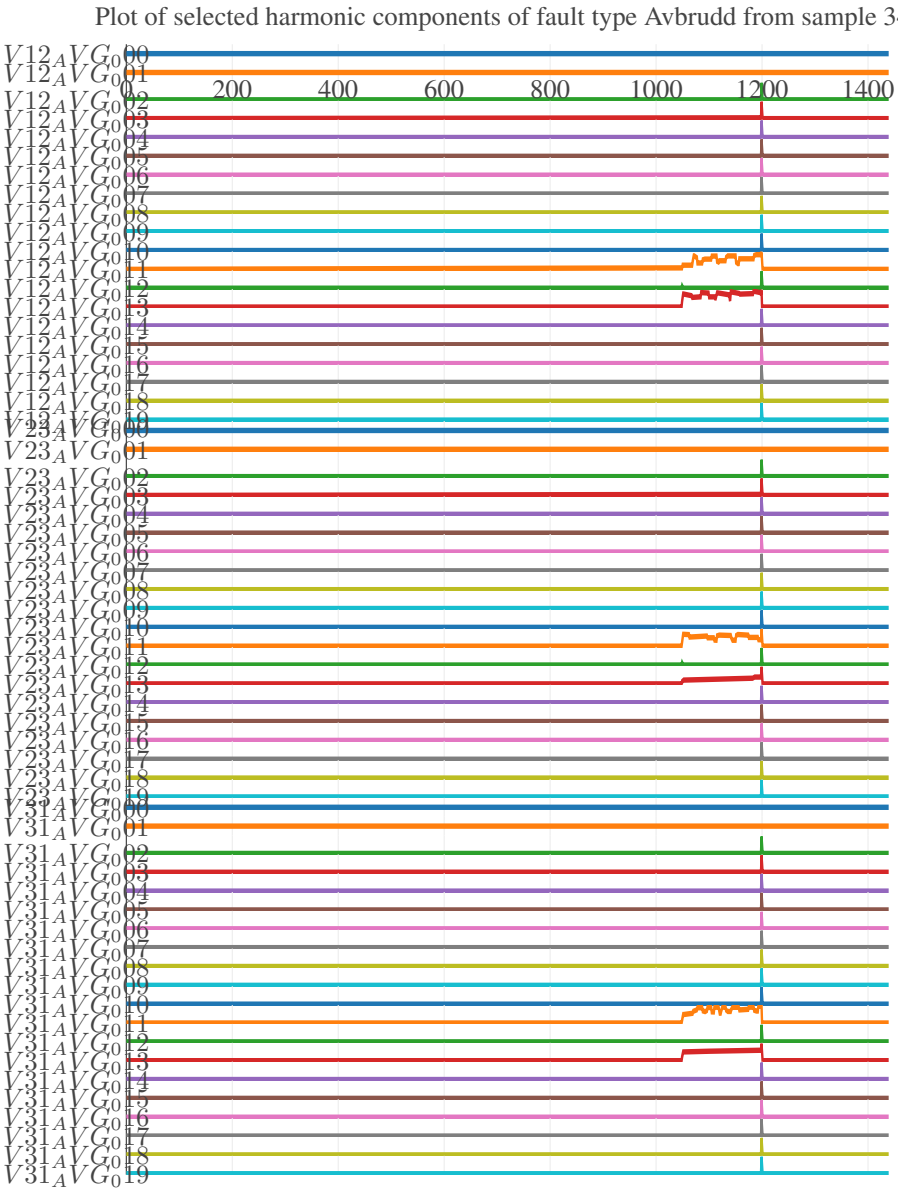


Figure 5.4: Plot of individual harmonic components of three-phase line voltages before an interruption. The harmonics are mostly zero until approximately 330 seconds before the interruption occur. In the inspection tool the 3rd harmonic is present. Note that the 0th and 1st components are zero and should not be plotted.

No pre-train versus pre-train on surge samples

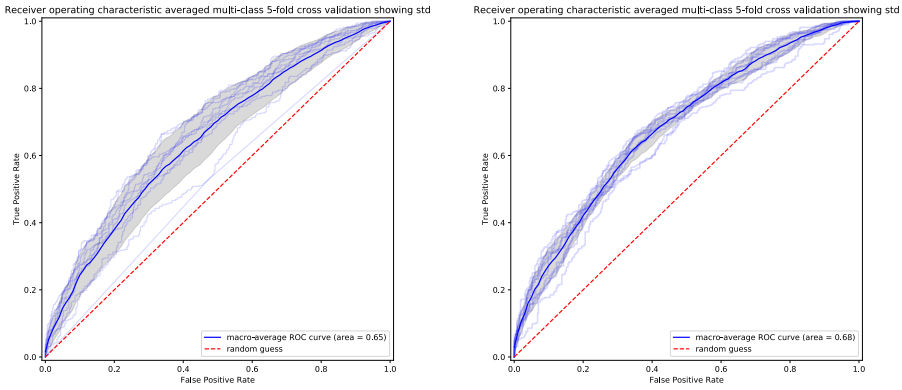


Figure 5.5: Left: ROC plot obtained by training the model with 5-fold cross validation and no pre-training. Right: ROC plot from a pre-trained model with 5-fold cross validation. The model was trained on . The opaque blue lines are the ROC curves for each class, the solid blue is the macro average of all the curves, and the dashed line corresponds to random guess. The gray area represents the standard deviation of the calculation of the average. The area under the curve (AUC) is also computed and displayed in the plot. Trained on balanced classes of non-fault and voltage swell

No pre-train versus pre-train on earth fault samples

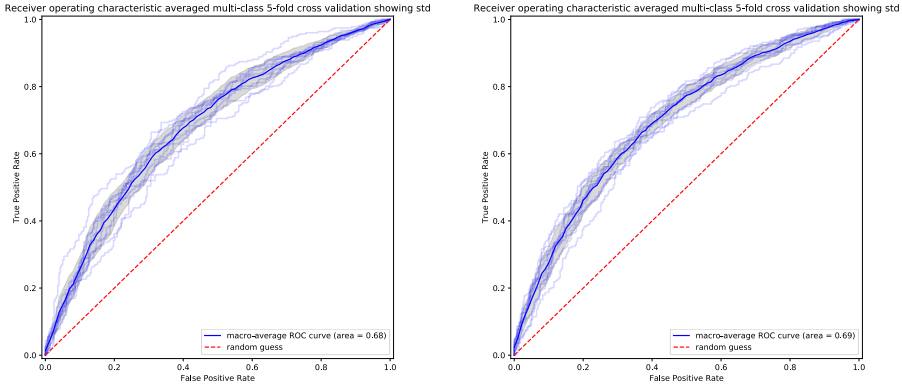


Figure 5.6: Left: ROC plot obtained by training the model with 5-fold cross validation and no pre-training. Right: ROC plot from a pre-trained model with 5-fold cross validation. The model was trained on . The opaque blue lines are the ROC curves for each class, the solid blue is the macro average of all the curves, and the dashed line corresponds to random guess. The gray area represents the standard deviation of the calculation of the average. The area under the curve (AUC) is also computed and displayed in the plot. Trained on balanced classes of non-fault and earth fault

Surge

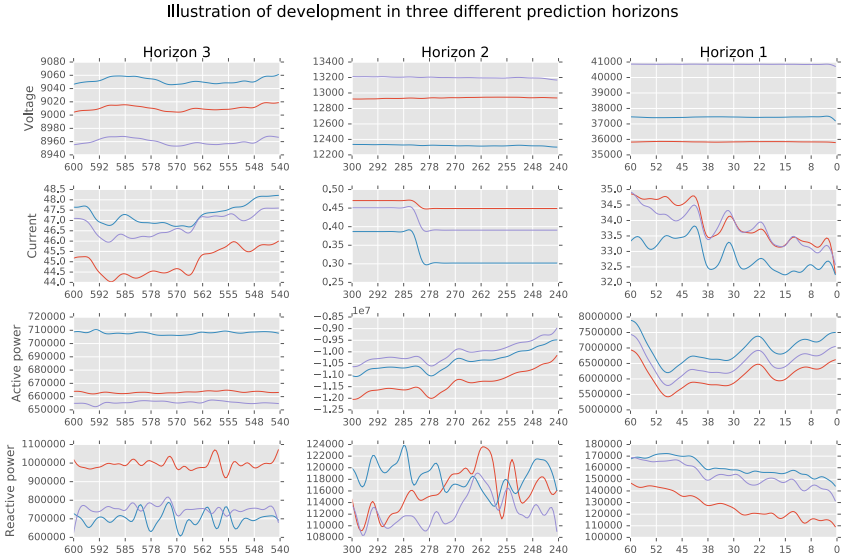


Figure 5.7: Illustration of the development of a voltage dipp sample, in the same spacing as some of the prediction horizons. The plot displays the voltage, current, active power and reactive power of the same sample in descending order. The x-axis states the time from the fault event in seconds.

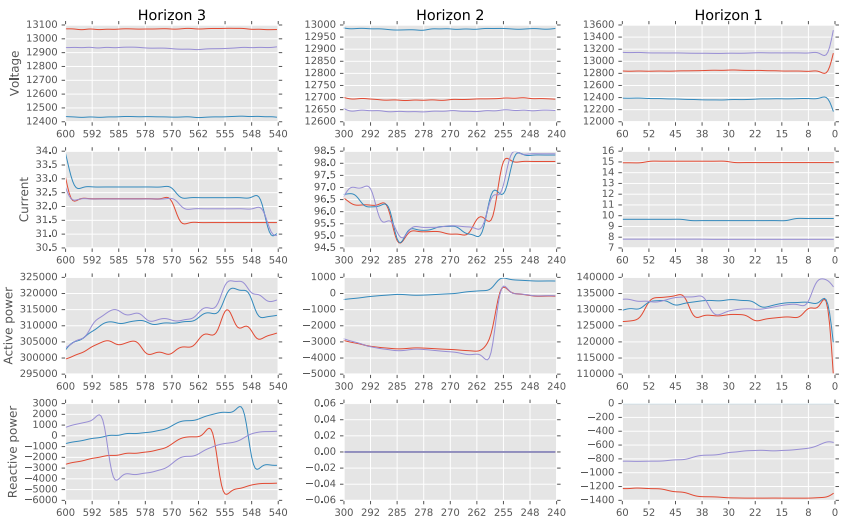


Figure 5.8: Illustration of the development of a earth fault sample, in the same spacing as some of the prediction horizons. The plot displays the voltage, current, active power and reactive power of the same sample in descending order. The x-axis states the time from the fault event in seconds.

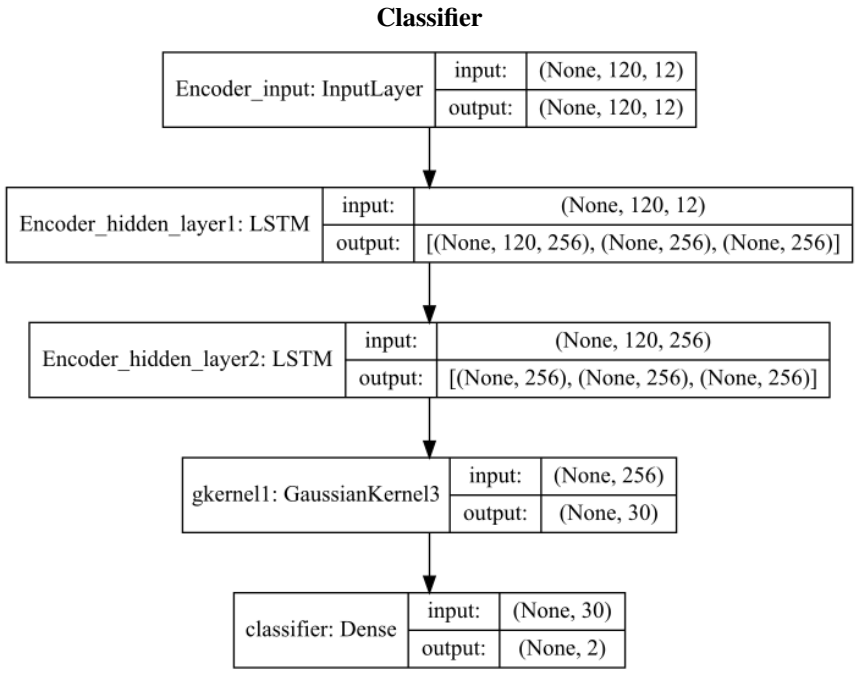


Figure 5.9: Structural architecture of the classifier part of the model.

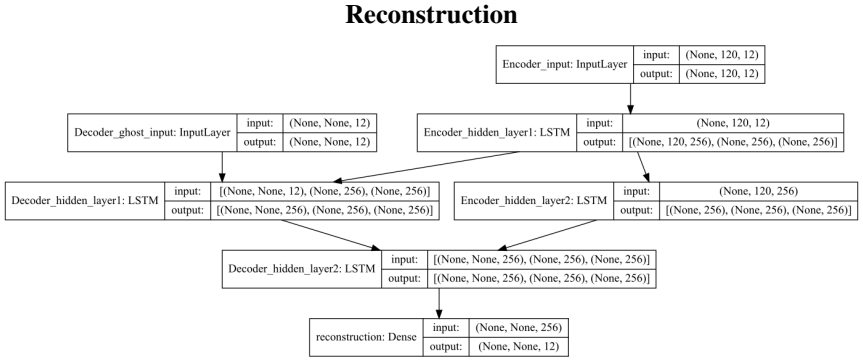


Figure 5.10: Structural architecture of the reconstruction part of the model. This is a variation of a shifted many-to-many LSTM architecture. In the scope of this research the input layer of the decoder functions as ghost layer to make the Keras API happy by feeding the input zero vectors with the same shape as the input of the encoder layer.

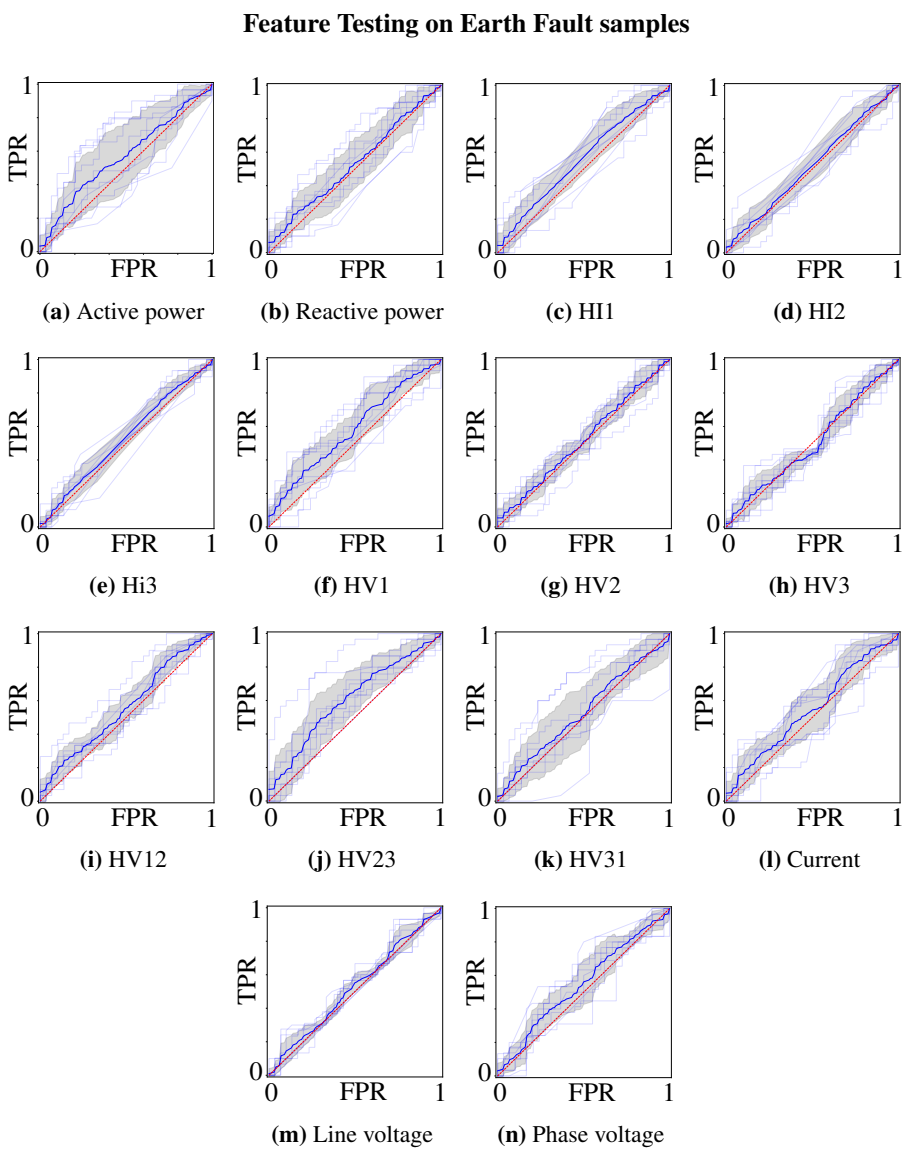
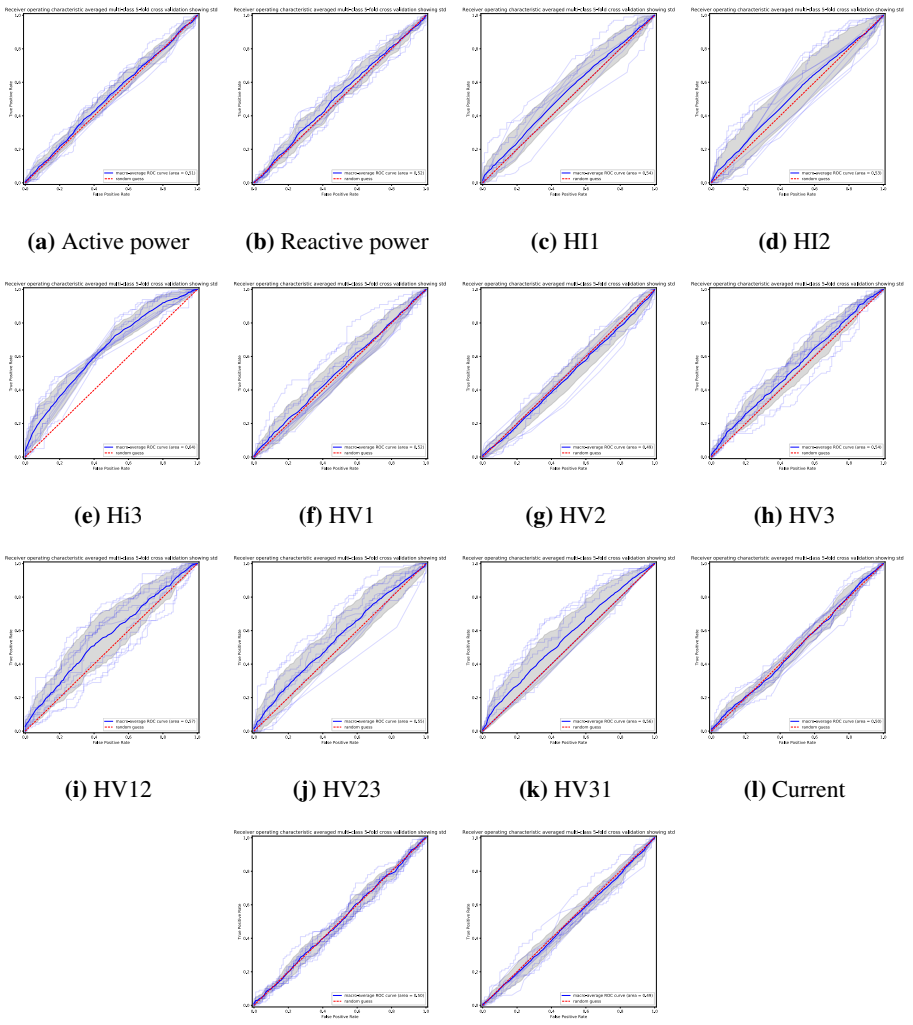


Figure 5.11: ROC-plots over the model trained on the corresponding feature a-n. The training set contained non-faults and earth fault. See **Table. 5.1** for metric results for each feature.

Table 5.1: Results from training on individual features classifying non-faults and earth fault. Each fold was trained for 10 epochs, no pre-training of the model, with batch size of 128 and train/test samples of 800/200. The harmonic features contains six harmonic components; 2nd, 3rd, 5th, 7th, 9th, 11th and 13th, the rest consists of three columns, one for each phase. See **Fig. 5.12** for ROC-plots of each feature.

	Earth Fault		
Features	**Classification Accuracy**	**Classification MCC**	**Reconstruction Loss**
Active power	50.00% (+/- 0.00%)	0.00 (+/- 0.00)	0.93 (+/- 0.04)
Reactive power	50.00% (+/- 0.00%)	0.00 (+/- 0.00)	0.96 (+/- 0.01)
hi1	52.70% (+/- 3.46%)	0.05 (+/- 0.07)	0.23 (+/- 0.02)
hi2	51.10% (+/- 1.02%)	0.02 (+/- 0.02)	0.23 (+/- 0.02)
hi3	53.60% (+/- 4.41%)	0.07 (+/- 0.09)	0.23 (+/- 0.03)
hv1	51.50% (+/- 3.00%)	0.03 (+/- 0.06)	0.53 (+/- 0.02)
hv2	54.00% (+/- 3.33%)	0.08 (+/- 0.07)	0.52 (+/- 0.01)
hv3	53.10% (+/- 4.79%)	0.06 (+/- 0.10)	0.49 (+/- 0.02)
hv12	58.50% (+/- 5.31%)	0.17 (+/- 0.11)	0.58 (+/- 0.01)
hv23	54.40% (+/- 4.14%)	0.09 (+/- 0.08)	0.57 (+/- 0.01)
hv31	55.30% (+/- 4.95%)	0.11 (+/- 0.10)	0.58 (+/- 0.02)
Current	50.70% (+/- 1.40%)	0.01 (+/- 0.03)	0.87 (+/- 0.03)
Line voltage	50.60% (+/- 1.46%)	0.01 (+/- 0.03)	0.96 (+/- 0.02)
Phase voltage	50.30% (+/- 0.40%)	0.01 (+/- 0.01)	0.93 (+/- 0.02)



(a) Active power **(b)** Reactive power **(c)** HI1 **(d)** HI2

(e) Hi3 **(f)** HV1 **(g)** HV2 **(h)** HV3

(i) HV12 **(j)** HV23 **(k)** HV31 **(l)** Current

(m) Line voltage **(n)** Phase voltage

Figure 5.12: ROC-plots over the model trained on the corresponding feature a-n. The training set contained non-faults and voltage dip. See **Table. 5.2** for metric results for each feature.

Table 5.2: Results form training on individual features classifying non-faults and voltage dips. Each fold was trained for 10 epochs, no pre-training of the model, with batch size of 64 and train/test samples of 800/200. The harmonic features contains six harmonic components; 2nd, 3rd, 5th, 7th, 9th, 11th and 13th, the rest consists of three columns, one for each phase. See **Fig. 5.12** for ROC-plots of each feature.

		voltage dip	
Features	**Classification accuracy**	**Classification mcc**	**Reconstruction loss**
Active power	50.00% (+/- 0.00%)	0.00 (+/- 0.00)	0.92 (+/- 0.03)
Reactive power	50.10% (+/- 0.20%)	0.00 (+/- 0.00)	0.95 (+/- 0.01)
hi1	53.00% (+/- 3.70%)	0.06 (+/- 0.07)	0.28 (+/- 0.01)
hi2	55.50% (+/- 4.54%)	0.11 (+/- 0.09)	0.27 (+/- 0.02)
hi3	53.00% (+/- 4.09%)	0.06 (+/- 0.08)	0.30 (+/- 0.03)
hv1	51.80% (+/- 3.36%)	0.04 (+/- 0.07)	0.50 (+/- 0.01)
hv2	50.20% (+/- 0.40%)	0.00 (+/- 0.01)	0.50 (+/- 0.02)
hv3	53.80% (+/- 3.39%)	0.08 (+/- 0.07)	0.47 (+/- 0.01)
hv12	55.60% (+/- 5.01%)	0.11 (+/- 0.10)	0.55 (+/- 0.02)
hv23	53.40% (+/- 3.89%)	0.07 (+/- 0.08)	0.54 (+/- 0.01)
hv31	53.60% (+/- 5.77%)	0.07 (+/- 0.12)	0.55 (+/- 0.01)
Current	50.20% (+/- 0.40%)	0.00 (+/- 0.01)	0.87 (+/- 0.01)
Line voltage	50.00% (+/- 0.63%)	0.00 (+/- 0.01)	0.96 (+/- 0.02)
Phase voltage	50.00% (+/- 0.00%)	0.00 (+/- 0.00)	0.93 (+/- 0.03)

Appendix B: Code

```
1
2  ## Versions used ##
3  # Python: 3.6.8
4  # Keras: 2.2.4
5  # TensorFlow: 1.13.1
6  # Numpy: 1.16.3
7  # Pandas: 0.23.4
8  # Plotly: 3.5.0
9  # Sklearn: 0.20.3
10 ##################
11
12 import itertools
13 import tables
14 import os
15 import time
16 import plotly
17 import pandas as pd
18 import numpy as np
19 import tensorflow as tf
20 import seaborn as sns
21 import matplotlib.pyplot as plt
22 import keras.backend as K
23 import plotly.plotly as py
24 import plotly.graph_objs as go
25
26 from keras.models import Model
27 from keras.layers import Input, LSTM, Dense, Activation
28 from keras.utils import plot_model
29 from keras.regularizers import l2
30 from keras.engine.topology import Layer
31 from keras import initializers, constraints
32 from keras import losses
33 from tensorflow.python.keras.callbacks import TensorBoard
34 from numpy.random import seed
35 from sklearn.manifold import TSNE
36 from sklearn.model_selection import StratifiedKFold, train_test_split
37 from sklearn.metrics import roc_curve, auc, f1_score, matthews_corrcoef,
       confusion_matrix, accuracy_score
38 from sklearn.preprocessing import StandardScaler
39 from sklearn import preprocessing
40 from scipy import interp, signal
41 from datetime import datetime
42 from matplotlib.colors import ListedColormap
43 from IPython.core.interactiveshell import InteractiveShell
44 from plotly.offline import init_notebook_mode, iplot
45
46 # For displaying plotly in jupyter notebook
```

```
47 InteractiveShell.ast_node_interactivity = 'all'
48 plt.style.use('bmh')
49 init_notebook_mode(connected=True)
```

Code 5.1: Imports

```
###                Plotly functions                    ###
def random_sample(y, event_type='Avbrudd'):
    # random sample selector of specific event type
    # Jordfeil, None, Spenningsdipp, Avbrudd
    sample = np.random.choice(np.argwhere(y==event_type).reshape(1,),size
    =1)[0]
    return sample

def find_fault_event(metadata, x):
    resolution = float(metadata['resolution_ms'])
    idx_fault_event = int(int(metadata['N_points'])   int(metadata['
    Time_buffer_sec'])*1000/resolution)
    result = pd.concat([pd.Series(x.iloc[idx_fault_event]), pd.Series(
    idx_fault_event)], axis = 1)
    result.columns = ['value', 'time']
    return result.set_index('time')

# Building tyhe annotaion
def build_annotations(fault):
    annotation = [dict(x = time, y = value[0],
                       xref = 'x', yref = 'y',
                       font=dict(color = 'blue'),
                       text = f'{time} <br> {value[0]:.4f}')
                  for time, value in zip(fault.index, fault.values)]
    return annotation[0]

def create_plot_data(selected_sample, metadata, feature_name, feature,
    annotations=True, color=None):
    try:
        plot_series = selected_sample.iloc[:,feature]
        names = feature_name[feature].split('_')
    except:
        plot_series = selected_sample.loc[:,feature]
        names = feature_name[feature].split('_')

    if names[0].lower() == 'harmonic':
        yaxis = 'y2'
    else:
        yaxis = 'y'

    plot_data = go.Scatter(
        x=plot_series.index,
        y=plot_series.values.reshape(1,),
        line=dict(color=color, width=1.1),
        opacity=0.8,
        name=feature_name[feature],
        yaxis=yaxis,
        hoverinfo = 'text+x',
        text = [f'{names[0]} {names[2]} {names[1]}: {x:.4f}' for x in
    plot_series.values.reshape(1,)]
        )

    if annotations:
        fault_event = find_fault_event(metadata, plot_series)
        fault_event_annotation = build_annotations(fault_event)
        return plot_data, fault_event_annotation

```

```
    return plot_data


def plot_sample_data(selected_sample, metadata, feature_name, feature,
    fault_type, annotations_flag=True, save=False):

    names = [y[0]+' '+y[1]+' '+y[2] for y in [[x.split('_')[i]if len(x.
    split('_'))>4 else x.split('_')[j] for j,i in enumerate([0,2,4]) ] for
     x in feature_name[feature]]]

    data_annotations = [create_plot_data(selected_sample, metadata,
    feature_name, feature=x, annotations=annotations_flag) for x in
    feature]



    if annotations_flag:
        data = [x[0] for x in data_annotations]
        annotations = [x[1] for x in data_annotations]
    else:
        data = data_annotations
        annotations = []

    num_features = len(data)

    visible = np.append(list(np.diag(np.ones(num_features))==1),list(np.
    ones(num_features)==1)).reshape(num_features+1,num_features)
    visible = [list(i) for i in visible]

    if annotations_flag:
        button_list = [dict(label=names[i],
                            method='update',
                            args=[{'visible':list(visible[i])
                                  },{
                                    'title': f'Fault event annotation of
    fault {fault_type} on {names[i]} from sample {sample+1}',
                                    'annotations': [annotations[i]]
                            }
                                 ]) for i in range(num_features)]

        button_list.append(dict(label='All', method='update', args=[{'
    visible':list(visible[ 1])
                                                            },{
                                                              'title': f'
    Fault event annotation of fault {fault_type} from sample {sample+1}',
                                                              'annotations':
     annotations}

                                                                    ]))
    else:
        button_list = [dict(label=names[i],
                            method='update',
                            args=[{'visible':list(visible[i])
                                  },{
                                    'title': f'Fault event of fault {
    fault_type} on {names[i]} from sample {sample+1}',
                            }
```

```
                                ]) for i in range(num_features)]

    button_list.append(dict(label='All', method='update', args=[{'visible':list(visible[:1])
                                                                 },{
                                                                  'title': f'Fault event of fault {fault_type} from sample {sample+1}'
                                                                  }
                                                                  ]))


updatemenus = list([
    dict(
    active=0,
    buttons=button_list)
    ])

def zoom(layout, xrange):
    in_view = df.loc[fig.layout.xaxis.range[0]:fig.layout.xaxis.range[1]]
    fig.layout.yaxis.range = [in_view.High.min()   10, in_view.High.max() + 10]
    print(in_view)

# decide correct axis to plot on
yaxis=dict()
yaxis2=None
for x in data:
    if not yaxis and x.yaxis == 'y':
        yaxis = dict(title='Voltage')
    if not yaxis2 and x.yaxis == 'y2':
        yaxis2= dict(title='Harmonic', color='blue',
                                       overlaying='y',
                                       side='right')
    if yaxis and yaxis2:
        break
if not yaxis:
    yaxis2['side']='left'
    for x in data:
        x.yaxis = 'y'
        yaxis['title']='Harmonic'
        yaxis2 = dict()

    #yaxis2['overlaying']=None

resolution_plot_tic = 20

layout = go.Layout(height=800, width=1000,
                   title=f'Fault event annotation of fault {fault_type} from sample {sample+1}',
                   annotations=annotations,
                   updatemenus=updatemenus,
                   xaxis=dict(title='Time from beginning of timeseries in seconds',
                              # Sliding for selecting time window
                              rangeslider=dict(visible=True),
                              tickvals=list(np.around(np.arange(
```

```
                                    0,
                                    int(metadata['N_points'])+1,
                                    int(metadata['N_points'])/
    resolution_plot_tic),decimals=1)),
                                ticktext=list(np.around(np.arange(
                                    0,
                                    int(metadata['total_duration_sec'])
    +1,
                                    int(metadata['total_duration_sec'])/
    resolution_plot_tic),decimals=1)),
                                tickangle=45,
                                tickformat=',d'
                                # Type of xaxis
                                #type='date'
                                        ),
                        # yaxis is unchanged
                        yaxis=yaxis,
                        yaxis2=yaxis2
                       )


    if save:
        img_name = 'my plot'
        dload = os.path.expanduser('~/masteroppgave/')
        save_dir = '~/figurer'

        iplot(fig)
        plotly.offline.plot(fig, image_filename=img_name, image='svg')

        ### might need to wait for plot to download before copying
        sleep(1)

        #copyfile('{}/{}.svg'.format(dload, img_name),
         #        '{}/{}.svg'.format(save_dir, img_name))
    else:
        iplot(fig)


def plot_train(X_train, y_train, balanced_index, event_type='Avbrudd',
    sample=None):

    i, j = np.where(y_train)
    if not sample:
        sample=random_sample(pd.Series(y_train.columns[j], i),event_type)
        sample = y_train.index.values[sample]

    sub_index = np.argwhere(np.array(balanced_index) == sample)[0][0] #
    get the right sample for information
    pd.DataFrame(X_train[sub_index,::1,:]).plot()
    print(pd.Series(y_train.columns[j], i)[sub_index])

def plottt(selected_sample, harmonic_feature, fault_type, sample,
    feature_name):
    '''Plot selected harmonic features on separate y axis with shared x
    axis.
    '''
    harmonics_df = selected_sample.iloc[:, harmonic_feature].fillna(method
```

```
    ='backfill')
    h_names = np.array([x.split('NT_')[1] for x in feature_name[
    harmonic_feature]])#feature_name[:31]])
    n_harmonics = harmonics_df.shape[1]
    times = harmonics_df.index.values

    step = 1. / n_harmonics
    kwargs = dict(domain=[1    step, 1], showticklabels=False, zeroline=
    False, showgrid=False)

    # create objects for layout and traces
    layout = go.Layout(yaxis=go.layout.YAxis(kwargs),
                       title=f'Plot of selected harmonic components of
    fault type {fault_type} from sample {sample}',
                       showlegend=False);
    traces = [go.Scatter(x=times,
                         y=harmonics_df.values[:, 0],
                         #hoverinfo='text',
                         #text=[f'{names[0][0]} {names[0][2]} {names
    [0][1]}: {harmonics_df.values[:, ii]:.4f}' for x in plot_series.
    values.reshape(1,)]
                         )]

    # loop over the channels
    for ii in range(1, n_harmonics):
            kwargs.update(domain=[1    (ii + 1) * step, 1    ii * step]);
            layout.update({'yaxis%d' % (ii + 1): go.layout.YAxis(kwargs),
    'showlegend': False});
            traces.append(go.Scatter(x=times, y=harmonics_df.values[:, ii
    ],
                                     #hoverinfo='text',
                                     #text=[f'{names[ii][0]} {names[ii
    ][2]} {names[ii][1]}: {harmonics_df.values[:, ii]:.4f}' .reshape(1,)
    ],
                                    yaxis='y%d' % (ii + 1)));

    # add channel names using Annotations
    annotations = go.Annotations([go.layout.Annotation(x=0.1, y=0, xref='
    paper', yref='y%d' % (ii + 1),
                                               text=h_name, font=go.layout.
    annotation.Font(size=8), showarrow=False)
                                  for ii, h_name in enumerate(h_names)])
    layout.update(annotations=annotations);

    size_scale = n_harmonics/30
    # set the size of the figure and plot it
    layout.update(autosize=True, width=800, height=size_scale*800);
    fig = go.Figure(data=go.Data(traces), layout=layout)
    iplot(fig)

    img_name = 'fancy plot'
    # to save image, open the generated 'temp plot.html'... it works.. ;\
    plot(fig, image_filename=img_name, image='svg')

if __name__ == '__main__':
    ### example use ###

```

```
245     sample = 3422
246     # Plot training sample
247     plot_train(prep.X_train, prep.y_train, prep.balanced_index, sample=
        sample)
248     #sample = random_sample(y, 'None')
249     fault_type = prep.y[sample]
250     selected_sample = pd.DataFrame(prep.X_init[sample])
251
252     #plot full sample
253     plot_sample_data(selected_sample, prep.metadata, prep.feature_name,
        prep.feature, fault_type, annotations_flag=True, save=True)
```

Code 5.2: Visualisation tools

```
class Prep_Dataset():
    def __init__(self):
        self.list_of_event = ['None', 'Avbrudd']
        self.all_features()
        self.skip = 1 # reduce the size of the dataset timeseries by halve
        self.smooth = True
        self.std = True
        self.test_set = True
        self.test_size=0.33
        self.dataset_size = None # total size of dataset sampled from
    balanced_subsample function

    ##  Preparation of data set  ##
    def balanced_subsample(self, y):
        '''Returns the indexes to the sample matrix that gives balanced
        distributed dataset of the events present in the target vector.'''

        subsample = []

        if self.dataset_size is None:
            n_smp = y.value_counts().min()
        else:
            n_smp = int(self.dataset_size / len(y.value_counts().index))

        for label in y.value_counts().index:
            samples = y[y == label].index.values
            index_range = range(samples.shape[0])

            indexes = np.random.choice(index_range,
                                       size=n_smp,
                                       replace=False
                                      )
            subsample += samples[indexes].tolist()

        return subsample

    def standardise_dataset(self,X):
        '''Standardise feature columns in the dataset using sikit learn's
    standard
        scaler.'''

        X_std = np.array([])
        scaler = StandardScaler()
        num = X.shape[0]
        print('Standardising...')
        for i in range(num):
            X_std = np.append([X_std],[scaler.fit_transform(X[i])])
        X_std = X_std.reshape(X[:num].shape)
        print('Finished standardising')
        return X_std

    ## Sliding window indexer ##
    def window_indexer(self, metadata, len_time_series=60,
    prediction_horizon=60):
        '''
        len_time_series seconds of time series
        prediction_horizon   seconds from end of time series to fault
```

```
    event
55
56          Returns the numpy array indexes that corresponds to the
    initialization.
57          '''
58          resolution = float(metadata['resolution_ms'])
59          idx_fault_event = int(int(metadata['N_points'])   int(metadata['
    Time_buffer_sec'])*1000/resolution) #  30
60
61          end_idx = int(idx_fault_event   prediction_horizon*1000/resolution
    )
62          start_idx = int(end_idx   len_time_series*1000/resolution)
63          return start_idx, end_idx
64
65      def filter_event_by_type(self):
66          '''Returns the indexes of the numby array containing the event
    types.
67          Possible event types are None, Avbrudd, Jordfeil, and
    Spenningsdipp.
68          Need the target array as input.'''
69
70          self.sample_idx = np.array([])
71          for event_type in self.list_of_event:
72              self.sample_idx = np.append(self.sample_idx, np.argwhere(self.
    y==event_type).reshape(1,)).astype(np.int64)
73
74      def smooth_series(self, X, Wn=0.2, N=3):
75          '''Applies butterfiltering to each column to smooth the series.
76          N: order of the filter
77          Wh: crtical frequency.  The point at which the gain drops to 1/
    sqrt(2) that of the passband.
78          Returns the filtered version of the input.'''
79
80          n = X.shape[2]
81          samples = X.shape[0]
82          print('Smoothing..')
83          for s in range(samples):
84              for ftr in range(n):
85                  sig = X[s, :, ftr]
86                  b, a = signal.butter(N, Wn)
87                  filtered = signal.filtfilt(b, a, sig)
88                  X[s, :, ftr] = filtered
89          print('Finished smoothing')
90          return X
91
92
93
94      def load_dataset(self, path='/data/kristianh/init_dataset_20190419.h5'
    ):
95          f = tables.open_file(path,'a')
96          print(f)
97          self.X_init = f.get_node('/inputdata').read()
98          target = f.get_node('/target').read().astype(np.str)
99          self.feature_name = f.get_node('/featurename').read().astype(np.
    str)
100         self.metadata = f.get_node('/metadata').read()
101         f.close()
```

```
102
103         self.target = pd.Series(target)
104         self.y = self.target
105         self.sample_id = pd.Series(range(len(target)))
106
107     def delete_dataset(self):
108         del self.X_init
109
110     def time_var(self, len_tsrs=60, prd_hrz=1):
111         '''len_tsrs: length timeseries in seconds
112         prd_hrz: prediction horizon in seconds '''
113         self.start_idx, self.end_idx = self.window_indexer(self.metadata,
114                                                            len_tsrs,
115                                                            prd_hrz)
116
117     def select_events(self, event_type=['None', 'Avbrudd']):
118         self.list_of_event = event_type
119
120
121     def all_features(self):
122         '''Index of the features as they will appear in the data set.'''
123         self.ap = [0,1,2] #active power
124         self.rp = [183,184,185] #reactive power
125         self.hi1 = [5,6,8,10,14,16] #current harmonics phase1
126         self.hi2 = [25,26,28,30,34,36] # current harmonics phase2
127         self.hi3 = [45,46,48,50,54,56]# current harmonics phase3
128         self.hv1 = [65,66,68,70,74,76] # voltage harmonics phase1
129         self.hv2 = [85,86,88,90,94,96]# voltage harmonics phase2
130         self.hv3 = [105,106,108,110,114,116]# voltage harmonics phase3
131         self.hv12 = [125,126, 128,130,134,136]# voltage harmonics line
    phase1
132         self.hv23 = [145,146,148,150,154,156]# voltage harmonics line
    phase2
133         self.hv31 = [165,166,168,170,174,176]# voltage harmonics line
    phase3
134         self.cur = [ 7, 8, 9] # current
135         self.volli = [ 4, 5, 6] # phase voltage
136         self.volph =[ 1, 2, 3] # line voltage
137
138     def selected_features(self):
139         self.feature = self.ap+self.rp+self.hi1+self.hi2+self.hi3+self.hv1
    +self.hv2+self.hv3+self.hv12+self.hv23+self.hv31+self.cur+self.volli+
    self.volph
140
141     def generate_dataset(self):
142         y_sel = self.y[self.sample_idx]
143         # Get a balanced distributed dataset
144         balanced_index = self.balanced_subsample(pd.Series(y_sel))
145         y_sub = pd.Series(y_sel[balanced_index])
146
147         X_sub = np.nan_to_num(self.X_init[balanced_index, self.start_idx:
    self.end_idx,:][:,:, self.feature])
148
149         X_sub = X_sub[:,::self.skip,:] # reducing the data set by skipping
     every skip
150
151         # Get onehot encoded matrix of the target vector
```

```
        y_sub_dummy = pd.get_dummies(y_sub)

        self.nb_classes = y_sub_dummy.shape[1]
        target_names = y_sub_dummy.columns.values
        print(f'Target has the shape {y_sub_dummy.shape}, and the
    following categories: {target_names}.')

        if self.smooth:
            X_sub = self.smooth_series(X_sub) # smoothing first...

        if self.std:
            X_sub = self.standardise_dataset(X_sub) # ...then standardise

        if self.test_set:
            X_train, X_test, y_train, y_test = train_test_split(X_sub,

    y_sub_dummy,
                                                                test_size=
    self.test_size,

    random_state=self.seed,
                                                                stratify=
    y_sub_dummy
                                                               )
            print(f'Training set has the shape {X_train.shape}\n')
            print('Train set')
            print(y_train.sum(),'\n')
            print('Test set')
            print(y_test.sum())

            del X_sub # maybe this does something good...

            return X_train, X_test, y_train, y_test
        else:
            X_train = X_sub
            y_train = y_sub_dummy
            X_test = None
            y_test = None
            print(f'Training set has the shape {X_train.shape}\n')
            print('Train set')
            print(y_train.sum(),'\n')

            del X_sub # maybe this does something good...

            return X_train, y_train
```

Code 5.3: Data set preparation

```
class S2S_Autoencoder():
    def __init__(self):
        self.batch_size = 64  # Batch size for training.
        self.epochs = 20  # Number of epochs to train for.
        self.latent_dim = 250  # Latent dimensionality of the encoding
    space.
        self.num_samples = 10000  # Number of samples to train on.
        self.val_split=0.1
        self.verbose=1

        self.C = 0.01
        self.lr_cla = 0.01
        self.lr_rec = 0.001
        #variables for building Seq2Seq
        self.latent_dim1 = 250
        self.latent_dim2 = 250

        # initialize loss functions optimizers and metrics
        self.activation_cla='tanh' # softmax
        #self.activation_rec = 'softmax' # none
        self.optimizer_cla = 'adam'
        self.optimizer_rec='rmsprop'
        self.loss_cla='categorical_hinge' #categorical_crossentropy
        self.loss_rec= 'mse'
        self.loss_weights=[0.8,1.0]
        self.metrics_cla=['accuracy', self.matthews_correlation]
        self.metrics_rec=['mse']

    def fit_autoencoder(self):
        '''Trains the encoder and decoder layers of the autoencoder model.
        The reconstruction sequence is the reverse of the input sequence.
    '''

        print('\nTraining autoencoder\n')

        self.decoder_input_data = np.zeros(self.X_train.shape)
        self.decoder_target_output = np.flip(self.X_train, 2)
        encoder_input_data = self.X_train

        self.compile_autoencoder()

       K.set_value(self.s2s_aec.optimizer.lr, self.lr_rec) # specify
    learningrate for model optimizer

        self.history_aec = self.s2s_aec.fit([encoder_input_data, self.
    decoder_input_data],
                                            self.decoder_target_data,
                                             batch_size=self.batch_size,
                                             epochs=self.epochs,
                                             validation_split=self.val_split
    ,
                                             verbose=self.verbose,
                                             callbacks=[self.tensorboard]
                                             )

    def compile_autoencoder(self):
        self.s2s_aec.compile(optimizer=self.optimizer_rec,
```

```
                          loss=self.loss_rec,
                          metrics=self.metrics_rec)

    def fit_cla(self):
        '''Pre treains only the classifier related layers.'''

        print('\nTraining classifier\n')
        self.s2s_cla_model.layers[1].trainable = False # set the layers
    corresponding to the encoder to non trainable
        self.s2s_cla_model.layers[2].trainable = False

        self.compile_classifier()

       K.set_value(self.s2s_cla_model.optimizer.lr, self.lr_cla) #
    specify learningrate for model optimizer

        self.history_cla = self.s2s_cla_model.fit(
            self.X_train,
            self.y_train,
            batch_size=self.batch_size,
            validation_split=self.val_split,
            epochs=self.epochs,
            verbose=self.verbose,
            callbacks=[self.tensorboard]
            )

    def compile_classifier(self):
        self.s2s_cla_model.compile(optimizer=self.optimizer_cla,
                                   loss=self.loss_cla,
                                   metrics=self.metrics_cla)

    def fit_composite(self):
        '''Trains all layers of the model, takning into account both the
        reconstruction loss and the prediction loss. The reconstruction
        sequence is the reverse of the input sequence.'''

        print('\nTraining compsitemodel\n')
        self.s2s_composite_model.layers[2].trainable = True # set the
    layers corresponding to the encoder to trainable
        self.s2s_composite_model.layers[4].trainable = True

        self.compile_composite()

       K.set_value(self.s2s_composite_model.optimizer.lr, self.lr_rec) #
    specify learningrate for model optimizer

        self.history_composite = self.s2s_composite_model.fit(
                                        [self.X_train, self.
    decoder_input_data],
                                        {'reconstruction': self.
    decoder_target_data,
                                        'classifier': self.y_train},
                                        validation_split=self.val_split,
                                        epochs=self.epochs,
                                        verbose=self.verbose,
                                        callbacks=[self.tensorboard]

```

```
        )

    def compile_composite(self):
        self.s2s_composite_model.compile(
                optimizer=self.optimizer_cla,
                loss=self.losses,
                loss_weights=self.lossWeights,
                metrics=self.metrics
                )

    def encode_input(self, X=[]):
        '''Encodes the input and creates the representation into activity
        vector of the output unit '''
        # Encode the input as state vectors.
        if len(X)==0:
            X = self.X_train
        h1, c1  = self.encoder_model1.predict(X) # state vectors of first
    layer
        h2, c2  = self.encoder_model2.predict(X) # state vectors of second
     layer

        self.h = [h1, h2]
        self.c = [c1, c2]


    def predict(self, X):
        '''Predicting the event of the input data and reconstructing the
    input sequencees.'''
        self.X = X

        self.reconstruction, predicted_event = self.s2s_composite_model.
    predict(self.X)
        predict_event_index = predicted_event.argmax(axis=1)
        y_pred = self.y_train.columns.values[predict_event_index]
        return y_pred

    def reconstruct(self):
        '''Returns the reconstruction from the decodert of the input
    sequence.'''
        return self.reconstruction

    def return_state(self, sequences):
        '''Returns the hidden states of the output of the last layer in
    the encoder.'''
        #yhat = self.encoder(self.X)
        return self.output_tokens, self.h

    def save_model(self, model, filename='s2s.h5'):
        model.save(filename)

    def encoder_decoder_layers(self):
        '''Defines the base layers of the encoder decoder which the model
    is
        constructed upon.'''

        # Define an input sequence and process it.
        self.encoder_inputs = Input(shape=(self.timesteps, self.input_dim)
```

```
    ,name='Encoder_input')
        encoder = LSTM(self.latent_dim1,
                       #activation='relu',
                       return_sequences=True,
                       return_state=True,
                       #dropout=0.1,
                       name='Encoder_hidden_layer1')
        self.encoder_outputs1, state_h1, state_c1 = encoder(self.
    encoder_inputs)
        encoder2 = LSTM(self.latent_dim2,
                        #activation='relu',
                        return_state=True,
                       # dropout=0.1,
                        name='Encoder_hidden_layer2')
        self.encoder_outputs2, state_h2, state_c2 = encoder2(self.
    encoder_outputs1)
        # We discard `encoder_outputs` and only keep the states.
        self.encoder_states1 = [state_h1, state_c1]
        self.encoder_states2 = [state_h2, state_c2]

        # Set up the decoder, using `encoder_states` as initial state.
        self.decoder_inputs = Input(shape=(None, self.input_dim), name='
    Decoder_ghost_input')
        # We set up our decoder to return full output sequences,
        # and to return internal states as well. We don't use the
        # return states in the training model.
        decoder_lstm1 = LSTM(self.latent_dim1,
                             #activation='relu',
                             return_sequences=True, return_state=True,
                             name='Decoder_hidden_layer1')
        decoder_outputs1, _, _ = decoder_lstm1(self.decoder_inputs,
                                               initial_state=self.
    encoder_states1)
        decoder_lstm2 = LSTM(self.latent_dim2,
                             #activation='relu',
                             return_sequences=True, return_state=True,
                             name='Decoder_hidden_layer2')
        self.decoder_outputs2, _, _ = decoder_lstm2(decoder_outputs1,
                                               initial_state=self.
    encoder_states2)


        decoder_dense = Dense(self.input_dim, activation='linear', name='
    reconstruction')
        self.decoder_outputs = decoder_dense(self.decoder_outputs2)

    def classifier_model(self):
        # classifier
        #hidden = Dense(self.input_dim,activation='relu')(self.
    encoder_states2[0]) #RBFLayer(10,0.1)(self.encoder_states2[0])
        num_landmark = 30
        #inp = Input(shape=(self.latent_dim2,), name='inp')
        self.output = self.encoder_outputs2
        #self.output = Dense(self.latent_dim2,activation='relu')(self.
    output)
        self.output = GaussianKernel3(num_landmark, self.latent_dim2, name
    ='gkernel1')(self.output)
```

```
        #self.output =self.output(self.encoder_outputs2)#self.
    encoder_states2[1])#)
        self.output = Dense(self.nb_classes, activation=self.
    activation_cla, name='classifier')(self.output)


        #cla
        #hidden = Dense(self.nb_classes, kernel_regularizer=l2(self.C),
    name='cla_fc')
        #hidden = hidden(self.encoder_states2[0])#self.encoder_outputs2)
        #self.output = Activation(self.activation_cla, name='classifier')(
    hidden)

        # predict model
        self.s2s_cla_model = Model(
            inputs=self.encoder_inputs,
            outputs=self.output,
            name='s2s_cla'
        )

    def reconstruction_model(self):
        # reconstruction model
        self.s2s_aec = Model([self.encoder_inputs, self.decoder_inputs],
    self.decoder_outputs)

    def composite_model(self):
        # composite model
        self.s2s_composite_model = Model(
            inputs=[self.encoder_inputs, self.decoder_inputs],
            outputs=[self.decoder_outputs, self.output],
            name='s2s_cla_rec'
        )

        # definig which loss function each layer of the composite model
    should have
        self.losses = {
            'reconstruction': self.loss_rec,
            'classifier': self.loss_cla
        }

        # defining the weights of the loss functions used
        self.lossWeights = {'reconstruction': self.loss_weights[0], '
    classifier': self.loss_weights[1]}
        self.metrics ={'reconstruction': self.metrics_rec, 'classifier':
    self.metrics_cla}
        #self.optimizers = {'reconstruction': self.optimizer_rec, '
    classifier': self.optimizer_cla}

    def encoder_model(self):
        # isolating only the encoder part
        self.encoder_model1 = Model(self.encoder_inputs,
                                    self.encoder_states1
                                   )

        self.encoder_model2 = Model(self.encoder_inputs,
                                    self.encoder_states2
                                   )
```

```
249
250     def build_model(self,
251                     X_train,
252                     y_train,
253                     summaries=True,
254                    ):
255         ''' This function creates a rnn model. Number of layers and memory
     units\
256         are defined by the hidden_layer argument where the number defines
    the \
257         units and the length the number of layers. y_train is a dummy
    matrix.'''
258
259         # create tensorboard monitoring. Awesome in Jupyterlab!
260         time = datetime.today().strftime('%d%m%y %H:%M:%S')
261         self.tensorboard = TensorBoard(log_dir=f"./log/{time}")
262
263         self.X_train = X_train
264         self.y_train = y_train
265
266         self.nb_classes = y_train.shape[1]
267
268         self.timesteps = self.X_train.shape[1]
269         self.input_dim = self.X_train.shape[2]
270
271         # define target and input data
272         self.encoder_input_data = self.X_train
273         self.decoder_input_data = np.zeros(self.X_train.shape)
274         self.decoder_target_data = np.flip(self.X_train, 2)
275         #timesteps = self.X_train.shape[1]; input_dim = self.X_train.shape
    [2]
276
277         # Create the diferent parts of the model
278         self.encoder_decoder_layers()
279         self.reconstruction_model()
280         self.classifier_model()
281         self.composite_model()
282         self.encoder_model()
283
284         if summaries:
285             print('Summary of the autoencoder')
286             print(self.s2s_aec.summary())
287
288             print('Summary of the classifier model')
289             print(self.s2s_cla_model.summary())
290
291             print('Summary of the composit model')
292             print(self.s2s_composite_model.summary())
293
294             print('Summary of the encoder part of the model')
295             print(self.encoder_model.summary())
296
297     def matthews_correlation(self, y_true, y_pred):
298         '''Calculates the Matthews correlation coefficient measure for
    quality
299         of binary classification problems.
300         '''
```

```
        y_pred_pos = K.round(K.clip(y_pred, 0, 1))
        y_pred_neg = 1   y_pred_pos

        y_pos = K.round(K.clip(y_true, 0, 1))
        y_neg = 1   y_pos

        tp = K.sum(y_pos * y_pred_pos)
        tn = K.sum(y_neg * y_pred_neg)

        fp = K.sum(y_neg * y_pred_pos)
        fn = K.sum(y_pos * y_pred_neg)

        numerator = (tp * tn   fp * fn)
        denominator = K.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn)
    )

        return numerator / (denominator + K.epsilon())

    # these plot functions creates diagram images of each part of the
    model
    def plot_aec(self, filename='fig/s2s_autoencoder.svg'):
        plot_model(self.s_aec, show_shapes=True,
                   show_layer_names=True,
                   to_file=filename
                   )

    def plot_cla(self):
        plot_model(self.s2s_cla_model,
                   show_shapes=True,
                   show_layer_names=True,
                   to_file='fig/s2s_autoencoder_cla.svg'
                   )

    def plot_composit(self):
        plot_model(self.s2s_composite_model,
                   show_shapes=True,
                   show_layer_names=True,
                   to_file='fig/s2s_autoencoder_composite.svg'
                   )
```

Code 5.4: Sequence-to-sequence autoencoder class

```
def build_rnn_model(X_train,
                    target,
                    rnn_type='lstm',
                    hidden_layer=[250],
                    optimizer='adam',
                    loss='categorical_crossentropy',
                    activation='softmax',
                    metrics=['accuracy']
                   ):
    ''' This function creates a rnn model. Number of layers and memory
    units\
    are defined by the hidden_layer argument where the number defines the
    \
    units and the length the number of layers.'''

    nb_classes = target.shape[1]

    if rnn_type == 'lstm':
        layer = LSTM
    elif rnn_type == 'gru':
        layer = GRU

    visible = Input(shape=(X_train.shape[1], X_train.shape[2])) # visible

    if len(hidden_layer) == 1:
        model_layer = layer(hidden_layer[0], kernel_regularizer=
    regularizers.l2(0.01))(visible)
    else:
        model_layer = layer(hidden_layer[0], kernel_regularizer=
    regularizers.l2(0.01), return_sequences=True)(visible)

        for units in hidden_layer[1:2]:
            model_layer = layer(units, kernel_regularizer=regularizers.l2
    (0.01), return_sequences=True)(model_layer)
        model_layer = layer(hidden_layer[1],)(model_layer)

    model_layer = GaussianKernel3(30,hidden_layer[1], name='gkernel1')(
    model_layer)
    output = Dense(nb_classes, kernel_regularizer=regularizers.l2(0.01),
    activation=activation)(model_layer)

    model = Model(inputs=visible, outputs=output)

    #model.layers[2].return_sequences = False   #set the last recurrent
    layer to not return sequences

    model.compile(optimizer=optimizer,
                  loss=loss,
                  metrics=metrics)
    model.summary()
    return model
```

Code 5.5: LSTM/GRU build function

```
class Tsne():
    def __init__(self):
        self.X = pd.DataFrame([])

    def fit_transform(self, X, n_components=2, verbose=1, perplexity=40,
    n_iter=300):
        self.X = X
        time_start = time.time()
        tsne = TSNE(n_components=n_components, verbose=verbose, perplexity
    =perplexity, n_iter=n_iter)
        self.tsne_results = tsne.fit_transform(self.X.values[:,: 2])

        print('t SNE done! Time elapsed: {} seconds'.format(time.time()
    time_start))

    def plot(self,
             x=None,
             y=None,
             n_sne=1,
             hue='label',
             legend='full',
             palette=None,
             alpha=0.1,
             s=70,
             style=None,
             filename='t sne plot',
             title='t SNE dimensions colored by digit'
             ):

        df_tsne = self.X.copy()
        df_tsne['x tsne'] = self.tsne_results[:,0]
        df_tsne['y tsne'] = self.tsne_results[:,1]

        if palette:
            num_labels = len(df_tsne[hue].unique())
            palette = sns.color_palette("bright", num_labels)

        fig, ax = plt.subplots(figsize=(10,7))
        ax.set_title(title)
        sns.scatterplot(data=df_tsne, x=df_tsne.columns[ 2],
                        y=df_tsne.columns[ 1],
                        hue=hue,
                        legend=legend,
                        palette=palette,
                        alpha=alpha,
                        s=s,
                        style=style
                        )
        plt.savefig(filename)
        plt.show()

    def tsne_results(self):
        return self.tsne_results

    def X(self):
        return self.X
```

Code 5.6: t-SNE

```
class GaussianKernel3(Layer):

    def __init__(self, num_landmark, num_feature,
                 kernel_initializer='glorot_uniform',
                 **kwargs):
        '''
        num_landmark:
            number of landmark
            that was number of output features
        num_feature:
            depth of landmark
            equal to inputs.shape[1]
        '''
        super(GaussianKernel3, self).__init__(**kwargs)

        self.output_dim = num_landmark
        self.num_feature = num_feature
        self.kernel_initializer = initializers.get(kernel_initializer)

        # for loop
        self.indx = K.arange(self.output_dim)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)

    def build(self, input_shape):
        assert len(input_shape) == 2
        input_dim = input_shape[1]

        self.kernel = self.add_weight(name='kernel',
                                      shape=(self.output_dim, self.
    num_feature),
                                      initializer=self.kernel_initializer)
        self.gamma_elm = self.add_weight(name='gamma_elm',
                                      shape=(1, ),
                                      initializer=initializers.
    random_uniform(2,  1))
        super(GaussianKernel3, self).build(input_shape)  # Be sure to call
     this somewhere!

    def call(self, x, training=None):
        return self.gauss(x, self.kernel, K.exp(self.gamma_elm))

    def gauss(self, x, landmarks, gamma):
        def fn(ii):
            lm = K.gather(landmarks, ii)
            return K.sum(K.square(x   lm), axis=1)
        d2 = K.map_fn(fn, self.indx, dtype='float32')
        d2 = K.transpose(d2)

        return K.exp( gamma * d2)


###     Evaluation functions        ###

def predict(model, X_test):
    return model.predict(X_test)
```

```
55
56 def wrongly_predicted(y_true, y_pred):
57     '''Returns the numpy array indices of the wrongly predicted samples.
       '''
58     indices = [i for i,v in enumerate(y_pred) if np.argmax(y_pred[i])!=np.
       argmax(y_true[i])]
59     return indices
60
61 def subset_of_wrongly_predicted(X_test, indices=[]):
62     '''Returns a subset of the wrongly predicted samples'''
63     subset_of_wrongly_predicted = [X_test[i] for i in indices ]
64     return subset_of_wrongly_predicted
65
66 # Confunsion matrix
67 def conf_matrix(y_true, y_pred):
68     '''Returns the confusion matrix of the true categories and the
       predicted.'''
69     matrix = confusion_matrix(y_true.values.argmax(axis=1),
70                                       y_pred.argmax(axis=1))
71     conf_matrix = pd.DataFrame(matrix, index=y_true.columns.values,
72                                columns=y_true.columns.values)
73     return conf_matrix
74
75 ## Plot functions
76
77 def plot_training_validation_curve(history):
78     # Plot training & validation accuracy values
79     plt.plot(history.history['cla_acc'])
80     plt.plot(history.history['val_cla_acc'])
81     plt.title('Model accuracy')
82     plt.ylabel('Accuracy')
83     plt.xlabel('Epoch')
84     plt.legend(['Train', 'Validation'], loc='upper left')
85     plt.show()
86
87     # Plot training & validation loss values
88     plt.plot(history.history['loss'])
89     plt.plot(history.history['val_loss'])
90     plt.title('Model loss')
91     plt.ylabel('Loss')
92     plt.xlabel('Epoch')
93     plt.legend(['Train', 'Validation'], loc='upper left')
94     plt.show()
95
96 def plot_confusion_matrix(cm, classes, filename=None,
97                           normalize=False,
98                           title='Confusion matrix',
99                           cmap=plt.cm.Blues):
100     """
101     This function prints and plots the confusion matrix.
102     Normalization can be applied by setting `normalize=True`.
103     """
104     if normalize:
105         cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
106         print("Normalized confusion matrix")
107     else:
108         print('Confusion matrix, without normalization')
```

```
109
110     print(cm)
111
112     plt.imshow(cm, interpolation='nearest', cmap=cmap)
113     plt.title(title)
114     plt.colorbar()
115     tick_marks = np.arange(len(classes))
116     plt.xticks(tick_marks, classes, rotation=45)
117     plt.yticks(tick_marks, classes)
118
119     fmt = '.2f' if normalize else 'd'
120     thresh = cm.max() / 2.
121     for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
122         plt.text(j, i, format(cm[i, j], fmt),
123                  horizontalalignment="center",
124                  color="white" if cm[i, j] > thresh else "black")
125
126     plt.ylabel('True label')
127     plt.xlabel('Predicted label')
128     plt.tight_layout()
129     if filename:
130         plt.savefig(filename)
131     plt.show()
132
133 def multiclass_ROC_AUC(y_test, y_score, filename):
134     '''Computes the ROC for all classes and plot all curves in the same
    figure'''
135
136     n_classes = y_test.shape[1]
137     class_names = list(y_test.columns)
138
139     # Compute ROC curve and ROC area for each class
140     fpr = dict()
141     tpr = dict()
142     roc_auc = dict()
143     for i in range(n_classes):
144         fpr[i], tpr[i], _ = roc_curve(y_test.values[:, i], y_score[:, i])
145         roc_auc[i] = auc(fpr[i], tpr[i])
146
147     # Compute micro average ROC curve and ROC area
148     fpr["micro"], tpr["micro"], _ = roc_curve(y_test.values.ravel(),
    y_score.ravel())
149     roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
150
151
152     # Compute macro average ROC curve and ROC area
153
154     # First aggregate all false positive rates
155     all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)])
    )
156
157     # Then interpolate all ROC curves at this points
158     mean_tpr = np.zeros_like(all_fpr)
159     for i in range(n_classes):
160         mean_tpr += interp(all_fpr, fpr[i], tpr[i])
161
162     # Finally average it and compute AUC
```

```
    mean_tpr /= n_classes

    fpr["macro"] = all_fpr
    tpr["macro"] = mean_tpr
    roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

    # Plot all ROC curves
    plt.figure(figsize=(8,8))
    lw = 2
    plt.plot(fpr["micro"], tpr["micro"],
             label='micro average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["micro"]),
             color='deeppink', linestyle=':', linewidth=4)

    plt.plot(fpr["macro"], tpr["macro"],
             label='macro average ROC curve (area = {0:0.2f})'
                   ''.format(roc_auc["macro"]),
             color='navy', linestyle=':', linewidth=4)

    colors = itertools.cycle(['aqua', 'darkorange', 'cornflowerblue', 'lime'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                 label='ROC curve of class {0} (area = {1:0.2f})'
                 ''.format(class_names[i], roc_auc[i]))

    plt.plot([0, 1], [0, 1], 'k  ', lw=lw)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Some extension of Receiver operating characteristic to multi class')
    plt.legend(loc="lower right")
    plt.savefig(filename)
    plt.show()

def prediction_overview(y_train, y_test, indices):
    print('Events trained on')
    print(y_train.sum(), '\n')

    print('Total events to be predicted')
    print(y_test.sum(), '\n')

    print('Samples wrongly classified')
    print(y_test.iloc[indices].sum())

def metric_evaluation(y_test, y_pred):
    mcc = matthews_corrcoef(y_test.values.argmax(axis=1), y_pred.argmax(axis=1))
    f1 = f1_score(y_test.values.argmax(axis=1), y_pred.argmax(axis=1), average='weighted')
    print(f'Matthesw correlation coefficient: {mcc}')
    print(f'F1 measure: {f1}')
    return mcc, f1

def get_smart(y):
```

```
216     '''Reverse dummy a dummy matrix'''
217     i, j = np.where(y)
218     return pd.Series(y.columns[j], i)
```

Code 5.7: Miscellaneous functions

Appendix C: Proof of harmonic component sequences

Proof of the sequence configuration of some central harmonic components in the electrical power system.

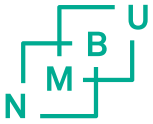
Fundamental - Positive sequence
$i_{a1} = I_{a1} sin\omega t$
$i_{b1} = I_{b1} sin(\omega t - 120°)$
$i_{c1} = I_{c1} sin(\omega t - 240°)$

Second harmonic - Negative sequence
$i_{a2} = I_{a2} sin(2\omega t)$
$i_{b2} = I_{b2} sin(2(\omega t - 120°)) = I_{b2} sin((2\omega t - 240°))$
$i_{c2} = I_{c2} sin(2(\omega t - 240°)) = I_{c2} sin((2\omega t - 480°)) = I_{c2} sin(2\omega t - 120°)$

Third harmonic - Zero sequence
$i_{a3} = I_{a3} sin(3\omega t)$
$i_{b3} = I_{b3} sin(3(\omega t - 120°)) = I_{b2} sin((3\omega t - 360°)) = I_{b2} sin(3\omega t)$
$i_{c2} = I_{c2} sin(3(\omega t - 240°)) = I_{c2} sin((3\omega t - 720°)) = I_{c2} sin(3\omega t)$

Fifth harmonic - Negative sequence
$i_{a5} = I_{a5} sin(5\omega t)$
$i_{b5} = I_{b5} sin(5(\omega t - 120°)) = I_{b5} sin((5\omega t - 600°)) = I_{b5} sin(5\omega t - 240°)$
$i_{c5} = I_{c5} sin(5(\omega t - 240°)) = I_{c5} sin((5\omega t - 1200°)) = I_{c5} sin(5\omega t - 120°)$

Seventh harmonic - Positive sequence
$i_{a7} = I_{a7} sin(7\omega t)$
$i_{b7} = I_{b7} sin(7(\omega t - 120°)) = I_{b7} sin((7\omega t - 840°)) = I_{b7} sin(7\omega t - 120°)$
$i_{c7} = I_{c7} sin(7(\omega t - 240°)) = I_{c7} sin((7\omega t - 1680°)) = I_{c7} sin(7\omega t - 240°)$



Norges miljø- og biovitenskapelige universitet
Noregs miljø- og biovitskapelege universitet
Norwegian University of Life Sciences

Postboks 5003
NO-1432 Ås
Norway