POLYPUBLIE
Polytechnique Montréal

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Titre: Title:	A Deep Learning Approach for Condition-Based Fault Prediction in Industrial Equipment
Auteur: Author:	Daniel Buades Marcos
Date:	2019
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Buades Marcos, D. (2019). A Deep Learning Approach for Condition-Based Fault Prediction in Industrial Equipment [Master's thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4120/

Document en libre accès dans PolyPublie
Open Access document in PolyPublie

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/4120/
Directeurs de recherche: Advisors:	Soumaya Yacout
Programme: Program:	Maîtrise recherche en génie industriel

Ce fichier a été téléchargé à partir de PolyPublie, le dépôt institutionnel de Polytechnique Montréal
This file has been downloaded from PolyPublie, the institutional repository of Polytechnique Montréal

https://publications.polymtl.ca

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

A Deep Learning Approach for Condition-Based Fault Prediction in Industrial Equipment

DANIEL BUADES MARCOS

Département de mathématiques et de génie industriel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

Génie industriel

Décembre 2019

© Daniel Buades Marcos, 2019.

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

A Deep Learning Approach for Condition-Based Fault Prediction in Industrial Equipment

présenté par **Daniel BUADES MARCOS**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*

a été dûment accepté par le jury d'examen constitué de :

BASSETTO Samuel Jean, président

YACOUT Soumaya, membre et directrice de recherche

ALOISE Daniel, membre

DEDICATION

"Más vale ~~prevenir~~ predecir que curar."

Refrán

"~~Prevention~~ Prediction is better than cure."

Proverb

ACKNOWLEDGMENTS

I would like to start by thanking the members of my jury – M. Samuel Bassetto and M. Daniel Aloise – who didn't hesitate to accept the task of reviewing this thesis when I first asked them. Thank you!

Next, I want to thank my research director, Mme. Soumaya Yacout. We first met three years ago, when I was just an exchange student, and you agreed to direct my final undergraduate project. Thank you for trusting in me during these years, both with the master's and with the course. I hope it was at least half as pleasant as it has been for me.

I would also like to thank R2 and especially Said Berriah. Thank you for giving me the chance to learn with a real project and, most importantly, making me feel welcome in your office during this past year and a half.

This thesis marks the end of my academic period (at least for the foreseeable future). Like every good run, it has been hard at times, but oh, it has been so worth it. However, I know it wouldn't have been so much fun hadn't it been for the people that have surrounded me during these years. From Murcia to Valencia to Montreal, you know who you are (and if you are reading this thesis, it is because you are definitely one of them).

On a special note, I want to mention my parents. You have supported me since I can remember (and surely before that too), I wouldn't have gotten here otherwise.

Finally, on a more informal note, I cannot miss this opportunity to thank Montreal's weather. Thank you for those cold days where I did not have any remorse for being locked up writing this thesis instead of going outside.

Thank you, merci, gracias.

Daniel

RÉSUMÉ

Tout type de système se dégrade avec l'usage et le temps. Adopter une bonne stratégie de maintenance est donc nécessaire afin que les équipements industriels fonctionnent adéquatement tout au long de leur durée de vie. Au fur et à mesure que l'importance accordée à la santé et la sécurité augmente dans les industries, la maintenance devient de plus en plus imposante. Ceci conduit donc au développement de stratégies de maintenance plus sophistiquées. Parmi ces stratégies, la Maintenance Conditionnelle (« Condition-Based Maintenance ») est la plus avancée d'entre elles. Par ailleurs, avec l'avènement de l'Industrie 4.0, le nombre de capteurs dans l'industrie augmente rapidement, ouvrant la porte à la surveillance continue et automatique des équipements industriels.

Plus particulièrement, nous présentons une technique où l'indicateur de fautes futures est basé sur la différence entre la valeur d'une variable mesurable quelconque et la valeur attendue de cette variable lors de son opération normale. Le fait de devoir estimer correctement la valeur de ce paramètre fait intervenir la maintenance en plus de la prévision. Nous proposons alors d'utiliser l'apprentissage machine (« Machine Learning »), plus précisément un réseau de neurones profond (« Deep Neural Network »), afin de prédire la valeur du paramètre. La théorie dernière ces techniques est présentée dans ce travail, portant une attention particulière aux réseaux LSTM (« Long Short-Term Memory »). Ce type d'architecture neuronale est adaptée pour la prévision impliquant des séries temporelles à plusieurs variables, qui est le type de données enregistrées par les capteurs en industrie.

Dans ce travail, la technique de prédiction de fautes est appliquée aux électrolyseurs. Un électrolyseur est un système composé de plusieurs cellules électrochimiques où l'électrolyse a lieu. La présence de fautes inattendues peut mener à des incidents catastrophiques pouvant causer de grands torts aux employés et à l'environnement. Les cellules électrochimiques se comportent comme une résistance électrique. En effet, lorsqu'un courant est appliqué à ses bornes, ceci provoque une chute de tension. L'amplitude de cette chute de tension dépend des conditions d'opération ainsi que de la dégradation de la cellule, qui n'est pas connue. De ce fait, la tension est la variable surveillée du système afin de pouvoir prévoir les fautes, puisque sa valeur augmente soudainement lorsqu'une faute est sur le point de se produire.

Nous présentons les limitations reliées aux données disponibles. Afin de surmonter ces limitations, nous voulons nous baser seulement sur les données provenant des capteurs et non pas sur une intervention humaine. Nous proposons donc une nouvelle approche basée sur un réseau de neurones de type encodeur-décodeur. Le réseau reçoit les conditions de fonctionnement en entrée. Le rôle de l'encodeur est de trouver une représentation fidèle de la dégradation et de la transmettre au décodeur. Quant à lui, son rôle est de prédire la tension de la cellule. Comme aucune donnée de dégradation n'est fournie au réseau, nous considérons notre approche comme étant un encodeur auto-supervisé (« Self-Supervised Encoder »). De même, la sortie de l'encodeur peut être représentée graphiquement, rendant possible l'interprétation du réseau de neurones développé.

Après le développement du modèle, il est testé et les résultats obtenus sont comparés avec ceux du modèle paramétrique présentement utilisé à cette fin, qui est défini par des experts. Les résultats montrent que le réseau de neurones est apte à prédire le voltage de plusieurs cellules comportant des niveaux de dégradation différents. Par ailleurs, l'erreur de prédiction est réduite de 53 % par rapport au modèle paramétrique. Cette amélioration permet à notre modèle de prédire une faute 31 heures avant qu'elle ne se produise. Ceci correspond à une augmentation du temps de réaction de 64 % comparativement aux résultats obtenus avec le modèle paramétrique.

ABSTRACT

Systems always degrade with use and time. A good maintenance strategy is necessary for keeping industrial equipment working as designed throughout its lifetime. As safety and quality concerns raised in industry, the importance of maintenance evolved, leading to the development of sophisticated maintenance strategies. Among them, Condition-Based Maintenance is the most advanced. It is based on the monitorization of the system's properties, detecting impending faults, and triggering maintenance actions only when the system is going to fail. Moreover, with the advent of Industry 4.0, the number of sensors in industry is rapidly increasing, opening the door to the continuous and automatic monitorization of industrial equipment.

In particular, we present a technique where the indicator of an incoming fault is based on the divergence between the value of a measurable feature and its expected value in a healthy system. The need to correctly estimate the variable's value involves both maintenance and forecasting. We propose using Machine Learning algorithms for predicting the variable and, more concretely, Deep Neural Networks. We introduce the theory behind them, paying particular attention to Long Short-Term Memory (LSTM) networks. This neural architecture is suited for forecasting multivariate time series data, which is the one registered by industrial sensors.

We apply this fault prediction technique to an electrolyzer. Electrolyzers are systems composed of multiple electrochemical cells, where electrolysis takes place. Unexpected cells' faults may lead to catastrophic incidents, such as explosions and fire, with the consequent harm to the plant's operators and the environment. Electrochemical cells act similarly to resistors, causing a voltage drop when a current is applied to them. The magnitude of this voltage drop depends on the operating conditions and the cell's degradation, which is not known. The voltage is the monitored variable for predicting the faults, as it increases suddenly when a fault is about to happen.

We present the limitations related to the available data. In order to overcome them, relying only on sensors' data and without human interaction, we propose a new approach based on an encoder-decoder neural architecture. The network receives the operating conditions as input. The encoder's task is to find a faithful representation of the degradation and to pass it to the decoder, which in turn predicts the cell's voltage. As no labeled degradation data is given to the network, we consider

our approach to be a self-supervised encoder. Moreover, the output of the encoder can be plotted in a graph, adding interpretability to the neural network model.

We compare the results obtained by our network to the expert-defined parametric model that is currently used for this task. Results show that our neural network is able to predict the voltage of multiple cells with different levels of degradation. Moreover, it reduces the prediction error that was obtained by the parametric model by 53%. This improvement enabled our network to predict a fault 31 hours before it happened, a 64% increase in reaction time compared to the parametric model.

TABLE OF CONTENTS

DEDICATION III

ACKNOWLEDGMENTS IV

RÉSUMÉ V

ABSTRACT VII

TABLE OF CONTENTS IX

LIST OF TABLES XII

LIST OF FIGURES XIII

LIST OF SYMBOLS AND ABBREVIATIONS XV

CHAPTER 1 INTRODUCTION 1

1.1 Historical Evolution of Maintenance 2

1.2 Maintenance Strategies 6

1.2.1 Corrective Maintenance 6

1.2.2 Preventive Maintenance 6

1.2.2.1 Fixed interval replacement 7

1.2.2.2 Fixed age replacement 8

1.2.3 Condition-Based Maintenance 8

1.2.3.1 Fault Prediction 11

1.3 Research Objective 13

CHAPTER 2 LITERATURE REVIEW 15

2.1 Supervised Machine Learning 15

2.1.1 Bias-Variance trade-off 16

2.1.2 Data Types 17

2.2 Neural Networks 18

2.2.1 Recurrent Neural Networks 21
2.2.1.1 Long Short-Term Memory Networks 23
2.2.2 Autoencoders 25
CHAPTER 3 ARTICLE INTRODUCTION 27
CHAPTER 4 ARTICLE 1: USING A SELF-SUPERVISED ENCODER FOR PREDICTING FAULTS IN ELECTROCHEMICAL CELLS 28
4.1 Problem Description 29
4.2 Previous Work 30
4.2.1 Originality 33
4.3 Data preparation 34
4.3.1 Features 34
4.3.2 Cycles 36
4.4 Methodology 38
4.4.1 Data Processing 38
4.4.2 Model Architecture 41
4.4.2.1 Encoder 43
4.4.2.2 Predictor 44
4.4.3 Training Procedure 45
4.4.3.1 Padding 46
4.4.3.2 Shuffling 47
4.4.3.3 Window Striding 48
4.4.4 Testing Procedure 49
4.4.4.1 Parametric Model 49
4.4.4.2 Method 49

4.5 Results and Discussion 50
4.5.1 Network insight 50
4.5.2 Accuracy 54
4.5.2.1 Between cycles 54
4.5.2.2 Inside cycle 55
4.5.3 Fault Detection 56
4.6 Conclusion 57
4.7 Future Work 58
4.8 Acknowledgments 58
CHAPTER 5 GENERAL DISCUSSION 59
CHAPTER 6 CONCLUSION AND RECOMMENDATIONS 61
REFERENCES 63

LIST OF TABLES

Table 4.1: Subset of the dataset. n represents the last observation, and m the last electrolyzer's cell. 35

Table 4.2: Pseudo-code for detecting possible cycles. 38

Table 4.3: Pseudo-code for confirming cycles. 38

Table 4.4: Subset of data from Cycle A after being processed. Data has been scaled and missing observations, replaced by '-1'. Note that, as expected, the input features at each time-step are the same for cells one and m, where m represents the last cell of the electrolyzer. 40

Table 4.5: Pseudo-code for the stochastic training algorithm. 45

Table 4.6: Padded startup. "i" represents the last startup observation before padding. 47

Table 4.7: Statistics of the average absolute error of both models across the different cells and cycles, presented for different percentiles. 54

Table 4.8: Average statistics of the absolute error inside a cycle, presented for different percentiles. 55

LIST OF FIGURES

Figure 1.1: Timeline of a fixed interval replacement strategy. Rf means replacement after failure and RP, preventive replacement. 7

Figure 1.2: Timeline of a fixed age replacement strategy. Again, Rf means replacement after failure and RP, preventive replacement. $t1$ and $t2$ represent aleatory lifetimes which are shorter than tp. 8

Figure 1.3: CBM Data. Adapted from (Ragab, Yacout, Ouali, & Osman, 2015) 10

Figure 1.4: P-F Interval (Blann, 2013) 11

Figure 2.1: Bias-Variance trade-off. The graph on the left presents an overfitted model and the graph on the center, an underfitted model. The graph on the right is the optimal model, with a good compromise between bias and variance. Adapted from (Seema Singh, 2018). 17

Figure 2.2: Toy NN with one hidden layer. Weights and biases are represented. 19

Figure 2.3: Unrolled RNN (Olah, 2015) 22

Figure 2.4: LSTM Cell (Olah, 2015) 23

Figure 2.5: Architecture of an autoencoder (Singh & Palod, 2018) 25

Figure 4.1: Evolution of the electrical current of the electrolyzer during a complete cycle. The startup phase is plotted in blue and the operation phase, in orange. Notice the length's difference between both phases. 37

Figure 4.2: Voltage during the startup phase of the same cell (Y) for three different cycles (A, B, C). 37

Figure 4.3: Voltage during the startup phase of three different cells (X, Y, Z) for the same cycle (A). 37

Figure 4.4: Architecture of the proposed model. Rows in the matrices represent the different features: V ≡ Cell's voltage, I ≡ Electrical current, T ≡ Temperature, X ≡ Caustic concentration. C_x and C_y are the two outputs of the encoder. V_{pred} is the cell's voltage outputted by the model. Columns in the matrices represent the number of timesteps. Initially, there are

always 720 startup timesteps but, after the masking layer, they differ for each cycle and are represented by the letter *n*. The operation timesteps are always four. 42

Figure 4.5: Encoder subnetwork. 44

Figure 4.6: Predictor subnetwork. 44

Figure 4.7: Example of how striding works. Each index number represents a window. For a stride of one, consecutive windows are picked for training. For a stride of four, only one observation every four is picked. 48

Figure 4.8: Steps taken by each model for making predictions. 50

Figure 4.9: Evolution of the encoding of cells X, Y, and Z along cycles A, B, and C. The axes of the graph do not have units, as they are the result of a dimensionality reduction. They take values in the interval [0,1] because the output of the encoder is a sigmoid function. 51

Figure 4.10: Predicted voltage for cells X, Y, and Z during cycle A. A solid line represents the real voltage of each cell, while a dotted line represents the predicted voltage. 52

Figure 4.11: Extract of cycle A. The upper subgraph shows the absolute error between the predicted and the measured voltage, averaged over Cells X, Y, and Z. The lower subgraph presents the electrical current. 53

Figure 4.12: Divergence between the predicted voltage and the measured voltage during the 48 hours preceding a cell's fault. 56

LIST OF SYMBOLS AND ABBREVIATIONS

AE	Absolute Error
API	Application Programming Interface
CBM	Condition-Based Maintenance
CPU	Central Processing Unit
DCS	Distributed Control System
DNN	Deep Neural Network
DTW	Dynamic Time Wrapping
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
ML	Machine Learning
NN	Neural Network
PID	Proportional-Integral-Derivative
PLC	Programmable Logic Controller
PM	Preventive Maintenance
RCM	Reliability Centered Maintenance
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SAE	Society of Automotive Engineers
SE	Squared Error
SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
TPM	Total Productive Management
VAE	Variational Autoencoder

CHAPTER 1 INTRODUCTION

Fault prediction is a crucial component for planning better maintenance strategies. Consequently, it is first needed to define the concept of maintenance. We define maintenance as the ensemble of techniques whose goal is to conserve systems and equipment working as designed. These systems must work reliably for as long as possible, with as little downtime as possible. Furthermore, it has to be achieved cost-efficiently.

The concept of maintenance evolved throughout history, especially during the last century. A historical recapitulation is presented, illustrating this evolution. We present how reliability in industry augmented progressively and how it influenced maintenance strategies. Next, we get into more details of the different maintenance strategies. We highlight Condition-Based Maintenance, which is the required base for implementing successful fault prediction techniques.

Among fault prediction techniques, we are interested in those that don't require the full stoppage of the system and so, can be implemented in real-time. This is a substantial advantage for industrial processes, as stopping and restarting them is a lengthy and costly process. We set two main design objectives: to improve the accuracy of currently used models (in order to predict the failures earlier) and to not rely on human expertise to fine-tune parameters once the model is deployed, easing the reuse of the model for multiple systems.

For achieving those objectives, we propose a model based on Artificial Neural Networks. This model follows an encoder-decoder architecture that characterizes the degradation of the system and predicts the expected response that it would produce in a healthy state. This expected response is then compared with the measured response in real-time and a maintenance alarm is triggered if a substantial divergence is found. The model is composed of LSTM layers, so, in the literature review, we introduce the theory behind them, as well as the principles of neural encoders.

In the article that axes this thesis, we test our approach in electrochemical electrolyzers. Electrolyzers are composed of multiple cells, where the unexpected failure of only one of them causes the full stoppage of the whole electrolyzer and might lead to fires or explosions. The voltage of each cell is related to their failure likelihood, so it is the variable monitored. However, the input of each cell of the electrolyzer is the same – being the voltage the only measurable variable that is different for each cell. We use the encoder of our proposed model to individually differentiate each

electrolyzer's cell (with its different degradation level and specific characteristics). The decoder uses the individual result of the encoder for each cell and the general input features of the electrolyzer and outputs the expected voltage for each cell. The divergence between this expected voltage and the measured voltage of the cell is monitored for discrepancies.

1.1 Historical Evolution of Maintenance

Equipment always degrades with use and time. Ever since the human being started building machinery and infrastructure, maintenance has been an essential factor in order to ensure that it operated as designed. In ancient cultures, maintenance operations were elementary, as most of them consisted of replacing the broken equipment and keeping it free of debris (Hill, 1996). However, the Roman Empire started to realize the importance of a properly working machinery and created a separated task in their army that was specialized in these matters (Landels, 2000).

Machinery complexity evolved with the centuries and different civilizations, but we must fast-forward to the industrial revolution of the 18th century to find the most significant change. Machines were now being used to build other machines, as humankind was moving away from hand manufacturing to machine-based manufacturing. Factories started to emerge, and production rates to increase at a pace never seen before. For the first years, however, this evolution in manufacturing techniques was not reflected in better maintenance techniques, as failure rates were very high for these increasingly complex systems. Demanding work environments and unskilled workers that pushed systems to the limit played a key role in these failures.

Maintenance acquired a leading role during the Great Wars of the 20th century. With the army relying more and more on machinery, there was a need to assure its correct functioning at all times, as failures during the battle could be fatal. Maintenance mechanics were responsible for these tasks, as they assessed if the machines were ready before the battle (Mueller-Hillebrand, 1988). Preventive Maintenance (PM) became the norm, scheduling the replacement of components of the machinery with the hope of doing it before it broke down completely. The first references to PM in non-war environments are also found during this period, for example, covering the necessity of changing oil filters in automotive engines (Nostrand & Moore, 1943).

At the same time, as technological development advanced, industrial plants grew in complexity. They were now formed by many systems that relied on each other to accomplish a mission. A failure in one of those systems might lead to a chain reaction and halt the whole plant. This resulted in a loss of revenue due to the operational downtime and, in a society more evolved than ever and worried about workers' conditions, a decrease in operational safety. The need for diminishing service disruptions continued to rose and, with it, the development of new maintenance techniques.

It was during the second half of the 20th century that maintenance planning started to be considered during the design of the system. Total Productive Maintenance (TPM) (Nakajima, 1988) was developed in Japan for managing the maintenance strategy of the automotive industry, specifically at Toyota. It implements PM along the whole production cycle in order to minimize losses. It shares many principles with what would later be known as *Lean Manufacturing* (Holweg, 2007) and which summarized the Japanese manufacturing culture that made them quality leaders during the 1970s and 1980s.

Another essential production paradigm to remark is Reliability Centered Maintenance (RCM) (Nowlan & Heap, 1978). It was developed in the United States by United Airlines with the goal of finding the optimal strategy for aircraft maintenance. It was developed almost simultaneous in time to TPM, but both techniques differ significantly in their implementation and their primary objective. On the one hand, TPM is based on applying a strict PM schedule in order to minimize downtime and costs. On the other hand, RCM revolves around understanding the types of failure, learning to manage them, and improving the system's design to overcome them. Eventually, just like TPM expanded to other domains, RCM was adopted by other industries, especially those in where a failure was most critical – like mining and oil and gas drillings. An SAE standard was defined for guiding its implementation in organizations (JA1011, 1999). Both TPM and RCM can even be integrated in order to increase reliability and reduce downtime (Ben-Daya, 2000).

Through multiples studies on planes' degradation, one of the main breakthroughs that RCM brought was that, contrary to industry beliefs, most of the failures were not related to the age of the equipment. This discovery meant that, in order to maintain a safe and reliable process, it was mandatory to understand how each system and component could fail, and what were the symptoms leading to their failure. As resources were limited, it was needed to prioritize those systems that caused a more prominent disruption when they failed. With all this in mind, it became clear that a

maintenance strategy based only on PM was not enough, opening the door to a new technique: Condition-Based Maintenance (CBM). The idea behind CBM is to anticipate the failures before they happen. In CBM, the critical factor is not the age of the piece. Instead, a measurable indicator of performance is used to carry out the maintenance actions only when it is required.

In parallel to those new paradigms in maintenance planning, the second half of the 20th century also saw the disruption of machine computing. It is not possible to understand current maintenance strategies without first taking a look at how industrial processes were controlled. Up to now, control systems in an industrial plant ranged from primitive hardwired relays and actuators to more complex analog PID controllers. They were connected to analog sensors and configured to execute specific actions when predefined input conditions were met. They were also used to keep a controlled system variable between defined limits employing a particular control loop (Willis & Tham, 1994). However, they had two main limitations. Firstly, every change in the control logic had to be hardwired for each controller, so a little change required considerable downtime to implement. Secondly, even if all these control systems were physically located in the same room, each system was independent of each other. This meant that the plant's operators had to walk around the control room all the time to read the measurements in analog gauges and interact with each system using switches and levers.

These limitations were resolved with the advent of digital minicomputers. Minicomputers were developed in the 1960s and rapidly started gaining industry adoption. Their much lower cost and space requirements were significant advantages against the mainframe systems previously used in other applications (Stout & Williams, 1995). They allowed to implement the control loops dynamically via software instead of relying on individual hardwired controllers. This new method was called Direct Digital Control and enabled the centralization of all the measurements and controls under the same digital control panel and terminal (Cutler, 1995). Plant's operators could now focus their attention on a single dashboard to see a global picture of what was happening in the plant, and so, have a faster reaction in case that problems arose. This provided a considerable advantage in terms of reliability. Indeed, it helped minimize human errors in the operation of the system, which were common with the analog systems used before.

As the computer industry continued to evolve during the 1980s, microcomputers replaced minicomputers (Fosdick, 1980). They were simpler devices commanded by a microcontroller,

orders of magnitude cheaper than the minicomputers used until then. Along came the development of the Programmable Logic Controller (PLC), which is a type of microcomputer specifically designed to receive multiple inputs and execute control actions in response (Rohner, 1996).

PLCs were a breakthrough in plant control because, with the advancement in network connections, new distributed control architectures could be developed. Among them, Distributed Control System (DCS) was the most important. In a DCS architecture, each PLC controls a different part of the plant, and they are all hierarchically grouped in a modular and fault-tolerant way (Bhullar, 1993). The objective of such an architecture is to improve the plant's reliability by eliminating single points of failure, so the malfunction of a PLC would not halt the whole process (US Patent No. 4,517,637, 1985). It also speeds up maintenance actions, as the controllers may be replaced without stopping the production line. All of these advancements resulted in a highly optimized industrial environment, where systems were tightly controlled by PLCs and monitored and orchestrated by the DCS of the plant. It all led to better control of operating conditions, less stress to the systems, better reliability, and so, less maintenance needed.

Nevertheless, the impact of computers from a maintenance standpoint did not end on increasing the reliability of the plants' control systems. Seeing the opportunities of a new market, many companies developed applications to leverage the capabilities of computers. Computerized Maintenance Management System, for example, helped to track the physical assets of the plant by centralizing all the information of the plant's maintenance operations (Wireman, 1994). In other words, they were the digital evolution of noting by hand the maintenance history of each system. This software made it possible to keep a history of what had been done at each inspection, so it was easy to keep track of the changes. It was especially useful for following PM strategies, as it facilitated to schedule the following maintenance services and to note the lifetime of each asset at failure.

Indeed, industries changed significantly after the arrival of computers. This digitalization of the industry was implemented with the idea of increasing process reliability and production rates. As a side gain, the pieces needed for implementing CBM strategies were starting to be in place. So far, measured variables were related to the plant's control, as there was no incentive to measure by-product variables that could not be controlled. However, as the computing power had significantly increased, it was now possible to process multiple data streams simultaneously. As a

result, the use of sensors in the industry skyrocketed. Some researchers started to find new uses for this new data, and so, CBM continued to gain traction (Tsang, 1995).

CBM is still a very active field of research nowadays. Advances in storage techniques during the first decade of the new century made it possible to store progressively more sensors' information. This massive amount of data, combined with new techniques for processing it, has brought the interest back to finding new techniques for improving maintenance planning. Among them are *fault prediction* techniques, of which a new one is presented in this work.

1.2 Maintenance Strategies

There are multiple ways of planning maintenance operations, but we can highlight three main strategies: *Corrective Maintenance*, *Preventive Maintenance,* and *Condition-Based Maintenance*.

Corrective maintenance is a passive strategy, as no action is taken to maintain the equipment during its lifetime, only when it breaks. On the contrary, Preventive Maintenance and Condition-Based Maintenance are proactive strategies, both aiming to replace the equipment before it fails. Nonetheless, both techniques are not to be confused despite the similarities in their names, as they approach the problem in two very different ways.

1.2.1 Corrective Maintenance

Corrective maintenance is the simplest type of maintenance. It consists in fixing or replacing the equipment after it fails. It is the *de facto* maintenance strategy when no maintenance is planned. It is used for non-critical equipment, where a failure does not result in a disruption of service or a safety concern.

1.2.2 Preventive Maintenance

Preventive Maintenance (PM), also known as Preventative Maintenance, is the most extended strategy. Maintenance actions are carried out periodically based on a simple criterion of the system: its age, number of operation cycles, kilometers traveled, etc. When the value of the chosen criterion surpasses a defined limit, a maintenance operation is performed. This limit is defined based on a statistical analysis of previous systems' lifetime. By scheduling the maintenance actions, it is

possible to, with a certain degree of confidence, replace the system before a failure happens. Thus, minimizing the appearance of unexpected failures.

Manufacturers give lifetime estimations of their products, so the simplest PM technique is to replace the product before that time arrives. An example of this is when people replace every certain number of kilometers the oil of their car's engine.

In order to choose this strategy over simply performing corrective maintenance after failure, the following suppositions must apply:

- The cost of replacement after failure (C_f) is more expensive than the cost of preventive replacement (C_p). Both costs are known at the time of planning the maintenance strategy.
- The downtime caused by a failure (T_f) is longer than the downtime caused by a preventive replacement (T_p). Both downtimes are known when planning the maintenance strategy.

There are two ways of carrying out PM on schedule: replacement at *Fixed Interval* or replacement at *Fixed Age*.

1.2.2.1 Fixed interval replacement

Equipment is replaced preventively at fixed intervals. An interval, or cycle, is the time between two consecutive preventive replacements, independently of the number of failures that might occur inside the cycle. The unit of this cycle can be specified in kilometers traveled, data written to disk, time passed, etc. It is an iterative process, and it is represented in Figure 1.1. t_p is the length of the cycle and the variable to optimize. The goal is to find the optimal t_p that minimizes the operation costs or maximize the availability of the system.

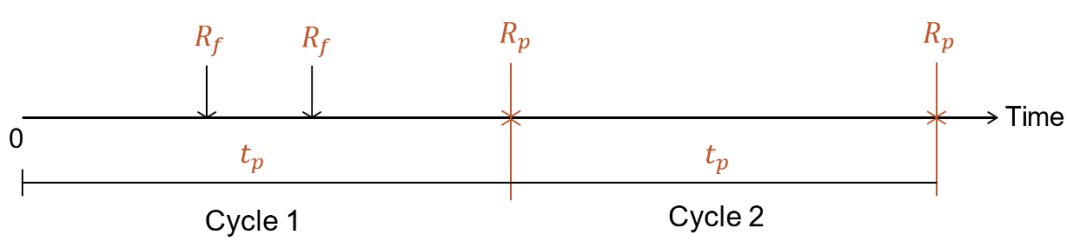

Figure 1.1: Timeline of a fixed interval replacement strategy. R_f means replacement after failure and R_P, preventive replacement.

It is straightforward to implement, as it does not require to modify the maintenance planning after a system's failure. For this reason, the maintenance actions can be scheduled months in advance. This replacement strategy is typically used when the cost of the system is inferior to the labor costs.

1.2.2.2 Fixed age replacement

Equipment is preventively replaced when it reaches a certain age (or another indicator, age is just used as an example for simplicity). The decision to intervene is independent of the nature of the previous replacement, be it corrective after a failure or preventive. In this case, t_p is the age of the system at which the maintenance is performed, which is always the same. The timeline of this strategy is represented in Figure 1.2. In contrast to fixed interval replacement, there is only one intervention per cycle.

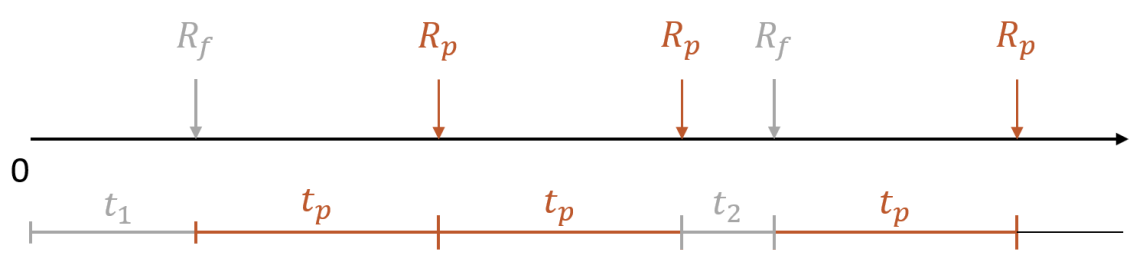

Figure 1.2: Timeline of a fixed age replacement strategy. Again, R_f means replacement after failure and R_P, preventive replacement. t_1 and t_2 represent aleatory lifetimes which are shorter than t_p.

It is slightly more involved to implement than the fixed interval strategy, as it requires to keep a register with the age of the equipment. It also requires rescheduling the following PM action after every corrective or preventive replacement. This strategy is used when the equipment is more expensive than the labor.

1.2.3 Condition-Based Maintenance

Ideally, equipment would be replaced or fixed at the optimal time, and there would be no need to perform any action if it is still working correctly. However, PM presents a major drawback that

makes this goal unattainable, as it just considers observations from past pieces of equipment and do not actively monitor the state of the equipment that is currently being used. Indeed, PM is based on the supposition that similar systems have similar life expectancies. However, in a real-life production environment, this supposition does not hold. Differences in manufacturing and operating conditions along their lifetime, influence the specific life expectancy of each system.

Condition-Based Maintenance (CBM) overcomes that supposition and, as such, it is the next evolution in maintenance planning. CBM is the technique of monitoring over time the state of the system and taking maintenance decisions based on it.

Continuing with the car's engine example, we would measure the viscosity of the oil every certain number of kilometers. When this measured viscosity is superior to a defined threshold (given by the manufacturer), the system becomes more likely to fail. At that point, it must be replaced in order to avoid an unexpected engine failure.

It is a more precise approach than PM, but it comes at the expense of a more complex implementation. In CBM, the system is monitored from commissioning until failure. This is a substantial advantage compared to PM, where only time-of-failure data was registered.

In Figure 1.3, it is possible to observe the difference between both types of data. Each path represents the degradation of a different system over its lifetime. PM only uses the points at the failure threshold. A probability density function is fitted to them, resulting in the black curve, which is then used to make statistical assumptions. Instead, CBM uses the complete degradation path, making decisions based on the degradation process that leads to the system's failure.

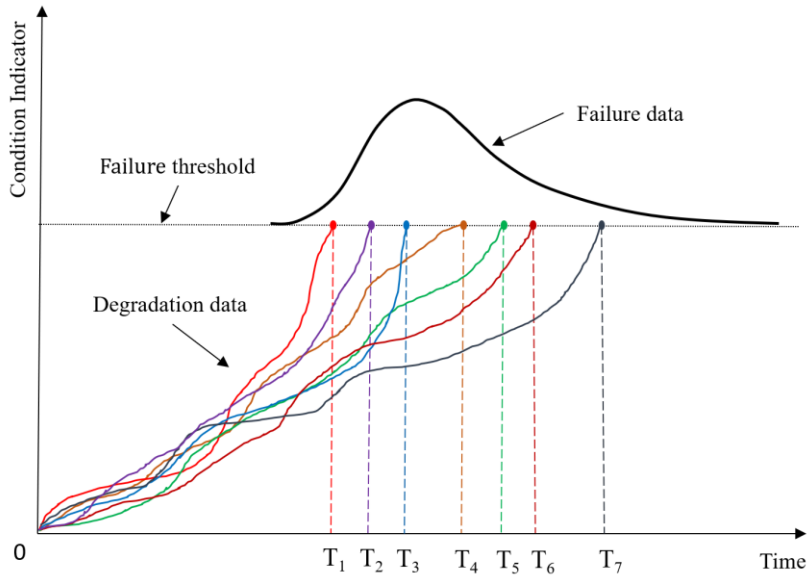

Figure 1.3: CBM Data. Adapted from (Ragab, Yacout, Ouali, & Osman, 2015)

We can see that each system has a different degradation path. However, we can also see that all the paths share some similarities. This is expected because the base degradation process is similar for every system, except for specific differences due to their manufacturing and operating conditions.

Having this data, we can estimate the expected system's lifetime more precisely. Now, we do not need to rely on the average lifetime of past systems. Instead, we consider only the lifetime of those past systems whose degradation path resembles the most to our system's path.

It is possible to find the resemblance between the systems by performing a similarity analysis using, for example, Dynamic Time Wrapping (DTW) (Berndt & Clifford, 1994). We calculate a similarity index between the path of our system and each one of the previous systems. The system with the highest index is the one associated. This calculation is repeated for each new measure that we take of our system.

Another approach is to use clustering. Clustering consists in making different groups and placing into each group the systems that are more similar between them. By definition, the systems placed in a group are closer between them than to those placed into the other groups. K-means is a clustering algorithm used for this task (Peng, Zhou, Hepburn, Judd, & Siew, 2013).

1.2.3.1 Fault Prediction

Fault prediction englobes a set of techniques used to perform CBM. Please note that in this thesis, we refer to fault and failure indistinctively. It relies on the fact that the condition of every system degrades over time with use and that, at some point, this condition becomes apparent and can be detected before the system fails. The lapse of time between the detection of the impending failure and the failure *per se* is known as *P-F Interval*. If the system's condition is in this interval, maintenance actions must be performed.

Two points delimit the P-F interval. The point *P* is the earliest moment at which the impending failure can be predicted. Its position on the degradation curve depends on the technique used for assessing the condition of the system. More precise techniques move the point P to the left, permitting to predict the failure earlier and giving the operator more time to perform the maintenance actions before the system fails. The point *F* represents the moment at which the failure occurs. It is independent of the point P, as it only depends on the specificity of each system and its operating conditions during its lifetime. Depending on the chosen technique, the possible locations of the point P on the curve are shown in Figure 1.4.

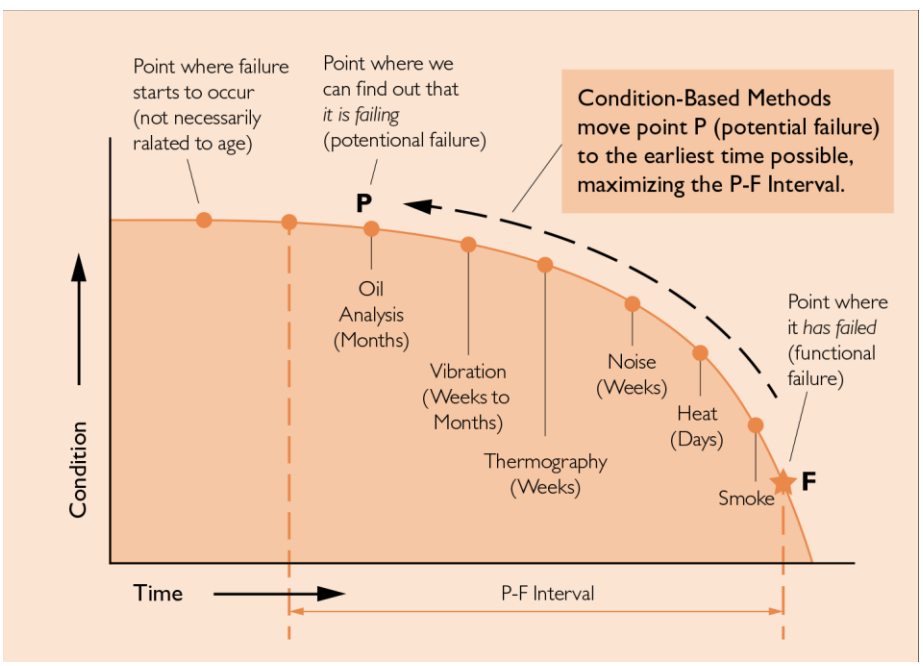

Figure 1.4: P-F Interval (Blann, 2013)

In order to predict an impending failure with enough time to intervene, the optimal strategy is a trade-off between the *choice of measurement technique* and the *frequency of measurements.* Indeed, if the measurements are performed manually, they must be carried out at time intervals shorter than the P-F interval. This problem disappears if the measurements can be carried out automatically by utilizing sensors. However, not every variable can be continuously monitored with sensors, requiring instead the use of complex manual inspections techniques. These inspection techniques are usually more precise but also more costly and involved to implement, usually requiring offline or even destructive tests.

The optimal outcome is then to develop methods capable of predicting faults using system's features that can be continually monitored with sensors. Thus, avoiding the need for carrying out thorough and costly inspections.

With that said, if the relationship between a monitorable feature and the failure of the system follows a monotonic trend, it is possible to use statistical control charts to define the failure threshold. It is also possible to use an expert system. In this case, instead of applying statistically significant thresholds to the expected range of values, they are defined based on expert knowledge on the matter (Mazurkiewicz, 2015).

However, some systems do not have any measurable feature that presents such a relationship. In these cases, the likelihood of failure must be inferred from the combination of multiple measurable properties of the system. Thus, more complex techniques must be used.

The first technique is to train a classifier with observations of past healthy systems. When using this technique, observations from the new system are given to the classifier, which determines if it is working correctly based on the learned data. This can be achieved by using classifying algorithms like Support Vector Machines (SVM) (Fernández-Francos, Martínez-Rego, Fontenla-Romero, & Alonso-Betanzos, 2013). The problem with this approach is that it is not time-aware, so the predictions may oscillate between healthy and faulty periodically, instead of following a trend.

Another possible technique is to detect sudden changes between two consecutive observations, which can be implemented by using Kalman-filters (Wei, Verhaegen, & van Engelen, 2010). The problem is that, if the change is sudden and significant, it is probably because the system is about to fail. Thus, there is not enough time for performing the required maintenance actions. Usually,

failures start to develop slowly, as previously seen in the P-F graph, so ideally, we want to detect them at their first stages.

In order to detect the failures earlier, there is another approach. Even if no directly measurable feature is monotonically related to the failure of the system, there are usually certain measurable features that respond differently to operating conditions when a fault is about to happen. Thus, if we are able to estimate the value that a healthy system would present for such features, we can compare it to our system's measured value (US Patent No. 7,818,276, 2010).

The difference between both is a new variable that is indeed monotonically related to the system's failure. This is because, when there is an impending failure, the estimated value remains constant, and the measured value increases or decreases. Thus, the absolute difference between the two increases steadily. We can then put limits to this difference, by using either a control chart or an expert system, as previously explained.

In order to predict the failure with enough anticipation, the estimation of the variable must be accurate. Indeed, the more precise is the technique used, the lower the detection threshold can be set. In this work, we develop a new method for performing this kind of fault prediction.

1.3 Research Objective

In the previous sections, we have introduced how maintenance operations have evolved over time and presented the different maintenance strategies that are used nowadays. Condition-Based Maintenance presents many advantages over Preventive and Corrective Maintenance, at the expense of requiring data from the system's operation during its lifetime. This is not a problem anymore as, with the progression of Industry 4.0 (Lasi, Fettke, Kemper, Feld, & Hoffmann, 2014), it is increasingly common to have multiple sensors installed in production plants. In this work, we take advantage of this data to predict possible faults in the equipment before they arise.

The last technique presented in section 1.2.3.1 describes a way of performing fault prediction by measuring the divergence between a monitorable system's value and the estimated value that a healthy system would present. This estimation is usually obtained by using parametric methods, which are based on simplifications and assumptions about the inner working of the system.

The objective of this work is to develop a new method that improves the accuracy of this estimation. At the same time, we do not want to rely on domain knowledge for the development of such a model. In order to do so, we propose to replace the parametric models with a deep learning model. The model is trained with operation data from previous systems, instead of needing a team of domain experts to model the process.

We seek a model that accomplishes these requirements because it eases its deployment to different electrolyzers and plants. Also, a better estimation accuracy permits us to use a stricter fault threshold without increasing the risk of false positives, which in turn allows us to predict an oncoming fault earlier. By predicting the fault earlier, plant operators have more time to plan the maintenance actions accordingly. This leads to a safety increase and to a better distribution of the available resources, reducing the associated maintenance costs.

CHAPTER 2 LITERATURE REVIEW

Fault prediction is the intersection between maintenance and forecasting. In the previous chapter, we introduced the different maintenance strategies available. In this one, we focus on the forecasting task. In particular, and according to our research objective, we center our review on Neural Networks (NNs) and the theory behind them. However, as NNs are Machine Learning (ML) algorithms, we start by making a brief introduction to ML.

ML is based on statistical techniques and computing algorithms whose goal is to make a computer learn to do a certain task, without being explicitly programmed to do it. These methods effectively learn from experience, as they find insight in data that is given to them. They are divided into three families: *Supervised Learning* (Hastie, Robert, & Friedman, 2009), *Unsupervised Learning* (Celebi & Aydin, 2016), and *Reinforcement Learning* (Sutton & Barto, 2018). Each family is appropriate for different objectives and data.

Our problem is appropriate for supervised learning algorithms, as we are interested in inferring the value of a certain feature from other features, which is known as *regression*. Therefore, we are going to focus on this family of algorithms. We also introduce the different types of data available, highlighting *multivariate time series*, which is the type of data most commonly found in industry.

2.1 Supervised Machine Learning

These algorithms map an input vector X to a dependent output vector y. They learn to infer the relationship between them so, after training, when only the vector X is given, they are capable of inferring the corresponding vector y.

Arguably, the simplest supervised ML algorithm is a linear regression. A linear regression is defined by the equation $y = a * x + b$, where a is the slope, and b is the intercept. The regression model must find the optimal value for both. There are many more algorithms, like Decision Trees, Support Vector Machines, Neural Networks, etc. A review of different algorithms with practical examples is presented in *An Introduction to Statistical Learning* (James, Witten, Hastie, & Tibshirani, 2014).

Stored data is organized in *datasets,* and datasets themselves are composed of *observations*. Data given to an ML model for learning receives the name of *training* dataset. Data used to make predictions after training receives the name of *testing* dataset. The training and testing datasets are a subset of the whole data available.

2.1.1 Bias-Variance trade-off

The bias-variance trade-off is introduced in order to explain the difficulty of reaching a good prediction accuracy and the importance of choosing an appropriate dataset and algorithm for the task at hand. The prediction error of a supervised ML model is divided into three. Two errors are minimizable, and another one is not, which is called the irreducible error and it is the result of noise in the data. Among the minimizable errors, it is possible to differentiate between the error due to *bias* and the one caused by *variance*. The trade-off comes from the difficulty of minimizing both errors at the same time.

- Bias error: due to a model that is not accurate enough to find relationships in the data. In this case, it is said that the model *underfits* the data. Using more complex models usually helps to minimize it.
- Variance error: due to differences between the training dataset and the testing dataset. The model approximates the training dataset in excess, modeling the noise present in the dataset instead of the actual underlying trend. This leads to a model that is not capable of generalizing its results from the training to the testing dataset. This behavior is referred to as *overfitting*. Using more variated data or, when that is not possible, a less complex model, helps to minimize it. It is also possible to add *regularization* to the model (Cucker & Smale, 2002).

In order to find a good balance between both errors, the model and the training dataset must be chosen carefully. Figure 2.1 represents the result of this trade-off on three different models with the same data. This complicates the development of ML models, as many iterations are usually required until finding a good compromise.

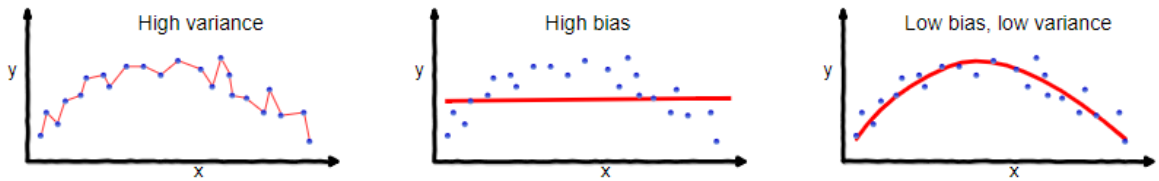

Figure 2.1: Bias-Variance trade-off. The graph on the left presents an overfitted model and the graph on the center, an underfitted model. The graph on the right is the optimal model, with a good compromise between bias and variance. Adapted from (Seema Singh, 2018).

2.1.2 Data Types

Due to the bias-variance trade-off, data used for training must be representative of the expected conditions of the testing dataset.

Depending on the shape of each observation, we can distinguish between:

- Univariate data: each observation is composed of a single numerical value.
- Multivariate data: each observation's value receives the name *feature*. When we refer to multivariate data, we are usually referring to a vector of values. However, it can also be a tensor of values, which is a matrix of any dimension. An image, for example, is a tensor of pixel values. If it is grey-scaled, its shape is defined by (height, width). However, if the image is colored, its shape is (3, height, width).

Depending on the interaction between observations, we can differentiate between:

- Still data: the order of the observations is not important for the predicted outcome. Observations can belong to the same system or to different systems.
- Temporal data: the order at which the observations were registered is significant. In other words, the temporal difference between observations is important. Following the previous example of the colored image, a colored video is composed of an ordered succession of colored images. Thus, the shape of each one of its frames is: (frame number, 3, height, width).

In industrial environments, the most common data registered are sensors' measurements. Data coming from sensors is ordered, as it is registered sequentially. If the sensor only measures a single feature, we call its data a *univariate time series*. If a single sensor is responsible for monitoring multiple features, or if we join measurements coming from multiple sensors, the resulting data is known as a *multivariate time series* instead.

An approach to make predictions for time series is to use *Autoregressive Models*. These models use the value of the dependent variable at time-steps $t-1, t-2, \ldots$ to infer its value at the actual time-step t (Fuller, 2009). Mathematically, their interaction is defined as $y_t \sim y_{t-1}, y_{t-2}, y_{t-3}, \ldots$ They are mainly used with univariate time series, but covariates can be included in the equation to use them with multivariate time series (Penny & Harrison, 2007). In our article, we use LSTM neural networks for this task, as they present certain advantages that are described.

2.2 Neural Networks

Neural Networks are ML models formed by multiple mathematical entities called *neurons*. Neurons are distributed in *layers*. They always have at least two layers, the *input layer,* and the *output layer.* If more layers are added, they are called *hidden layers*. The term *Deep Learning* englobes NNs with multiple hidden layers, as including more hidden layers makes the NN *deep* (Goodfellow, Bengio, & Courville, Deep Learning, 2016).

In the most basic architecture, called a *feedforward neural network*, each neuron of a layer is connected to all the previous layer's neurons. These connections are the *weights*. The value of a neuron is the sum of the values of all the neurons of the previous layer, each one multiplied by their corresponding weight. To this weighted sum, we add a new constant term, called the *bias*. In turn, this makes each neuron act like a linear regressor, with an equation given by $y = w * X + b$, where y is the output of the neuron, X the vector of inputs, w the matrix of weights and b the bias. Biases are usually initialized with zeroes, but the initialization of the weights is more involved. Different initializations have been proposed, as the NN may not converge if its initial weights are not correctly set (Glorot & Bengio, 2010). Once the initial weights and biases are set, it is time to train the NN. Training the NN is an iterative process whose objective is to find a suitable set of parameters – weights and biases – such that $y = f(X, parameters)$.

So far, the output of the NN is just a linear combination. The capability of approximating non-linear functions comes from the activation function $g(x)$ that is added to the output of each neuron. This function is a non-linear transformation, being the most used the Sigmoid (Han & Moraga, 1995) and the Rectified Linear Unit (ReLU) (Nair & Hinton, 2010). Thus, the output of a neuron after being non-linearly transformed is: $a = g(y) = g(w * x + b)$. Equation (2.1) presents the general equation for the output of a layer l, where $l = 1, .., L$ and L is the total number of layers in the NN. w_l represents the matrix of weights between layers $l - 1$ and l, and b_l is the bias vector for layer l.

$$a_l = a_l(a_{(l-1)} * w_l + b_l) \tag{2.1}$$

The only layer that does not have an activation function is the input layer, as it is directly the input vector, so $h_{l=1} = X$ (Nielsen, 2015). When using non-linear activations and enough neurons plus data, NNs are considered to be universal approximators (Pinkus, 1999). Figure 2.2 represents an example of a basic NN with its weights and biases.

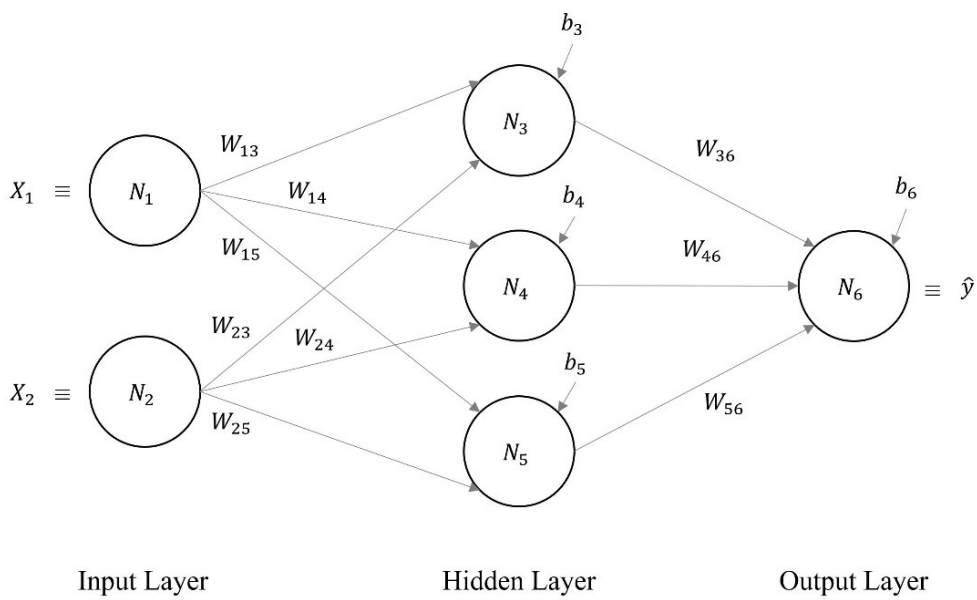

Figure 2.2: Toy NN with one hidden layer. Weights and biases are represented.

Each iteration of the training procedure is composed of a *forward pass* and a *backward pass*.

The forward pass is the process of calculating the output of the NN. It takes an input vector, which is successively transformed by all the neurons in the NN until the last layer is reached. Then, the difference between the output of the NN ($\hat{y}$), and the expected value (y) is calculated according to a *loss function.* For regression, the most common loss functions are the Absolute Error (AE), also known as *L1 loss* and the Squared Error (SE), also known as *L2 loss*. The average of the loss function across all the observations of the dataset is the *cost function,* and it is the function that must be minimized to optimize the NN.

Following Equation (2.1), and for an input vector $[X_1, X_2]$, the output $\hat{y}$ of the toy NN introduced in Figure 2.2 is given by Equation (2.2). In this example the output of the NN is linear, or in other words, the neuron N_6 doesn't use an activation function. The SE loss with respect to the expected output y is calculated in Equation (2.3).

$$a_{N_3} = g(y_{N_3}) = g(X_1 * w_{13} + X_2 * w_{23} + b_3) \tag{2.2.1}$$

$$a_{N_4} = g(y_{N_4}) = g(X_1 * w_{14} + X_2 * w_{24} + b_4) \tag{2.2.2}$$

$$a_{N_5} = g(y_{N_5}) = g(X_1 * w_{15} + X_2 * w_{25} + b_5) \tag{2.2.3}$$

$$\hat{y} = a_{N_6} = y_{N_6} = a_{N_3} * w_{36} + a_{N_4} * w_{46} + a_{N_5} * w_{56} + b_6 \tag{2.2}$$

$$SE\ Loss = (y - \hat{y})^2 \tag{2.3}$$

After the loss is calculated, the parameters of the NN must be modified to minimize such loss. This is done by an optimizer function, following the concept of *gradient descent*. It is based on calculating the derivative of the loss with respect to each parameter of the model – weights and biases. The value of each parameter is reduced if its corresponding gradient is positive, and it is increased if the gradient is negative. The idea is that, in order to converge to the global minima of the cost function, each parameter is modified according to its influence on the network's loss.

In order to update the parameters smoothly, the gradient is multiplied by a scalar known as the *learning rate*. Choosing the optimal learning rate is still an active area of research. Multiple modifications adding an adaptive learning rate to the original Stochastic Gradient Descent (SGD) have been proposed. In fact, the Adam optimizer has a different adaptive learning rate for each parameter, and it is one of the optimizers most commonly used today (Kingma & Ba, 2014).

For calculating the gradient of each parameter with respect to the loss in a computationally efficient manner, the *backpropagation* algorithm is used (Rumelhart, Hinton, & Williams, 1985). The gradients are first calculated for the output layer, and then they are propagated to the previous layers by using the *Chain Rule* (Encyclopedia of Mathematics, 2013). We do not enter into details of the calculation of backpropagation, as they are symbolically involved and out of the scope of this thesis. However, more details can be found in *Neural Networks and Deep Learning* (Nielsen, 2015), if the reader so wishes.

The general feedforward NN architecture, composed of multiple layers, is also known as *Multi-Layer Perceptron* (MLP). There are many architectures derived from it, each of them specifically tailored to work with a different kind of input data and objective. We are interested in those architectures developed to work with multivariate time series, like Recurrent Neural Networks.

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a special family of neural networks which deals with temporal relations in the input data. Indeed, the output of an RNN for a certain time-step depends not only on the input of that time-step but also on the inputs of all the previous time-steps. They achieve so by keeping an *internal state*, or *recursive loop*, that is updated for every time-step of the sequence. This internal state serves the purpose of a memory, allowing past information to persist in the network and pass from one time-step to the next (Karpathy, 2015).

By unrolling these recursive loops over the temporal axis, an RNN results in multiple equal feedforward NNs, one per time-step. Each of these feedforward networks is connected to each other and shares the same weights. Figure 2.3 shows a diagram of an RNN with the recursive loop in the left and its unrolled equivalent on the right.

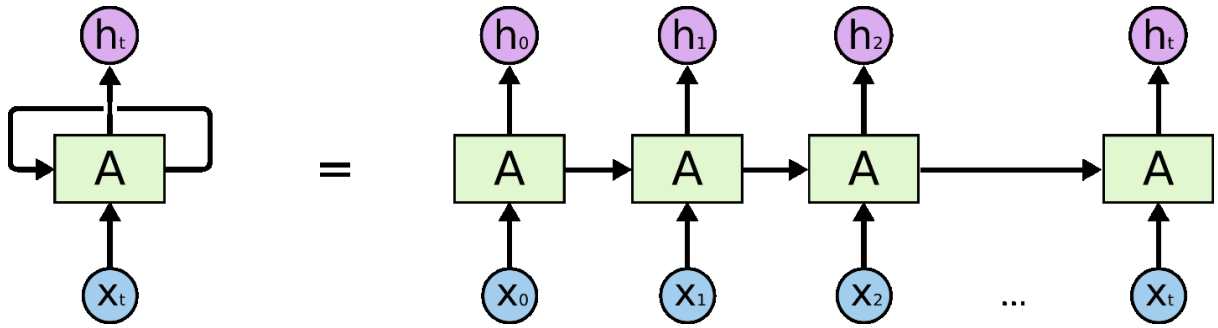

Figure 2.3: Unrolled RNN (Olah, 2015)

Thus, it is possible to train RNNs like feedforward NNs by using gradient descent. Their derivatives are calculated for each time-step individually and then added up before updating their weights. This process is called *Backpropagation Through Time* (Werbos & others, 1990). However, RNNs suffer from a fundamental problem that impedes them to obtain accurate results with long sequences. As more time-steps are added, the unrolled RNN becomes longer and more complex to train. This is because the calculated gradients are successively multiplied at each time-step, and so, they end up saturating, which is known as the exploding/vanishing gradients problem. The *exploding gradients* problem can be solved by clipping the gradients, for example, but the *vanishing gradients* are more complex to deal with (Bengio, Simard, & Frasconi, 1994).

The output of an RNN layer for a certain time-step t is equal to its internal state, and it is noted as h_l^t, where $t = 1,..,T$ and T is the total number of time-steps in the sequence. The general equation of an RNN layer can be inferred from Equation (2.1) by adding the temporal context. The result is Equation (2.4), where u_l represents the recursive weights matrix, and w_l represents the weight matrix between layers $l-1$ and l. Both are common for all the time-steps, respecting the constraints $w_l = w_l^{t=1} = \cdots = w_l^{t=T}$ and $u_l = u_l^{t=1} = \cdots = u_l^{t=T}$.

$$h_l^t = g_l(h_{(l-1)}^t * w_l + h_l^{(t-1)} * u_l + b_l) \tag{2.4}$$

Notice that the first layer of the RNN block is the input vector, thus $h_{l=1}^t = X^t$. The first internal state is defined as $h_l^{t=1}$ and it is initialized with zeros. The output of a block of RNN layers is the output of the last time-step of the last layer, equal to $a = h_{l=L}^{t=T}$.

RNNs are superseded by Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks. These two types of NNs control the internal state of each cell, overcoming the vanishing gradient problem and being able to deal with longer sequences. In the following section, we detail LSTM networks, as we use them in our article.

2.2.1.1 Long Short-Term Memory Networks

The internal design of an LSTM cell is very different from the one of an RNN cell. As previously mentioned, the hidden state of RNN cells is simply the output of the previous time-step. LSTMs, on the other hand, separate the hidden state from the output of the previous time-step. The hidden state is represented as C_l^t and the output of the time-step as h_l^t. Thus, the internal state can remain unaltered over multiple time-steps, avoiding the vanishing gradient problem (Hochreiter & Schmidhuber, 1997).

LSTM cells present three *gates* to control which information is kept and which one is discarded from its *hidden state* at each time-step. These gates are called *forget*, *update,* and *output*. They have learnable weights, so the network learns not only the output of the cell but also the interaction with its hidden state. These weights are common for all the time-steps of the same layer. Figure 2.4 presents a diagram of an LSTM cell.

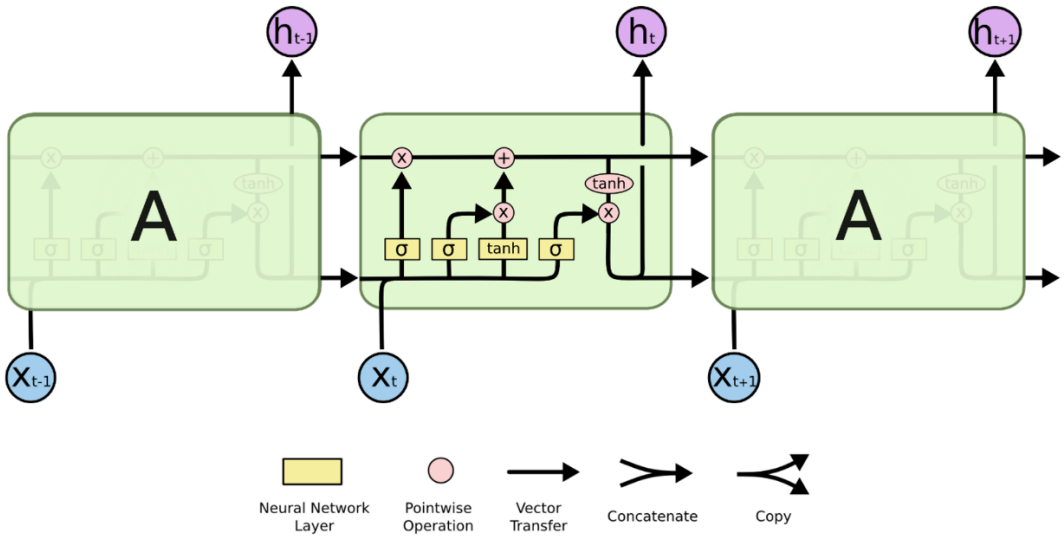

Figure 2.4: LSTM Cell (Olah, 2015)

The first step inside the cell is to concatenate the output of the previous time-step and the input of the current time-step. In order to lighten the notation, we define $input_l^t = concat(h_l^{(t-1)}, h_{(l-1)}^t)$. For the first layer of the LSTM block, $input_{l=1}^t = concat(h_{l=1}^{(t-1)}, x^t)$.

From left to right in the cell pictured in Figure 2.4, the forget gate is the first yellow block. It is responsible for choosing which information contained in the hidden state is still kept, and it is defined in Equation (2.5). The update gate decides which new information is added to the hidden state, defined by Equation (2.6). It is formed by two layers, the two yellow boxes in the middle of Figure 2.4. Finally, the output gate chooses the part of the current input that should be transferred. It is the rightmost yellow block in Figure 2.4, and it is defined in Equation (2.7).

$$f_l^t = sigmoid(input_l^t * w_{f_l} + b_{f_l}) \tag{2.5}$$

$$u_l^t = sigmoid\left(input_l^t * w_{i_l} + b_{i_l}\right) * \ tanh(input_l^t * w_{c_l} + b_{c_l}) \tag{2.6}$$

$$o_l^t = \ sigmoid(input_l^t * w_{o_l} + b_{o_l}) \tag{2.7}$$

The hidden state from the previous time-step ($C_l^{(t-1)}$) is modified according to the forget and update gates, as per Equation (2.8). Finally, considering the updated hidden state and the result of the output gate, the output of the time-step is calculated in Equation (2.9). Both this output (h_l^t) and the hidden state (C_l^t) are passed to the next time-step.

$$C_l^t = (C_l^{(t-1)} * f_l^t) + (u_l^t) \tag{2.8}$$

$$h_l^t = o_l^t * \ tanh(C_l^t) \tag{2.9}$$

The output of a block of LSTM layers is the output of the last time-step of the last layer, equal to $a = h_{l=L}^{t=T}$. Unless a *stateful* LSTM is used, the hidden state of the last time-step ($C_{l=L}^{t=T}$) is discarded (Remy, 2016).

This flexibility has made LSTMs become the *de-facto* neural networks used for dealing with time sequences in the last few years. We use them in our article as well to overcome certain limitations concerning the available data. Indeed, in our article, we use them as hidden layers of our encoder-decoder architecture. This architecture is derived from the general Autoencoder – presented in the next section.

2.2.2 Autoencoders

An autoencoder is a kind of neural network whose objective is to learn to keep only the critical information in an input vector (Kramer, 1991). It achieves this task by compressing the initial input vector into a reduced one called the *code* – which is a latent space representation – and then reconstructing the initial input vector from that code. It is composed of two subnetworks: an encoder and a decoder. The *encoder* is responsible for shrinking the original input vector into the code vector. The *decoder* subnetwork is responsible for reconstructing the code back to the original input vector. Both subnetworks are trained by minimizing the reconstruction error, i.e., the difference between the initial input and the output of the decoder (Hinton & Zemel, 1994). The structure of an autoencoder is shown in Figure 2.5.

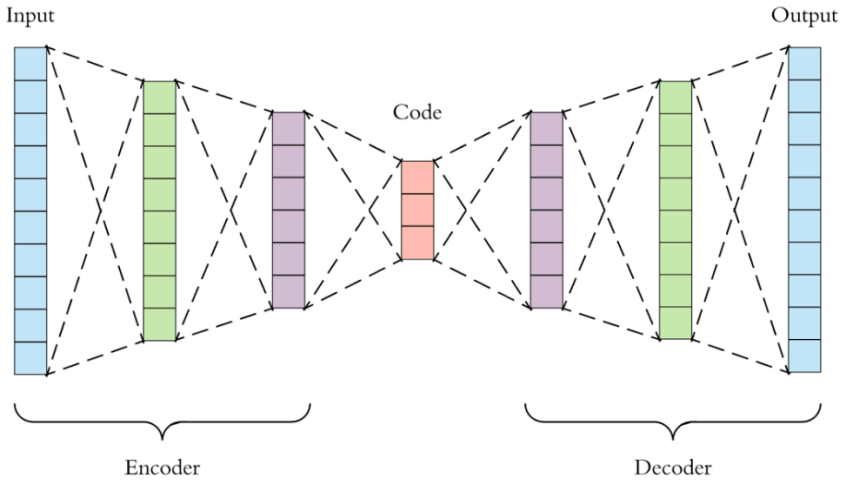

Figure 2.5: Architecture of an autoencoder (Singh & Palod, 2018)

Being x the initial input, the encoder can be presented as $h = f(x)$ and the decoder as $x' = g(h) = g[f(x)]$. Its objective is to minimize the divergence between x' and x. The layers that compose each one of the two subnetworks – encoder and decoder – must be chosen appropriately for the task in sight. If dealing with still data, fully connected layers are sufficient – although if dealing with images, convolutional layers are used (Masci, Meier, Cireşan, & Schmidhuber, 2011). However, when dealing with temporal data, recurrent layers like LSTMs must be employed (Gensler, Henze, Sick, & Raabe, 2016; Zhao, Feng, Zhao, Yang, & Yan, 2017).

They have a wide variety of applications such as denoising (Vincent, Larochelle, Bengio, & Manzagol, 2008), dimensionality reduction (Hinton & Salakhutdinov, 2006) or anomaly detection (Sakurada & Yairi, 2014; Zhou & Paffenroth, 2017).

A special kind of autoencoder is a Variational Autoencoder (VAE) (Kingma & Welling, 2013). A VAE is a generative model that forces prior suppositions about the distribution found in the latent space. This distribution is set beforehand, using, for example, a multivariate normal distribution. It is called generative because, after the VAE has been trained, it is possible to generate new samples out of it by sampling values from the probability distribution of the latent space – following the same example, the multivariate normal distribution (Doersch, 2016). This approach is different from the one followed by the other main kind of generative models, Generative Adversarial Networks (GANs), which don't use a probabilistic formulation of the latent space (Goodfellow, et al., 2014). VAEs are also used in conjunction with LSTM layers (Roberts, Engel, & Eck, 2017).

In our work, we don't use VAEs because we are not interested in generating new data samples and also because we don't have an expected distribution to use as prior in the latent space. We are only interested in performing dimensionality reduction to characterize the degradation of the system with the encoder.

To achieve our goal, we use a modification of an autoencoder. Instead of pairing the encoder with a decoder that reconstructs the original input, we pair it with a predictor. This predictor outputs the value that a healthy system should present in a healthy state. The network is then trained by minimizing the error between the value outputted by the predictor and the real measured value. By following this technique, and without giving labeled degradation data to the network, the encoder is able to learn the appropriate system's characterization that makes the predictor output the correct value for a particular input sequence. As previously mentioned, both the encoder and the decoder are composed of LSTM layers, which are responsible for taking into account the temporality found in the input sequence.

CHAPTER 3 ARTICLE INTRODUCTION

In Chapter 1, we introduced the importance of maintenance in industry. We showed how CBM is the most advanced maintenance strategy, as it can take advantage of the massive amounts of data collected in today's industries. By using this data and the appropriate technique, it is possible to predict faults in industrial equipment before they happen. In order to do so, we have presented the theory behind LSTM neural networks in Chapter 2. They are a type of supervised learning algorithm appropriate for dealing with multivariate time series, which is the most prominent data in industry and the type of data that we have.

In this article, we develop a fault prediction approach for an electrolyzer. An electrolyzer is a system composed of multiple electrochemical cells, where electrolysis takes place. Electrolysis is the process of decomposing a chemical product into various byproducts by applying an electric current (Wendt & Kreysa, 1999). Each electrochemical cell acts similarly to a resistor, causing a voltage drop when it receives an electrical current. The magnitude of this drop depends on the operating conditions and the cell's degradation, which is not known. Ensuring operation safety is of the utmost importance. An unexpected cell's fault may lead to catastrophic incidents, such as explosions and fire, with the consequent harm to the plant's operators and the environment.

Our approach relies on predicting the voltage that a healthy cell would present and comparing it to the cell's measured voltage in real-time, which increases steadily when a fault is about to happen. If there is a divergence, we signal a fault. In our article, this prediction is obtained by using a deep learning model. The objective of our model is to present a better voltage prediction and earlier fault detection than currently used techniques, proving that it can effectively improve safety in industries.

There are certain constraints related to the data available. In order to overcome them, relying only on sensor's data and without human intervention, we propose a new approach based on a neural encoder. The encoder is also interpretable, which permits to understand the decision process taken by the network. Our approach contributes to the literature to the extent that it is a novel way of using an encoder in a real-life application. New ways to remove the *black-box* stigma of neural network models are always a welcomed addition to the literature as well.

CHAPTER 4 ARTICLE 1: USING A SELF-SUPERVISED ENCODER FOR PREDICTING FAULTS IN ELECTROCHEMICAL CELLS

Manuscript submitted to *Expert Systems with Applications (ESWA)*

Poster accepted at *LXAI Research Workshop,* NeurIPS 2019, Vancouver

Authors.

Daniel Buades Marcos[1], daniel.buades-marcos@polymtl.ca

Soumaya Yacout[1], soumaya.yacout@polymtl.ca

Said Berriah[2] , said.berriah@r2.ca

Abstract.

Predicting faults before they occur helps to avoid potential safety hazards. Furthermore, planning the required maintenance actions in advance reduces operation costs. In this article, the focus is on electrochemical cells. In order to predict a cell's fault, the typical approach is to estimate the expected voltage that a healthy cell would present and compare it with the cell's measured voltage in real-time. This approach is possible because, when a fault is about to happen, the cell's measured voltage differs from the one expected for the same operating conditions. However, estimating the expected voltage is challenging, as the voltage of a healthy cell is also affected by its degradation – an unknown parameter. Expert-defined parametric models are currently used for this estimation task. Instead, we propose the use of a neural network model based on an encoder-decoder architecture. The network receives the operating conditions as input. The encoder's task is to find a faithful representation of the cell's degradation and to pass it to the decoder, which in turn predicts

---

[1] Polytechnique Montréal, 2900 Édouard Montpetit Blvd, Montréal, QC H3T 1J4, Canada

[2] R2 Inc., 380 Saint-Antoine St W Suite 7500, Montréal, QC H2Y 3X7, Canada

the expected cell's voltage. As no labeled degradation data is given to the network, we consider our approach to be a self-supervised encoder. Results show that we were able to predict the voltage of multiple cells while diminishing the prediction error that was obtained by the parametric models by 53%. This improvement enabled our network to predict a fault 31 hours before it happened, a 64% increase in reaction time compared to the parametric model. Moreover, the output of the encoder can be plotted, adding interpretability to the neural network model.

4.1 Problem Description

Electrolysis is the process of decomposing a chemical product into various byproducts by applying an electrical current (Wendt & Kreysa, 1999). It takes place in *electrolyzers*, which are systems composed of multiple *electrochemical cells*. These cells act similarly to resistors: electrical current passing through them causes a voltage drop. The magnitude of this voltage drop depends on the operating conditions and the cell's degradation, increasing steadily when a fault is about to happen.

Faults in electrochemical cells may become safety hazards. In order to diminish their occurrence, they are usually replaced every four years. This heuristic comes from the statistical analysis of the average lifetime of past cells, which represents an aggregation of data from multiple cells. Thus, it does not take into consideration the specificity of each cell, which is needed to take action concerning their maintenance. In order to implement a more efficient strategy that adequately considers such specificity, the cell's degradation must be monitored. However, degradation can only be determined directly by performing offline, and sometimes destructive, tests (Causserand & Aimar, 2010). Our objective is to use non-invasive methods that do not require the full stoppage of the electrolyzer. As such, the cell's degradation must be inferred indirectly from other measurable properties. In this article, our approach relies on predicting the voltage that a healthy cell would present ($\widehat{V}_t$) and comparing it to the cell's measured voltage in real-time (V_t). If there is a divergence between the two of them, a fault is signaled. The divergence threshold and the type of fault diagnosed depend on the shape of the divergence, following a set of rules proposed by experts.

In order to predict the voltage of a healthy cell, the following limitations are considered:

1. The cell's voltage drifts slowly over time due to its degradation.

2. Even for the same operating conditions and degradation level, the voltage of a cell differs from the ones of similar cells. This difference is induced by disparities in manufacturing, installation procedures, and other factors that cannot be directly quantified. The latter is referred to in this article as the specificity of each cell.
3. There is a delay between a change in the operating conditions and the response of the cell. In order to account for this delay, the operating conditions at the previous time-steps must be considered by the prediction model.
4. Inside the electrolyzer, cells are electrically connected in series and they all receive the same chemical input. This means that the only measurable variable that is different for each cell is their voltage. This voltage is stored according to the position of each cell in the electrolyzer, and not according to the unique identification number of each cell. Moreover, as previously stated, cells' degradation data is not available.
5. In order to predict faults, the predicted voltage must be independent from the measured voltage. This limitation entails that the measured voltage cannot be used as an input to the prediction model.
6. Electrolyzers are shut down and restarted many times throughout their lifetime. During each shutdown, the data collection is stopped. Any cell may be substituted or exchanged at the discretion of the plant's operator, without being reflected in the data. The intervals when the electrolyzer is operating are called cycles and, as all its cells are under tension, no changes are performed by the operator.

In this article, we present a new approach that overcomes these limitations to predict the voltage that a healthy cell would present for a certain set of operating conditions. In the next section, a review of the current methods and their application to this field is presented.

4.2 Previous Work

To be able to find the right model and techniques that apply to our problem, we first need to understand the system we are working with, which is a chemical electrolyzer. As previously stated, chemical electrolyzers are composed of multiple cells, where the electrolysis process takes place. There are different cell technologies, the most efficient being *Ion-Exchange Membranes*, on which

our article is focused. In these cells, the anode and the cathode are separated by a semi-permeable membrane that does not permit both electrolytes to mix, yet allows the ions needed for the electrolysis to travel across (Paidar, Fateev, & Bouzek, 2016). As time passes, holes start to appear in the membrane, and the electrodes start to lose their coating, thus increasing the voltage of the cell (Jalali, Mohammadi, & Ashrafizadeh, 2009). As the degradation advances, it reaches a point where an undesirable reaction between the solution in the anode and the solution in the cathode starts to occur. This reaction is characterized by a spike in the cell's voltage (The Chlorine Institute, 2018). It is a dangerous situation requiring the electrolyzer to be stopped immediately.

Therefore, detecting anomalies in the cell's voltage leads to predicting faults in the cell. As previously stated, we do so by predicting the expected voltage that a healthy cell would present and then comparing it with its actual measured voltage. The better the accuracy of the model used for predicting such voltage, the earlier it is possible to signal the fault.

In order to calculate the expected cell's voltage ($\widehat{V}_t$), the underlying chemical reaction needs to be approximated. This function is composed of many parameters, which need to be estimated. Some of them are operation-specific, i.e., identical for all the cells that perform the electrolysis for the same operating conditions. However, as per limitation 2, there are also cell-specific parameters. Moreover, limitation 1 implies that there are also degradation-specific parameters that change over time as the cell degrades.

In order to define a model that accounts for all these parameters, three types of data are needed: operation data, cell-specific data, and degradation-specific data. Nonetheless, as per limitations 4 and 5, we neither have cell-specific data nor degradation-specific data that we can use. Thus, if only operation-specific parameters are used, the same voltage would be predicted for each cell. This is not an acceptable result, because the specificity of each cell would be lost. In order to overcome this problem, the current approach relies on fitting a different model for each cell and retraining it periodically in order to update the degradation-specific parameters. As per limitation 6, the retraining frequency must be at least once per cycle.

An expert-defined parametric model is currently used for this task (US Patent No. 8,152,987, 2012). Operation-specific parameters are defined by the experts, while cell and degradation-specific parameters are estimated at the beginning of each cycle by using a linear regression. This

model has the advantage of being simple, thus needing fewer observations to train than equivalent non-parametric models (US Patent No. 7,818,276, 2010). This is crucial, as the observations used at each cycle for training the model cannot be used for predicting the cells' voltage. Therefore, during the training period, it is not possible to detect faults either. This model also has easy to explain results. Nonetheless, it requires an understanding of the chemical process to define it. Suppositions in the model entail an accuracy loss, as they do not account for all the details present in a production environment.

An alternative approach is to use non-parametric Machine Learning (ML) techniques. Support Vector Machines (SVM) were used to predict the voltage of a *chlor-alkali* cell and explore its response to different operating conditions (Shojai Kaveh, Mohammadi, & Ashrafizadeh, 2009). They obtained better accuracy than parametric models, but the scope of their work was limited to a single cell in a controlled lab environment with pre-defined operating conditions. They carried out a similar study using artificial Neural Networks (NNs) (Shojai Kaveh, Ashrafizadeh, & Mohammadi, 2008). However, their network only had two hidden layers and a reduced number of neurons, falling behind recently developed networks.

NNs are non-parametric models composed of multiple mathematical entities called *neurons*. Their name comes from the fact that they are inspired by how biological brains work (Goodfellow, Bengio, & Courville, Deep Learning, 2016). Neurons are grouped in layers. In the most basic neural architecture, called the Multi-Layer Perceptron (MLP), each neuron of a particular layer is connected to all the neurons of the previous layer. A scalar, called *weight*, quantify each connection. The neuron's output is the sum of the values of each neuron from the previous layer multiplied by their respective weights. Up to this point, the output of the model would be a linear combination. In order to approximate non-linear functions, the output of each neuron is transformed by an activation function (Nielsen, 2015). Indeed, when using non-linear activation functions and enough neurons combined with data, NNs are considered to be universal approximators (Pinkus, 1999). For our application, this is primordial: we are no longer forced to make assumptions about the underlying chemical function. There are many architectures derived from the general MLP, each of them specifically tailored to work with a different kind of input data and objective. Examples of these architectures are Recurrent Neural Networks (RNNs) and neural encoders, which are both used in this article.

As per limitation 3, we are interested in RNNs for our application. An RNN is a particular neural architecture that deals with sequential data. It accomplishes so by keeping an internal state that is updated for every time-step of the sequence (Karpathy, 2015). This internal state serves the purpose of a memory, allowing past information to persist in the network. In order to predict the output of a certain time-step, it considers the input for that time-step and the internal state from the previous time-steps. However, RNNs struggle when dealing with long temporal sequences (Bengio, Simard, & Frasconi, 1994). Long-Short Term Memory (LSTM) networks are a type of RNNs that does not suffer from this problem (Hochreiter & Schmidhuber, 1997). LSTMs are capable of modifying their internal state, either by removing or by adding new information, through the use of learnable gates (Olah, 2015). This flexibility makes LSTMs to be more commonly used in practice than RNNs.

A neural encoder is a type of neural architecture whose objective is to take an input vector and reduce its dimensionality to a desired one. It is usually paired with a decoder. The decoder receives the output of the encoder and transforms it to minimize an objective function. The whole network is trained *backpropagating* the loss of the decoder (Rumelhart, Hinton, & Williams, 1985). If the objective of the decoder is to reconstruct the original input vector, it is called an autoencoder (Hinton & Salakhutdinov, 2006). A review of different types of autoencoders is presented in the work of Tschannen et al. (Tschannen, Bachem, & Lucic, 2018). Autoencoders are often used for anomaly detection (Sakurada & Yairi, 2014) and noise reduction (Vincent, Larochelle, Bengio, & Manzagol, 2008). They are also applicable to time sequences in combination with LSTMs (Malhotra, et al., 2016).

4.2.1 Originality

In this article, we develop a method to apply NNs to the voltage prediction of all the electrolyzer's cells. The model works in a production environment, where operating conditions are not pre-defined, and each cell has a different level of degradation. The model also has better accuracy than the parametric model currently in use. Moreover, it does not need more observations per cycle for starting to make predictions.

As NNs require more data than parametric models to approximate a function, we take a different approach. Instead of fitting a different model for each cell and cycle, we propose a neural network

that is trained with data currently available, thus avoiding retraining once deployed. This network is based on an *encoder-decoder* architecture, where we substitute the *decoder* by a *predictor* – a subnetwork that predicts the cell's voltage. Therefore, instead of training the network by minimizing the reconstruction error of the input sequence, we train it by minimizing the error between the voltage predicted by the network and the measured voltage.

In order to overcome all the six limitations previously mentioned, the originality of our approach in comparison with the already existing approaches is:

1. The encoder subnetwork addresses limitations 1, 2, and 6. It does so by finding two features that represent the specificity of each cell and that are updated at each cycle to account for the degradation.
2. The predictor addresses limitation 3, as it takes into account the temporality of the observations.
3. Together, the encoder and the predictor address limitations 4 and 5. The predictor does not use the voltage as an input, yet it is still able to predict a different voltage for each cell, despite using the same operating conditions as input. It accomplishes so by taking the output of the encoder as an additional input, which is unique for each cell. Hence, the voltage prediction is not biased by the cell's measured voltage.

4.3 Data preparation

4.3.1 Features

Data is collected from two different sources. The first source is the plant's control system, which registers three features common to all the cells in the electrolyzer. These features are the electrical current that passes through the cells and the temperature plus the concentration of the caustic at the outlet of the electrolyzer. The second source is the output of sensors that measure the voltage of each cell individually.

Cells' measurements are carried out sequentially, which means that the controller reads a sensor and then proceeds to read the next one. This process is repeated for every cell in the electrolyzer and takes between one and two seconds to loop over all the cells. The plant's controller, on the

other hand, does not follow a strict pattern of data collection. Each sensor connected to it has a different sampling rate, varying from one to around thirty seconds, which results in data observations that are misaligned and sampled at different intervals. In order to solve this problem, we downsample and align the measurements to the minute. This approach provides enough resolution to detect possible faults, as faults develop in the scale of hours. It also reduces the amount of unnecessary training data and facilitates the deployment of the model in slower computing processors, as the latency requirements are less strict.

The resulting data after alignment is tabulated. Each row represents a different time of observation, and there is a different column for each electrolyzer's feature and cell. An electrolyzer is usually formed of around 160 cells. An excerpt of this data is presented in Table 4.1.

Table 4.1: Subset of the dataset. n represents the last observation, and m the last electrolyzer's cell.

Index	**Current [kA]**	**Temperature [ºC]**	**Concentration [%]**	**Voltage$_{\text{Cell 1}}$ [V]**	**…**	**Voltage$_{\text{Cell m}}$ [V]**
1	1.163044	76.667840	31.949131	2.475286	…	2.468372
2	1.887390	76.555115	31.944921	2.531715	…	2.517775
3	2.072368	76.501010	31.940712	2.561972	…	2.549233
4	2.035982	76.449160	31.936502	2.560415	…	2.554348
5	2.424838	76.397310	31.936700	2.576541	…	2.561825
…	…	…	…	…	…	…
n-4	16.31074	88.74237	32.67195	3.379979	…	3.317668
n-3	16.31025	88.73323	32.67271	3.379979	…	3.31797
n-2	16.30976	88.72409	32.67348	3.38039	…	3.317779
n-1	16.30928	88.71495	32.67425	3.380894	…	3.317477
n	16.30879	88.7058	32.67502	3.381398	…	3.317174

4.3.2 Cycles

Electrolyzers are stopped and started many times during their lifetime due to production constraints, changing demand, work shifts, or maintenance requirements. The interval of time between a consecutive start and stop is known as a *cycle,* and its length ranges from some hours to several weeks, depending on those conditions. Each cycle is divided into two phases of different lengths – the *startup* and the *operation* phase. Both cycle's phases are shown in Figure 4.1.

The startup phase occurs when the electrical current increases from zero to 16 kA. The 16 kA threshold, defined by an expert, represents the moment when the cells reach their full production conditions. The rate at which the current increases differs for each startup, due to changes in the operating practices decided by the plant's operators. The length of this phase ranges anywhere from 20 minutes to 12 hours, which complicates making comparisons between different startups. This length's difference is shown in Figure 4.2. As previously mentioned, each cell responds differently to the same operating conditions, resulting in a different voltage increase during the startup. The voltage increases of three different cells are shown in Figure 4.3.

Note that cycles A, B, and C, as well as cells X, Y, and Z, are used as examples through the article. We chose these cycles because their operating conditions are significantly different. The three cells are chosen because they have different levels of degradation. Please note that their color schemes remain the same for all the figures in the article.

The operation phase includes the rest of the cycle. The electrical current varies between 7 kA and 16 kA. We are especially interested in predicting the voltage during this phase, as a cell's fault here would cause a significant disturbance.

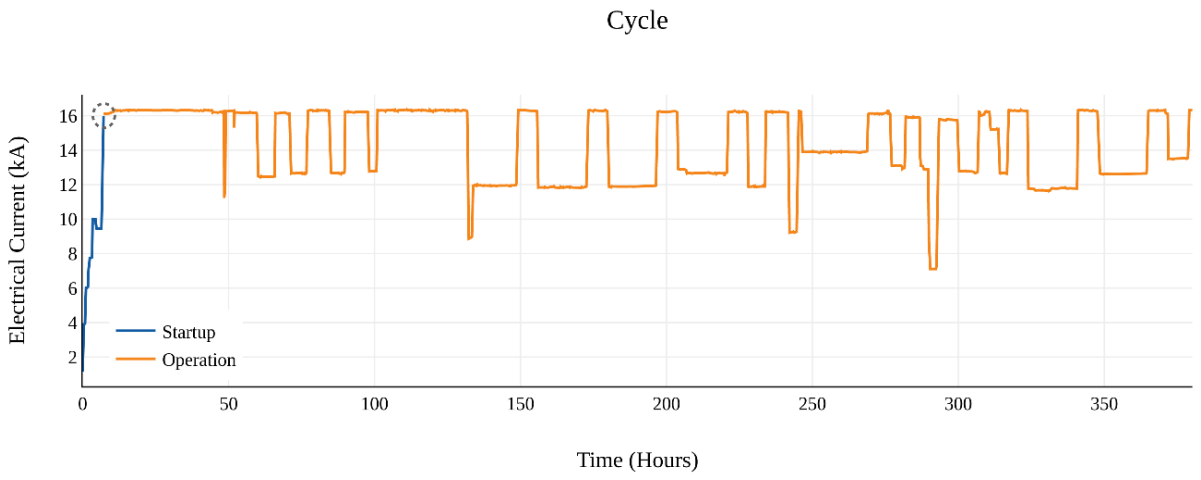

Figure 4.1: Evolution of the electrical current of the electrolyzer during a complete cycle. The startup phase is plotted in blue and the operation phase, in orange. Notice the length's difference between both phases.

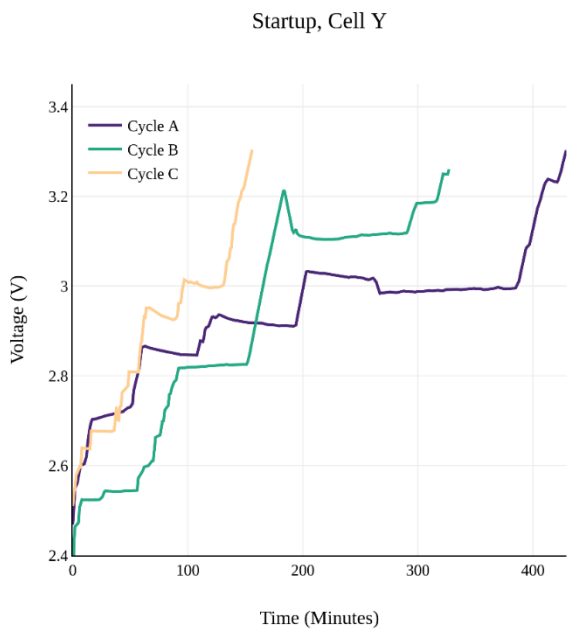

Figure 4.2: Voltage during the startup phase of the same cell (Y) for three different cycles (A, B, C).

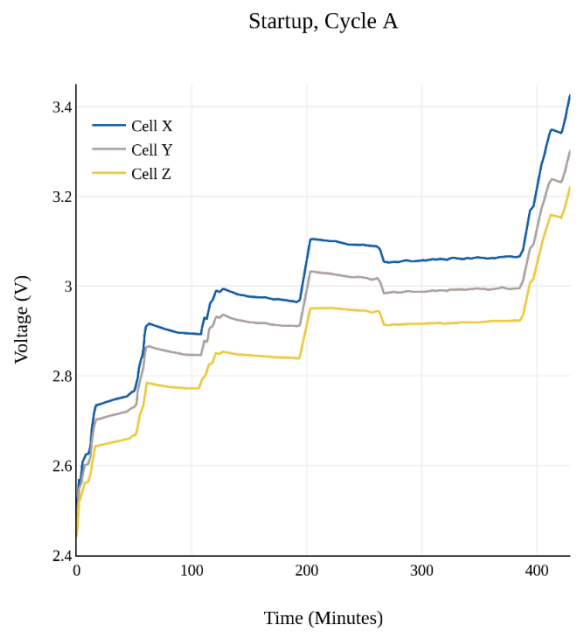

Figure 4.3: Voltage during the startup phase of three different cells (X, Y, Z) for the same cycle (A).

4.4 Methodology

4.4.1 Data Processing

As previously stated, when the electrolyzer is shut down, data collection is stopped, and so, it appears as missing data in the file. A new cycle is detected if the time difference between two consecutive observations is longer than 10 minutes. Moreover, the following experts' conditions must be met:

1. The electrical current reaches 16 kA during the cycle.
2. The duration of the startup phase is less than 12 hours.
3. The operation phase has at least the same duration as the startup phase.

Data files are processed in order to ensure the satisfaction of these conditions, following the procedure outlined in Table 4.2 and Table 4.3.

Table 4.2: Pseudo-code for detecting possible cycles.

```
for observation n in the file do
    Diff_n = Time_n − Time_{n−1}
    if Diff_n > 10 minutes then
        Save Time_{n−1} and Time_n;
    end
end
Each possible cycle is enclosed by the pair of dates corresponding to:
 (0, Time_{n−1}), (Time_n, Time_{n+1}), (Time_{n+2}, Time_{n+3})
```

Table 4.3: Pseudo-code for confirming cycles.

```
for each possible cycle_n with length k do
    if any observation of the cycle has current > 16 kA then
        idx = First observation where current > 16 kA
        if idx ≤ 12 hours then
            startup_n = cycle[0:idx]
            operation_n = cycle[idx:k]
            if lenght(operation) ≥ length(startup) then
                cycle_n is a valid cycle
            end
        end
    end
end
```

Once the data is structured in cycles, with their startup and operation phases defined, we proceed to scale it. For each data column, we use the *unity-based normalization* scaling method, also known as *Min-Max Scaling*. It scales the data linearly, so all the values are in the range [0,1] (Raghav, Lemaitre, & Unterthiner, 2018). We do not use more sophisticated scaling techniques since our data is not normally distributed and since outliers are already removed in the original database. The column is scaled following Equation (4.1).

$$X_{scaled} = \frac{X - min}{max - min} \tag{4.1}$$

The minimum (min) and maximum (max) values use for scaling each feature are common for all the production plants and are determined by experts. They are the same for all the electrolyzers, so it is possible to use the same trained model for all of them. Missing observations appear as *'NaN'* and are substituted by the value of *'-1'*, in order to keep them outside of the scaler's range. After following this procedure, a subset of the resulting data is shown in Table 4.4.

The final step is to export the data to TFRecord files, which is a binary format developed by Google and optimized for the preprocessing tf.data.Dataset API of TensorFlow 2.0 (Google, 2019). We export a different file for each cycle, containing all the cell's observations.

Table 4.4: Subset of data from Cycle A after being processed. Data has been scaled and missing observations, replaced by '-1'. Note that, as expected, the input features at each time-step are the same for cells one and *m*, where *m* represents the last cell of the electrolyzer.

Cell	Phase	Index	Current	Temperature	Concentration	Voltage
1	Startup	1	0.009939	0.416964	0.790073	0.440844
		2	0.054435	0.412617	0.78969	0.449887
		3	0.065797	0.41053	0.789307	0.454735
		4	0.063562	-1	0.788924	0.454486
		…	…	…	…	…
		i^a	0.919915	0.789966	0.851393	0.586628
	Operation	1	0.928491	0.806401	0.851979	0.586906
		2	0.928461	0.810509	-1	0.586698
		3	0.928431	0.814618	0.852273	0.586504
		4	0.928401	0.818727	0.852419	0.586343
		…	…	…	…	…
		j^b	0.940324	0.88117	0.856124	0.586044
…	…	…	…	…	…	…
m	Startup	1	0.009939	0.416964	0.790073	0.439736
		2	0.054435	0.412617	0.78969	0.447653
		3	0.065797	0.41053	0.789307	0.452694
		4	0.063562	-1	0.788924	0.453514
		…	…	…	…	…
		i^a	0.919915	0.789966	0.851393	0.573423
	Operation	1	0.928491	0.806401	0.851979	0.574696
		2	0.928461	0.810509	-1	0.574573
		3	0.928431	0.814618	0.852273	0.574476
		4	0.928401	0.818727	0.852419	0.574379
		…	…	…	…	…
		j^b	0.940324	0.88117	0.856124	0.575753

[a] *Last startup observation of cycle A;* [b] *Last operation observation of cycle A.*

4.4.2 Model Architecture

As previously explained, we want to predict the voltage over the whole operation phase of the cycle, so the training must only be performed during the startup phase. However, neural networks are complex non-linear methods that require a large amount of data to converge to an optimal solution. Hence, only using the startup data for training a new model at each cycle does not provide satisfactory results.

We have collected data from multiple cells across multiple cycles of different electrolyzers. Ideally, we would like to use all this data to train our network, but it is not straightforward to do so. The reason is because, as previously explained, each cell presents a different response to operating conditions, but no feature or labeled data is available to differentiate them.

One possible approach is to train a base naïve model and then use transfer learning to retrain the last layers of the model. However, we would need to do that for each cell at the beginning of each cycle, which is a computationally intensive task that would complicate the deployment (Weiss, Khoshgoftaar, & Wang, 2016). Instead, we propose a different approach inspired by how an expert may look at many cells and differentiate them based solely on the evolution of their voltage during the startup sequence. Based on this idea, we deduce that it is possible to use the voltage of each cell during the startup phase to infer meaningful features that characterize such cells.

These features account for both the specificity of the cell and its degradation and are used as input for the predictor model. This model also takes the operating conditions in order to predict the cell's voltage during the operation phase. The cell's voltage is only used as input during the startup phase for inferring the features. Indeed, the predictor model does not use the cell's voltage during the operation phase, so the predictions are not biased. Given enough data belonging to multiple cells and cycles, we would cover the whole subspace of possible features. Thus, new cells presented to the model would behave like cells that the model had already seen before, and so, it could also predict their voltage accurately.

By using this approach, it is possible to train a single model that works for all the cells. Such a model predicts a different voltage for each cell based not only on the operating conditions but also on the specific cell's features inferred during the startup. This effectively avoids the need to retrain

the model for each cycle and cell. At the same time, it allows us to train it beforehand with the whole dataset that we already have.

In order to implement this model, we propose a neural network formed by two subnetworks: a self-supervised encoder and a decoder – or voltage predictor. These two subnetworks are presented together in Figure 4.4, a figure that is referenced during the two following subsections, where both subnetworks are developed in detail. The algorithm needed for training such a network is developed in section 4.4.3.

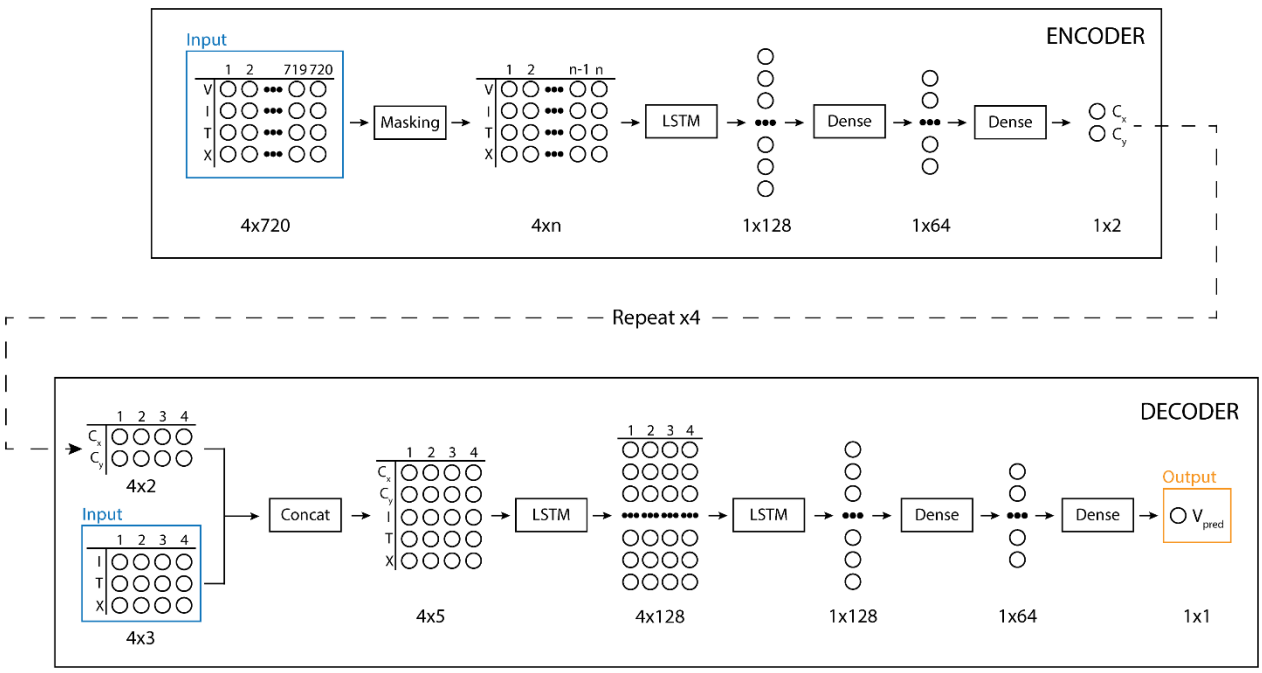

Figure 4.4: Architecture of the proposed model. Rows in the matrices represent the different features: V ≡ Cell's voltage, I ≡ Electrical current, T ≡ Temperature, X ≡ Caustic concentration. C_x and C_y are the two outputs of the encoder. V_{pred} is the cell's voltage outputted by the model. Columns in the matrices represent the number of timesteps. Initially, there are always 720 startup timesteps but, after the masking layer, they differ for each cycle and are represented by the letter *n*. The operation timesteps are always four.

4.4.2.1 Encoder

The encoder subnetwork is the key part of this model. Its goal is to infer the features that characterize the behavior of a particular cell during a certain cycle, using only its startup phase. It is a self-supervised method, as it does not need labeled degradation data. This subnetwork does not have a loss function on its own. Instead, it is trained with the loss backpropagated from the voltage predictor subnetwork.

At its core, it is a form of performing a dimensionality reduction. However, the encoded vector – the dimensionality-reduced vector – is not the result of a statistical procedure, but the optimal representation that eases the learning of the predictor subnetwork.

We use the three input features from the control plant – temperature, caustic concentration, and electrical current – as well as the voltage of the cell. The input features are needed to standardize the cell's voltage to the specific operating conditions of each startup. All in all, the shape of the input vector is [720 time-steps, 4 features]. The masking layer forces the successive layers to ignore a time-step if all the features of that time-step are equal to a masked value, which is *'-1'* in order to filter the time-steps added during batch padding. In order to account for the temporality of the sequence, the next layer is a Long Short-Term Memory (LSTM). After it, two dense layers are chained to make a smoother transition to the final two-positional encoded result.

Its output is a vector of coordinates [X, Y] for each cell and startup, with a shape of (1 time-step, 2 features). Moreover, these coordinates can be represented in a graph, providing an insight into the decision process taken by the network.

This subnetwork corresponds to the upper block in Figure 4.4 and its layers' characteristics – in TensorFlow's terminology – are depicted in Figure 4.5.

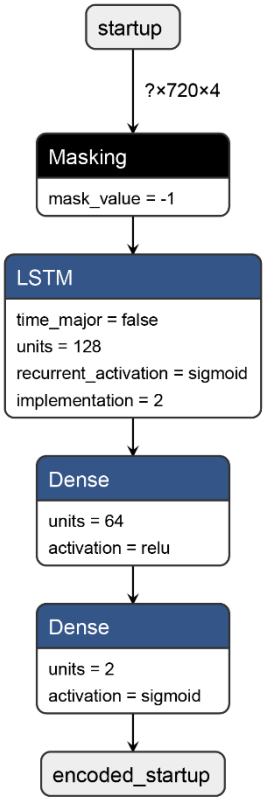

Figure 4.5: Encoder subnetwork.

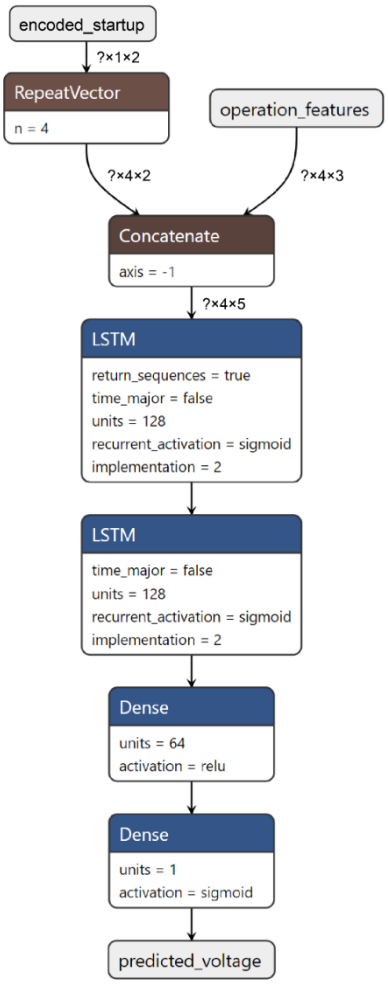

Figure 4.6: Predictor subnetwork.

4.4.2.2 Predictor

The predictor subnetwork, as its name indicates, is responsible for predicting the cell's voltage. It takes two inputs: a window of time-steps from the operation phase and the encoded representation of the cell's startup phase. We have heuristically determined that a window of four observations is enough to represent the dynamics of the chemical phenomena behind the cell's response.

The encoded cell's startup is repeated four times and concatenated with the window of operation features. Note that the voltage is not included in the window of operation features. Two LSTM layers are used to find temporal correlations between the observations of each window. Two dense

layers follow, in order to output the predicted voltage. The output layer has a sigmoid activation function, as the output voltage was scaled previously to the range [0, 1].

The whole model is trained by minimizing the loss between the voltage predicted by this subnetwork and the measured voltage. The Adam optimizer (Kingma & Ba, 2014) and the backpropagation algorithm are used. For this subnetwork to get a good accuracy in the voltage prediction, the encoder must learn a faithful representation of the characterization of the cell. This subnetwork is the bottom block of Figure 4.4 and its layers' parameters are presented in Figure 4.6.

4.4.3 Training Procedure

We start with the data presented in Table 4.4. As previously mentioned in section 4.4.2.2, in order to account for the temporality of the chemical reaction, we first need to group four consecutive time-steps into a single data entry. This data entry is known as *window* or *observation*. In order to train a model with this kind of data, we define a custom training loop. Table 4.5 presents the training loop for stochastic training – one observation per forward-backward pass.

Table 4.5: Pseudo-code for the stochastic training algorithm.

```
for each cycle in electrolyzer's lifespan do
    Load cycle's corresponding TFRecord file
    for each cell in electrolyzer do
        Separate cell_startup and cell_operation
        Divide cell_operation in windows of four timesteps
        for each window do
            operating_conditions = all window's features except the voltage
            real_cell_voltage = voltage for the last window's timestep

            # Forward pass
            encoded_cell_startup = encoder(cell_startup)
            predicted_voltage = predictor(operating_conditions, encoded_cell_startup)
            loss = mean_squared_error(predicted_voltage, real_cell_voltage)

            # Backward pass
            update_network_weights(loss)
        end
    end
end
```

However, such a training loop is not efficient. In order to leverage the parallel computation made possible by GPUs, we use *mini-batch* training instead. In this mode of training, many observations are grouped in a *batch* and ingested by the GPU at the same time.

In order to make it possible to tweak different parameters effortlessly, we decided to generate the windows and batches in real-time during training. The procedure is similar to that followed when doing *data augmentation*, which is used extensively in computer vision (Taylor & Nitschke, 2017). While the GPU is processing a batch of windows, the CPU is already processing the next batch. The GPU is fast, so the CPU quickly becomes the bottleneck in such a pipeline. In order to overcome this issue, we use the *tf.data.Dataset* API to parallelize the input pipeline. We trained the network with two GPUs in parallel and a combined batch size of 1024 observations. Thanks to this parallelized CPU input pipeline, the utilization of both GPUs was consistently over 90%.

Three operations are crucial to make the network converge efficiently: padding, shuffling, and window striding.

4.4.3.1 Padding

Not every cycle's startup has the same duration. However, all the batches that we feed to the GPU must have the same number of time-steps. We solve this problem by padding the sequences – adding '-1' values at the end of each observation's corresponding startup. This way, all the startups have the same duration of 720 minutes, which is the maximum duration of a startup. This padding value is later ignored by the masking layer of the encoder subnetwork, so it does not affect the results. Table 4.6 shows the startup phase of cycle A after being padded.

Table 4.6: Padded startup. "i" represents the last startup observation before padding.

Index	Current	Temperature	Concentration	Voltage
1	0.009939	0.416964	0.790073	0.440844
2	0.054435	0.412617	0.78969	0.449887
3	0.065797	0.41053	0.789307	0.454735
4	0.063562	-1	0.788924	0.454486
…	…	…	…	…
i	0.919915	0.789966	0.851393	0.586628
i+1	-1	-1	-1	-1
…	…	…	…	…
719	-1	-1	-1	-1
720	-1	-1	-1	-1

4.4.3.2 Shuffling

We observed that the time required by the network to converge to an optimal solution was reduced considerably by the introduction of shuffling in the training process. There is an explanation for this behavior. Without shuffling, most batches only have observations from a single cell and cycle of the same electrolyzer. This situation constitutes a problem because the gradient updates at each batch are very different. However, after introducing shuffling, each batch has now observations from different cells, cycles, and electrolyzers. As a result, we obtain less noise in the backpropagated gradient and a smoother training loss, which helps the network weights to converge.

However, due to the considerable number of observations in the training dataset, it is not possible to perform the shuffling operation entirely in-memory. In order to overcome this limitation, we combine two strategies:

1. Shuffle Buffer: instead of shuffling the whole dataset, we only shuffle one subset of observations at a time. This subset is the shuffle buffer, and its number of observations is limited by the size of the computer's RAM. After the observations have been shuffled, we

give them to the network for training. Once the buffer is depleted, a new subset is read, and the same operation is performed again.

2. Interleaving: as previously stated, we have a different file for each cycle and electrolyzer. We randomly chose a file, read an observation from it, and add it to the shuffle buffer. This process is repeated in a loop. Thanks to using interleaving, the shuffle buffer is filled with observations from multiple cycles and electrolyzers, thus increasing the randomness of the batches given to the network.

4.4.3.3 Window Striding

As shown in Table 4.5, the encoder subnetwork updates its weights for each batch of operation observations. However, the encoder task of inferring the cell's features during the startup is more complex than that of the predictor. Hence, it is more efficient to train the network with fewer observations per cycle and more different startup sequences. In order to solve this, we increase the stride of our windowing function. The *stride* is the number that defines how many windows of the sequence are ignored between two consecutive training observations. For example, a stride of 64 means that, for each cell and cycle, we only take one window every 64. Figure 4.7 presents an example of two different striding values.

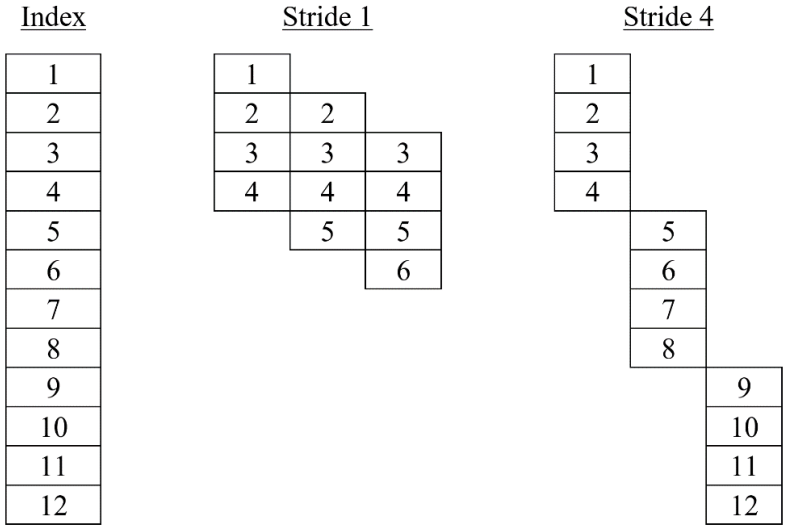

Figure 4.7: Example of how striding works. Each index number represents a window. For a stride of one, consecutive windows are picked for training. For a stride of four, only one observation every four is picked.

4.4.4 Testing Procedure

4.4.4.1 Parametric Model

We use the parametric model as the baseline for comparing the results obtained by our neural network. This model predicts the cell's voltage ($\hat{V}$) following Equation (4.2). The advantage of this model over other parametric models is the fewer required observations for getting an equivalent accuracy. This is crucial since its training data is limited to the startup phase of each cycle.

$$\hat{V} = \mathrm{u}_0 + [\mathrm{k} + (90 - \mathrm{T}) * C_t + (32 - \mathrm{C}) * C_c\,] * I/A \tag{4.2}$$

<u>Parameters</u>

$\mathbf{C_t}$: Temperature correction factor $[V/°\mathrm{C} * m^2/kA]$

$\mathbf{C_c}$: Caustic correction factor $[V/\% * m^2/kA]$

$\mathbf{u_0}$: Cell's Equilibrium Voltage [V]

$\mathbf{k}$: Load dependent resistance $[V * m^2/kA]$

$\boldsymbol{A}$: Membrane's Surface Area [m^2]

<u>Inputs</u>

$\mathbf{I}$: Electrical Current [kA]

$\mathbf{T}$: Temperature [°C]

$\mathbf{C}$: Caustic Concentration [%]

The cell's manufacturer gives parameter $\boldsymbol{A}$, which for the cells in our dataset is 2.721. Parameters $\boldsymbol{C_t}$ and $\boldsymbol{C_c}$ are estimated for each plant by experts. For our test plant, $\boldsymbol{C_t}$ = 0.0016 and $\boldsymbol{C_c}$ = -0.0031. Parameters $\boldsymbol{u_0}$ and $\boldsymbol{k}$ depend on the degradation of each specific cell and must be estimated at the beginning of each cycle. In order to estimate them, a linear regression is fitted by minimizing the sum of least squares: $\sum(\hat{V} - V)^2$ with all of the cycle's startup observations.

4.4.4.2 Method

All the tests were carried out on data collected from three years of an electrolyzer. During this period, the electrolyzer went through 40 different cycles. As the electrolyzer has 160 cells, there are 6400 combinations of cells and cycles in the dataset. The data used follows the structure previously presented in Table 4.4.

A new parametric model was fitted for each cell during the startup phase of each cycle, following Equation 2. Consequently, this means that 6400 different models were trained. In contrast, with our approach, only one neural network model was trained. The network was trained with data from six different electrolyzers from the same plant, each one of them having different cells and startup sequences. In total, around 43200 combinations of cells and cycles were seen by the network during training. Of course, the electrolyzer used for testing was neither used for training nor for validation to decide when to stop training the network. In order to make the predictions, the startup sequence is given to the network, but no retraining is performed. Figure 4.8 illustrates the differences between the process followed by the parametric model and the one followed by the neural network.

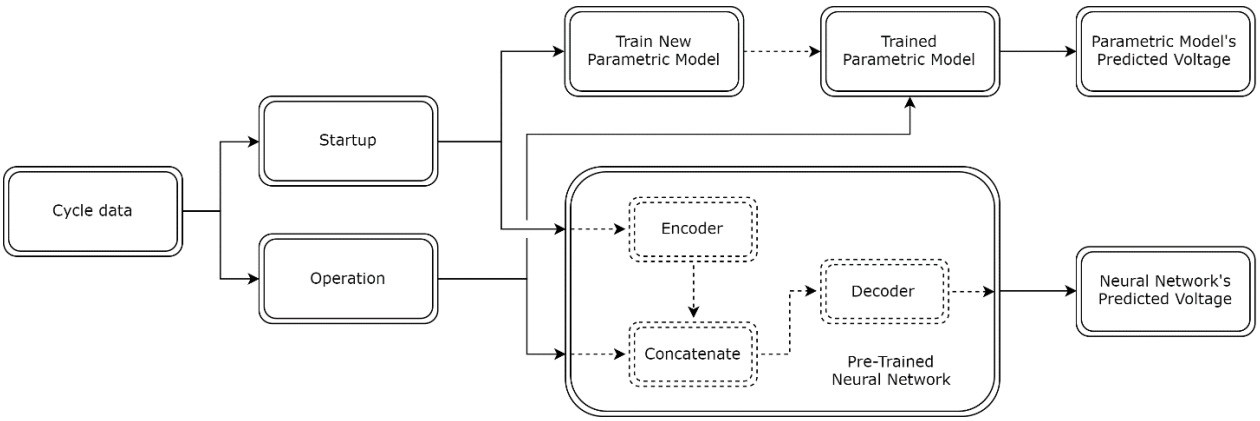

Figure 4.8: Steps taken by each model for making predictions.

4.5 Results and Discussion

4.5.1 Network insight

We start by discussing the encoder's results. In Figure 4.9, it is possible to see the evolution of the encoding of cells X, Y, and Z along cycles A, B, and C of the testing electrolyzer. Each point presented in Figure 8 corresponds to the encoded startup sequence of a specific cell and cycle. As previously explained, each cell has its own specificity and degradation. Let us take the example of cell Z to showcase the veracity of the encoder's results. We know thanks to an expert that cell Z is the most degraded cell of the whole electrolyzer for these three cycles. In the graph, we can clearly see this, as cell Z is plotted at the cycle's extreme. Since the startup's sequences of the testing

electrolyzer are different from those of the training electrolyzers, we prove that the encoder is effectively learning the degradation of the cells and not only memorizing the startup sequences.

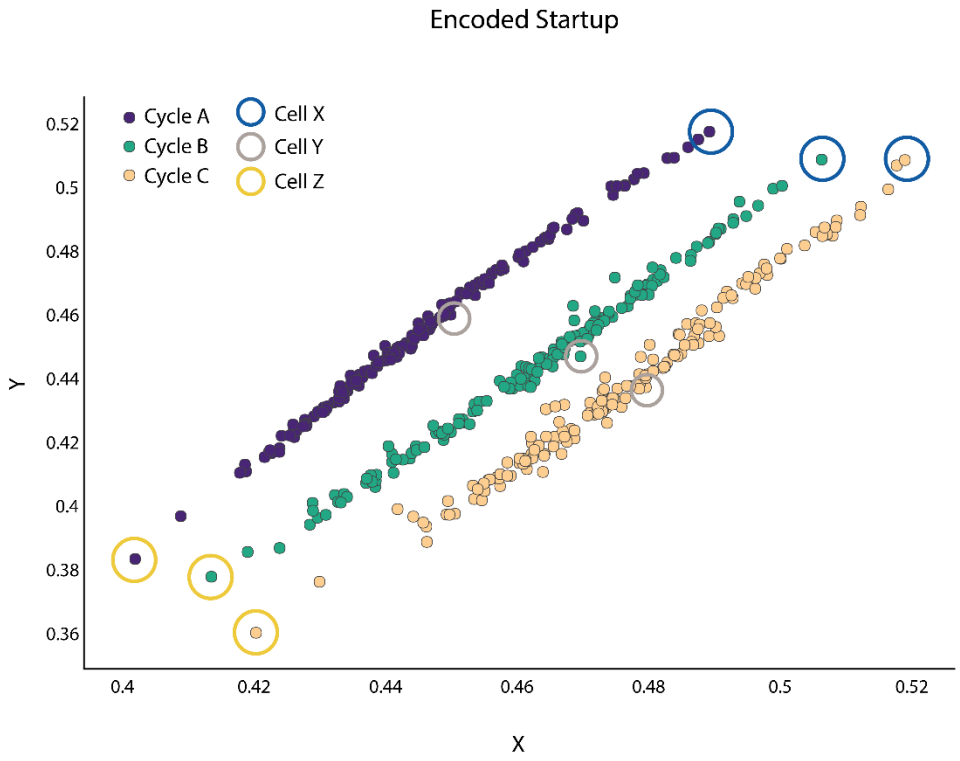

Figure 4.9: Evolution of the encoding of cells X, Y, and Z along cycles A, B, and C. The axes of the graph do not have units, as they are the result of a dimensionality reduction. They take values in the interval [0,1] because the output of the encoder is a sigmoid function.

Furthermore, the relative position of a cell compared to the rest of the cells of that cycle stays stable over time. This circumstance can be used as a safeguard for the prediction. Indeed, if we notice that the predicted voltage of a cell presents a significant error, we can check in the graph if the relative position of that cell has changed between the last cycle and the current one. If there is a significant variation, the voltage prediction error is most likely due to a problem with the encoder subnetwork and not with the cell. This is why we say that the network's results can be interpreted.

In Figure 4.10, we present the output voltage of the three cells highlighted in Figure 4.9 during a subset of the operation phase of cycle A. We show that our model is capable of predicting their

respective voltages. We can also see that the cell X, whose startup was plotted at the top of the cycle in Figure 4.9, present the lowest voltage of the three. Cell Y was encoded in the middle in Figure 4.9 and, indeed, its voltage is between the two other cells. Similarly, cell Z was at the bottom of the cycle in Figure 4.9, and it presents the highest voltage in Figure 4.10. This result is consistent in our tests with different cells. Thus, we can conclude that the magnitude of the predicted voltage is related to the position learned by the encoder.

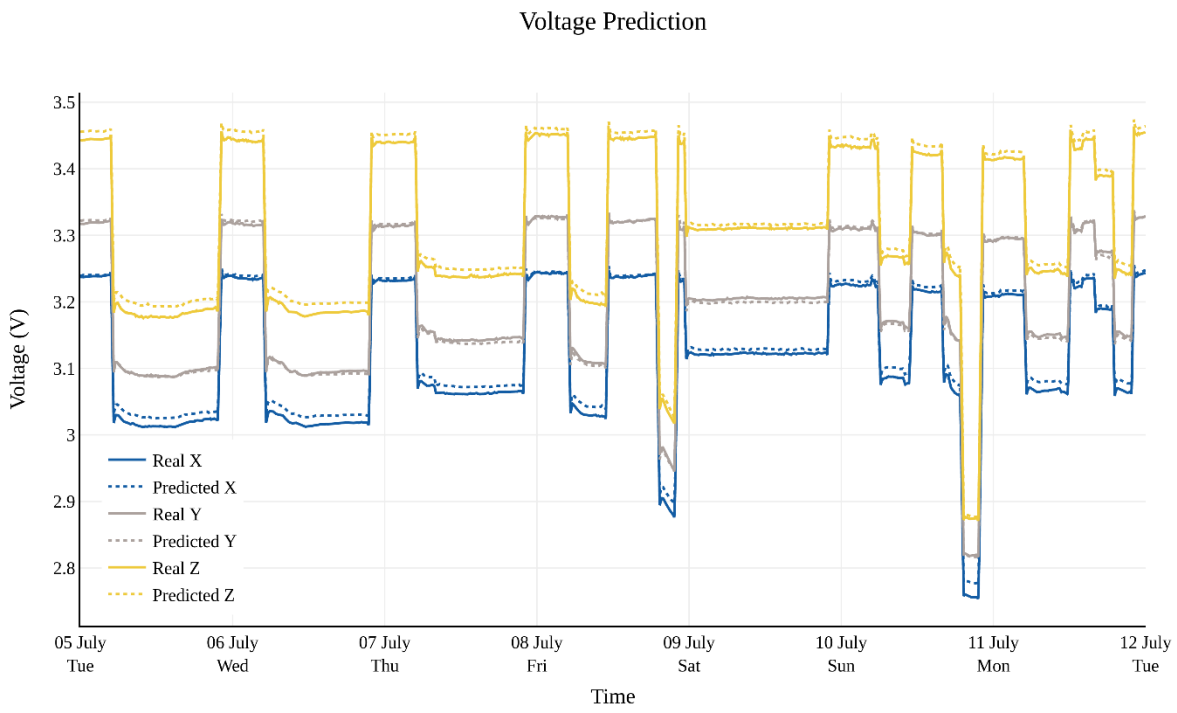

Figure 4.10: Predicted voltage for cells X, Y, and Z during cycle A. A solid line represents the real voltage of each cell, while a dotted line represents the predicted voltage.

In order to better visualize the prediction error, a closer look into a two-day period of the operation phase of cycle A is provided in Figure 4.11. We compare the behavior of the neural network against the parametric model and show that the voltage predicted by the network is more stable than the one predicted by the parametric model. Indeed, the parametric model produces an unstable behavior because the experts' understanding of the reaction's kinetics is limited, and because, as

previously explained, the parametric model must be simple in order to be trained using only the startup data.

The response of the cell varies between a *high load* and a *low load*. However, the parametric model is not capable of adapting its response to both of them, so it fits a response based on a *medium load*. Hence, an increased error is noticeable in Figure 4.11 when the current is around 16 kA or around 7 kA. Indeed, the parametric model works better around 13 kA, which corresponds to a *medium load*. The neural network, however, has no problem predicting the response for the different load levels, as it is not constrained to follow a linear relationship between the inputs. The fact that our model considers the temporality of the time series also contributes to decreasing the absolute error, as the kinetics of the cell are taken into account by the predictor subnetwork. It especially helps during load changes, smoothing the transition.

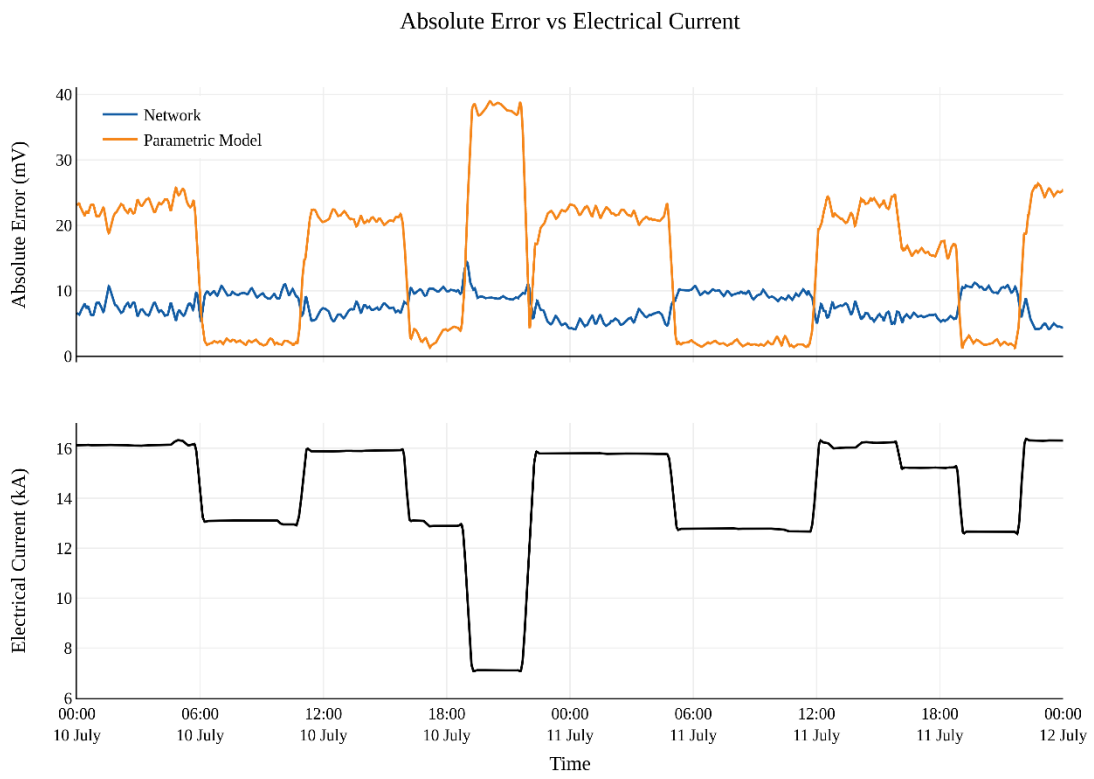

Figure 4.11: Extract of cycle A. The upper subgraph shows the absolute error between the predicted and the measured voltage, averaged over Cells X, Y, and Z. The lower subgraph presents the electrical current.

4.5.2 Accuracy

We are interested in comparing the accuracy between cycles (inter-cycle), as well as comparing the accuracy inside the cycle (intra-cycle). We seek accurate results in both cases.

4.5.2.1 Between cycles

For each cell and cycle, we calculate the average absolute error for all the observations. We then calculate a set of statistics for that averaged error, which are presented in Table 4.7. Measuring the accuracy between cycles is vital, as a model must work correctly for as many cells and cycles as possible. In fact, a model that is consistent with its predictions is preferred over a model that makes excellent predictions for some cells but weak ones for the rest of them. Even if the former has a higher error on average.

Based on the results presented in Table 4.7, we conclude that our neural network is more reliable than the parametric model. Not only is the average error lower, but also the standard deviation, which indicates that the error is less dispersed between different cells and cycles. The parametric model produces more outliers, as its two highest percentiles present a significant deviation compared to those of the network.

Table 4.7: Statistics of the average absolute error of both models across the different cells and cycles, presented for different percentiles.

Statistics	Neural Network [mV]	Parametric Model [mV]
Mean	11.977	25.668
SD	11.503	30.304
25th	5.470	8.398
50th	8.432	16.225
75th	13.488	30.760
90th	24.167	53.262
95th	35.461	82.745
99th	62.676	157.488

4.5.2.2 Inside cycle

For each combination of cell and cycle, we calculate a set of statistics that represents the distribution of the error and average them across all the 6400 combinations. Table 4.8 presents these results. The neural network obtains better results than the parametric model for all the statistics. As expected, the average voltage is the same as in Table 4.7.

Table 4.8: Average statistics of the absolute error inside a cycle, presented for different percentiles.

Statistics	Neural Network [mV]	Parametric Model [mV]
Mean	11.977	25.668
SD	4.586	6.109
25th	8.720	21.637
50th	11.850	25.435
75th	14.975	29.816
90th	17.712	33.184
95th	19.338	34.943
99th	22.213	39.088

We can conclude that the accuracy intra-cycle is better than the inter-cycle one. One possible explanation for this difference comes from the startup phase. For the neural network, when the encoder works correctly, the error stays constant inside the cycle. However, when the encoder fails to identify the degradation of the cell correctly, the accuracy of the whole cycle is penalized. The same holds true for the parametric model. When the startup is short, it does not have enough data to train, and the predictions are inaccurate for the whole cycle.

4.5.3 Fault Detection

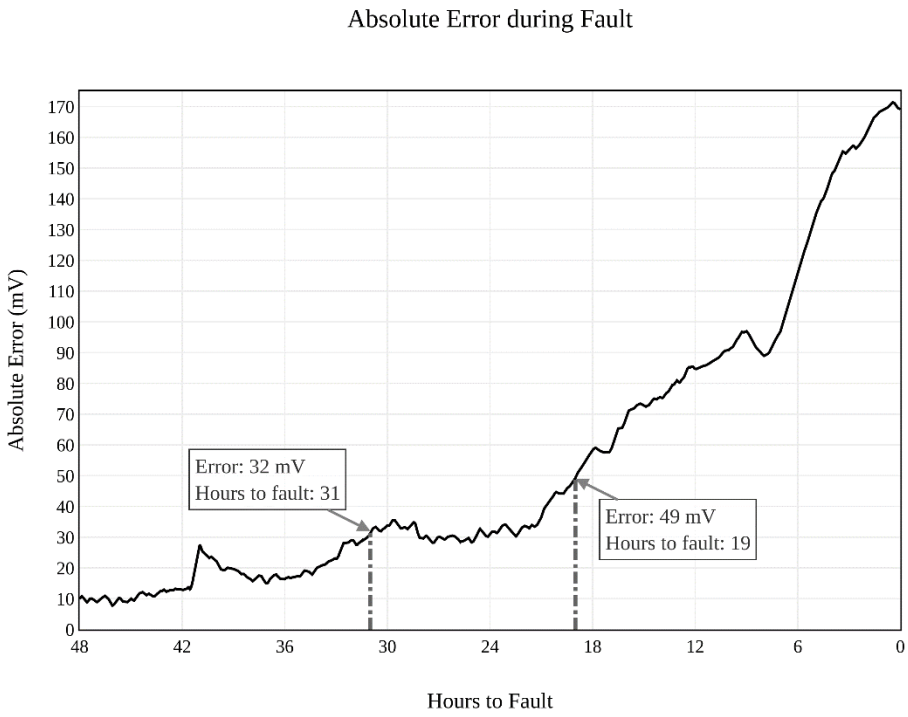

Figure 4.12: Divergence between the predicted voltage and the measured voltage during the 48 hours preceding a cell's fault.

As mentioned in the introduction, the end goal is to detect faults in cells before they happen. The indicator is the divergence between the cell's predicted voltage by a trained model and the cell's measured voltage.

In order to demonstrate that our neural network is appropriate for this task, we used the sequence of a faulty cell that was previously identified by an expert. Figure 4.12 shows how the error between the predicted voltage and the measured voltage increases during the 48 hours that precede the fault. The error increases slowly until it reaches a plateau. It then stays stable for some hours until it starts to increase again with a more pronounced slope. It is desired to detect the fault before the plateau, as that would give the plant's operators more time to react and plan the maintenance. Therefore, the more accurate the model is, the earlier the impending fault can be detected.

We set the fault detection threshold to the average 99th percentile intra-cycle error, as presented in the last row of Table 4.8. In order to minimize the appearance of possible false positives, we add an extra tolerance of 10 mV. This results in a threshold of 32 mV for the neural network and 49 mV for the parametric model. In this case, the network was able to detect the fault 31 hours before it happened, which is a gain of 12 hours compared to the parametric model.

4.6 Conclusion

In this article, we present a new approach for detecting faults in electrochemical cells by using a neural network model composed of an encoder and a decoder. Results show that this approach presents many advantages over expert-defined parametric models currently used for this task. Namely, the key points of our approach are:

1. Better accuracy. Our model is capable of predicting the cell's voltage more accurately. We use a neural network that is not based on suppositions of the underlying chemical function. Instead, it is trained with a substantial amount of data coming from previous electrolyzers. It also takes into account the temporal relations that exist between the observations, thanks to its LSTM layers.
2. Less human intervention required. There is no need for an expert to spend time finding the specific parameters required by each plant. This entails that the model may be deployed to more plants with little effort just by retraining the model, which in turn helps to improve their operational safety.
3. Interpretability. The output of the encoder can be plotted and explained, which helps the user to verify the correct functioning of the neural network. As the safety of the plant and its operators rely on the model, interpretability is very important.
4. Continuously improving method. As more data is collected from different cells and cycles, it is possible to retrain the model to include them. This will help reduce edge cases and the overall error of the neural network.
5. Simple deployment. A single model is valid for multiple cells and cycles without needing retraining once deployed. Therefore, the implementation of the model in the plant does not require extensive computing power.

Thanks to this model, we can confidently say that we are one step closer to a fully automatic management of plant maintenance.

4.7 Future Work

There are, however, some caveats and suggestions that could be further investigated in future models:

- If the encoder does not find a good representation of a cell's startup, the voltage prediction for its operation phase will be incorrect. This happens scarcely, and, with more data available, it should occur even less often. Nevertheless, a safety fallback must be designed for these cases.
- It would be interesting to try to replace the LSTM encoder by an attention-based encoder in order to see if better accuracy is attained (Vaswani, et al., 2017).
- The compression of the neural network is an active field of research that could help in requiring less computing power and memory for making predictions (Polino, Pascanu, & Alistarh, 2018).

4.8 Acknowledgments

We want to thank Mitacs and R2 Inc. for providing funding for this research.

CHAPTER 5 GENERAL DISCUSSION

In the article, we propose a neural network model for fault prediction in electrochemical cells. We aim to replace the expert-defined parametric model currently used in industry for this task.

The highly non-linear nature of neural networks allows to approximate complex relations without requiring expert knowledge, which is a key factor in expanding fault prediction to new industries. Moreover, due to the flexibility that neural networks provide, it is possible to explore novel architectures. Indeed, in our article, we overcome a real-life problem by using a different approach to the concept of neural encoder. Encoders are usually paired with a decoder that reconstructs the output of the encoder to its original input, forming an autoencoder. Instead, we make the encoder learn the degradation of each cell by minimizing the loss of the predicted voltage with respect to the measured voltage. Following this technique, and without giving the network labeled degradation data, the encoder is able to find the appropriate cell's degradation.

The operation of electrolyzers is divided into cycles. We take advantage of this by recalculating the encoding of each cell at the beginning of each cycle. We plot the encoder's result in a graph, and we verify that the position of each cell on it depends on its degradation. More specifically, it depends on its relative degradation compared to the rest of the electrolyzer's cells. Neural networks are notably known for being black-box models. However, by providing a visualization of the encoder's results, it is possible to interpret the decision taken by the network.

Using the encoded vector, which is specific for each cell and cycle, and the operating conditions during the cycle, we predict the cell's voltage. Indeed, we show that the same model can correctly predict the voltage of three different cells, the only difference in the input data being their encoded vector. This allows us to use a single model, making its deployment to multiple electrolyzers easier than the current approach of training a new parametric model per cell and cycle. It also helps to simplify the implementation of a CBM strategy, as less human intervention is required.

Subsequently, we compare the results obtained by our model with those obtained by the parametric model. We describe the different load levels existing during the operation of the electrolyzer. The parametric model fails to predict the voltage for different load levels. In comparison, our model is accurate for all the load levels, resulting in a more stable voltage prediction throughout the cycle.

We are interested in comparing the accuracy between cycles, as well as inside the cycle. We seek accurate results in both cases. In order to be reliable, a model must work correctly for almost the totality of the cells and cycles. Furthermore, the model must also be accurate for all the observations inside a cycle as, the more accurate the model is, the earlier the impending fault can be predicted. Results show that our model is not only more reliable but also more accurate for all the percentiles. Having a good accuracy in the high percentiles helps reducing the number of false positives due to outliers.

Finally, we show the evolution of the error of a trained model during a cell's fault. We see that our network detects the fault before the parametric model. Remembering the *P-F interval* presented in the introduction of this thesis, we can see that our network's point *P* is located to the left of the one of the parametric model. Thus, our model gives more time to the plant's operator to plan the required maintenance actions before the fault occurs.

CHAPTER 6 CONCLUSION AND RECOMMENDATIONS

In this thesis, we have presented the evolution of maintenance until our days. We have explained how Condition-Based Maintenance (CBM) is the most advanced strategy, but also how it requires more involvement and planning than the other strategies. We have introduced different CBM techniques, each of them being appropriated for a different objective depending on the type of data available. Based on that, we have presented fault prediction as a way of implementing a CBM strategy. We have explained how Machine Learning allows us to perform fault prediction. More precisely, we have presented Neural Networks and the theory behind them.

In the article that axes this thesis, we applied fault prediction to electrochemical cells. Unexpected faults in these cells may become safety hazards, so detecting them in advance is of the utmost importance. As previously stated, the cell's voltage is the crucial parameter to detect these faults. Indeed, the indicator of an impending fault is the divergence between the cell's measured voltage and the expected voltage that it would present in a healthy state. We used a neural network to estimate this expected voltage accurately. Our model showed smaller errors than the currently used expert-defined model, allowing us to detect the fault earlier. Also, the fact that our model does not require human intervention opens the door to a more automatic management of the plant's maintenance.

The most important limitation of our approach is that our method relies on a variable that responds differently to operating conditions when there is an impending fault. Although this is usually the case, it reduces the applicability of our system. Moreover, even if our approach does not require expert knowledge for modeling the variable's response, it still needs it in order to determine which variables to monitor. If we try to apply our approach to a new process without having expert knowledge, we would likely need to determine these variables beforehand by using statistics or by developing another model.

Another limitation is that we specifically tailored our approach for the use in electrolyzers, with all the constraints in the data that they present. This entails that there are probably simpler solutions if the data does not present the limitations that we had – mainly the fact that only the voltage was different for each cell, and that it could not be used for the prediction.

Finally, it would be interesting to test our approach with industries that use similar production processes and have similar constraints, like for example aluminum, which is produced employing electrochemical cells with the *Hall-Héroult* process (Thonstad, et al., 2001).

REFERENCES

Ben-Daya, M. (2000). You may need RCM to enhance TPM implementation. *Journal of Quality in Maintenance Engineering, 6*(2), 82-85.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks, 5*(2), 157-166.

Berndt, D., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. *KDD workshop*, *10*, pp. 359-370.

Bhullar, R. (1993). Strategies for implementing advanced process controls in a distributed control system (DCS). *ISA Transactions, 32*(2), 147-156.

Blann, D. (2013). Maximizing the P-F Interval Through Condition-Based Maintenance. *Maintworld, Magazine for Maintenance & Asset Management Professional*.

Cassell, D. (1985). *US Patent No. 4,517,637.*

Causserand, C., & Aimar, P. (2010). Characterization of Filtration Membranes. *Comprehensive Membrane Science and Engineering*, 311-335.

Celebi, M. E., & Aydin, K. (2016). *Unsupervised learning algorithms.* Springer.

Cucker, F., & Smale, S. (2002). Best choices for regularization parameters in learning theory: on the bias-variance problem. *Foundations of computational Mathematics, 2*(4), 413-428.

Cutler, C. (1995). An Industrial Perspective on the Evolution of Control Technology. In C. Cutler, *Methods of Model Based Process Control* (pp. 643-658). Springer Netherlands.

Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

Encyclopedia of Mathematics. (2013). *Leibniz Rule*. Retrieved 10 30, 2019, from http://www.encyclopediaofmath.org/index.php?title=Leibniz_rule&oldid=30941

Fernández-Francos, D., Martínez-Rego, D., Fontenla-Romero, O., & Alonso-Betanzos, A. (2013). Automatic bearing fault diagnosis based on one-class v-SVM. *Computers & Industrial Engineering, 64*(1), 357-365.

Fosdick, H. (1980). The Microcomputer Revolution. *Library Journal, 105*(13), 1467-72.

Fuller, W. A. (2009). Moving Average and Autoregressive Processes. In *Introduction to statistical time series.* John Wiley & Sons.

Gensler, A., Henze, J., Sick, B., & Raabe, N. (2016). Deep Learning for solar power forecasting — An approach using AutoEncoder and LSTM Neural Networks. *2016 IEEE international conference on systems, man, and cybernetics (SMC)* (pp. 2858 - 2865). IEEE.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, (pp. 249-256).

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning.* MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, (pp. 2672-2680).

Google. (2019). *TFRecord | TensorFlow Core r2.0*. Retrieved 10 07, 2019, from https://www.tensorflow.org/tutorials/load_data/tfrecord

Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. *International Workshop on Artificial Neural Networks*, (pp. 195-201).

Hastie, T., Robert, T., & Friedman, J. H. (2009). Overview of Supervised Learning. In *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (2 ed.). New York: Springer.

Hill, D. (1996). *A history of engineering in classical and medieval times.* Routledge.

Hinton, G. E., & Zemel, R. S. (1994). Autoencoders, minimum description length and Helmholtz free energy. *Advances in neural information processing systems*, (pp. 3-10).

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504-507.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735-1780.

Holweg, M. (2007). The genealogy of lean production. *Journal of Operations Management, 25*(2), 420-437.

JA1011, S. (1999). Evaluation criteria for reliability-centered maintenance (RCM) processes. *Society for Automotive Engineers*.

Jalali, A., Mohammadi, F., & Ashrafizadeh, S. (2009). Effects of process conditions on cell voltage, current efficiency and voltage balance of a chlor-alkali membrane cell. *Desalination, 237*(1-3), 126-139.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2014). *An Introduction to Statistical Learning: With Applications in R.* Springer.

Karpathy, A. (2015). *The unreasonable effectiveness of recurrent neural networks*. Retrieved 09 20, 2019, from Andrej Karpathy Blog: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal, 37*(2), 233-243.

Landels, J. (2000). *Engineering in the ancient world.* University of California Press.

Lasi, H., Fettke, P., Kemper, H.-G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. *Business & information systems engineering, 6*(4), 239-242.

LeCun, Y., Bottou, L., Orr, G., & Müller, K.-R. (2012). Efficient backprop. In *Neural networks: Tricks of the trade* (pp. 9-48). Springer.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *arXiv preprint arXiv:1607.00148*.

Masci, J., Meier, U., Cireşan, D., & Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction. *International Conference on Artificial Neural Networks* (pp. 52 - 59). Springer.

Mazurkiewicz, D. (2015). Maintenance of belt conveyors using an expert system based on fuzzy logic. *15*(2), 412-418.

Mueller-Hillebrand, B. (1988). *German tank maintenance in World War II.* Washington, D.C.: Center of Military History, United States Army.

Nair, V., & Hinton, G. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th international conference on machine learning (ICML-10)*, (pp. 807-814).

Nakajima, S. (1988). *Introduction to TPM: Total Poductive Maintenance.* Productivity Press.

Nielsen, M. (2015). *Neural Networks and Deep Learning* (Vol. 25). San Francisco, CA, USA: Determination press.

Nostrand, W., & Moore, L. (1943). *The necessity of oil filters in preventive maintenance.* No. 430096. SAE Technical Paper.

Nowlan, F., & Heap, H. (1978). *Reliability-Centered Maintenance.* United Air Lines Inc San Francisco Ca.

Olah, C. (2015). *Understanding LSTM Networks*. Retrieved 10 12, 2019, from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Paidar, M., Fateev, V., & Bouzek, K. (2016). Membrane electrolysis. History, current status and perspective. *Electrochimica Acta, 209*, 737-756.

Peng, X., Zhou, C., Hepburn, D., Judd, M., & Siew, W. (2013). Application of K-Means method to pattern recognition in on-line cable partial discharge monitoring. *IEEE Transactions on Dielectrics and Electrical Insulation, 20*(3), 754-761.

Penny, W., & Harrison, L. (2007). Multivariate autoregressive models. *Statistical Parametric Mapping: The Analysis of Functional Brain Images. Academic Press, Amsterdam*, 534-540.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta numerica, 8*, 143-195.

Polino, A., Pascanu, R., & Alistarh, D. (2018). Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*.

Ragab, A., Yacout, S., Ouali, M., & Osman, H. (2015). Multiple failure modes prognostics using logical analysis of data. *Proceedings - Annual Reliability and Maintainability Symposium. 2015-May.* Institute of Electrical and Electronics Engineers Inc.

Raghav, R., Lemaitre, G., & Unterthiner, T. (2018). *Compare the effect of different scalers on data with outliers*. Retrieved 09 28, 2019, from https://scikit-learn.org/stable/auto_examples/preprocessing/plot_all_scaling.html

Remy, P. (2016). *Stateful LSTM in Keras*. Retrieved 10 21, 2019, from http://philipperemy.github.io/keras-stateful-lstm/

Roberts, A., Engel, J., & Eck, D. (2017). Hierarchical Variational Autoencoders for Music. *NIPS Workshop on Machine Learning for Creativity and Design*, (p. 6).

Rohner, P. (1996). *PLC: automation with programmable logic controllers : a textbook for engineers and technicians.* UNSW Press.

Rumelhart, D., Hinton, G., & Williams, R. (1985). *Learning internal representations by error propagation.* No. ICS-8506. California Univ San Diego La Jolla Inst for Cognitive Science.

Sakurada, M., & Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (pp. 4 - 12). ACM.

Seema Singh. (2018). *Understanding the Bias-Variance Tradeoff*. Retrieved 09 25, 2019, from https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229

Shojai Kaveh, N., Ashrafizadeh, S., & Mohammadi, F. (2008). Development of an artificial neural network model for prediction of cell voltage and current efficiency in a chlor-alkali membrane cell. *Chemical Engineering Research and Design, 86*(5), 461-472.

Shojai Kaveh, N., Mohammadi, F., & Ashrafizadeh, S. (2009). Prediction of cell voltage and current efficiency in a lab scale chlor-alkali membrane cell based on support vector machines. *Chemical Engineering Journal, 147*(2-3), 161-172.

Singh, A., & Palod, R. (2018). Sentiment Transfer using Seq2Seq Adversarial Autoencoders. *arXiv preprint arXiv:1804.04003*.

Stout, T., & Williams, T. (1995). Pioneering Work in the Field of Computer Process Control. *IEEE Annals of the History of Computing, 17*(1), 6-18.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Taylor, L., & Nitschke, G. (2017). Improving Deep Learning using Generic Data Augmentation. *arXiv preprint arXiv:1708.06020*.

The Chlorine Institute. (2018). *Electrical Safety in Chlor-Alkali Cell Facilities* (6 ed.).

Thonstad, J., Fellner, P., Haarberg, G. M., Hives, J., Kvarde, H., & Sterten, A. (2001). *Aluminium Electrolysis: Fundamentals of the Hall-Herault Process.* Aluminium-Verlag.

Tremblay, G., Lademann, H., Simard, G., Veillette, M., & Berriah, S. (2012). *US Patent No. 8,152,987.*

Tsang, A. (1995). Condition-based maintenance: Tools and decision making. *Journal of Quality in Maintenance Engineering, 1*(3), 3-17.

Tschannen, M., Bachem, O., & Lucic, M. (2018). Recent Advances in Autoencoder-Based Representation Learning. *arXiv preprint arXiv:1812.05069*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A., . . . Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems. 2017-December*, pp. (5999-6009). Neural information processing systems foundation.

Veillette, M., Berriah, S., & Tremblay, G. (2010). *US Patent No. 7,818,276.*

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, (pp. 1096-1103).

Vitruvius Pollio, & Granger, F. (1931). *On Architecture.* Harvard University Press.

Wei, X., Verhaegen, M., & van Engelen, T. (2010). Sensor fault detection and isolation for wind turbines based on subspace identification and Kalman filter techniques. *International Journal of Adaptive Control and Signal Processing, 24*(8), 687-707.

Weiss, K., Khoshgoftaar, T., & Wang, D. (2016). A survey of transfer learning. *Journal of Big Data, 3*(1).

Wendt, H., & Kreysa, G. (1999). *Electrochemical engineering: science and technology in chemical and other industries.* Springer Science & Business Media.

Werbos, P., & others. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE, 78*(10), 1550-1560.

Willis, M., & Tham, M. (1994). Advanced process control. *Department of Chemical and Process Engineering, University of Newcastle Upon Tyne, UK*.

Wireman, T. (1994). *Computerized maintenance management systems.* Industrial Press.

Zhao, F., Feng, J., Zhao, J., Yang, W., & Yan, S. (2017). Robust lstm-autoencoders for face de-occlusion in the wild. *IEEE Transactions on Image Processing, 27*(2), 778 - 790.

Zhou, C., & Paffenroth, R. C. (2017). Anomaly detection with robust deep autoencoders. *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 665 - 674). ACM.