

Faculté des sciences

TIME SERIES FORECASTING: COMPARISON OF BOX-JENKINS, HYBRID AND MACHINE LEARNING METHODS

AUTHOR: GUEULETTE ARNAUD

SUPERVISOR: RAINER VON SACHS

READER: DONATIEN HAINAUT

ACADEMIC YEAR 2022 - 2023

MASTER [120] EN SCIENCE DES DONNÉES, ORIENTATION STATISTIQUE

Acknowledgments

First of all, I would like to thank all the people: my family, my friends, and my colleagues who have supported me during my studies and the writing of this thesis.

By keeping an eye on my progress, you have helped me a lot to keep going.

I would also like to thank all the teachers who got me interested in applied statistics and led me to what I am and do today: Prof. Johan Segers, Prof. Christian Hafner, Prof. Donatien Hainaut, and Prof. Rainer von Sachs. Thank you for your passion and dedication to teaching theoretical and applied statistics. I am grateful to you, without you I would not have chosen the path of statistics and data science.

I would like to extend my thanks to Prof. Rainer von Sachs and Prof. Donatien Hainaut. By taking an interest in this thesis and my work, you have strengthened my belief that it was worth doing and will help to move things forward. Thank you very much.

Finally, many thanks to Professor Rainer von Sachs. Thank you for believing in me and allowing me to write this thesis with all the constraints and problems I encountered. It was not easy, but we did it. Thank you for that.

Mum, I did this for you. It is my achievement, but it is also yours.

Sincerely,

GUEULETTE Arnaud

*

This thesis has been submitted in partial fulfillment of the requirements for the master degree in data science (statistics orientation) delivered by the Université Catholique de Louvain (Louvain-la-Neuve, Belgium).

Contents

1	Introduction	1
1.1	Introduction	1
1.2	Structure of the Thesis	2
2	Literature Review on Time Series Forecasting	4
3	Purpose of the Thesis	11
3.1	Critics of Forecasting Competitions	11
3.2	Research Question	12
4	Theoretical Background	14
4.1	Models	14
4.1.1	Seasonal Autoregressive Integrated Moving Average (SARIMA)	14
4.1.2	Multilayer Perceptron (MLP)	16
4.1.3	Recurrent Neural Network (RNN)	19
4.1.4	Long Short-Term Memory (LSTM)	22
4.1.5	Exponential Smoothing Recurrent Neural Network (ES-RNN)	27
4.2	Theoretical Foundations	35
4.2.1	Time Series Problem - Supervised Learning Problem	35
4.2.2	Activation Functions	36
4.2.3	Optimization Algorithms	39
4.2.4	Accuracy Measures	42
4.2.5	Time Series Validation Procedures	44
4.3	Conclusion	47
5	Experimental Settings	48
5.1	Libraries	48
5.2	Datasets	48
5.2.1	Mauna Loa's CO2 emissions between 1965 and 1980	49
5.2.2	Electricity production in Germany between 1955 and 1980	50
5.2.3	Iron production in Australia between March 1956 and September 1994	51
5.2.4	Minimum daily temperatures over 5 years (1986-1990) in the city of Melbourne, Australia	52
5.3	Training, Validation and Testing	53

5.4	Benchmarks	54
5.5	Conclusion	59
6	Study Results	60
6.1	Models	60
6.1.1	Multilayer Perceptron	61
6.1.2	Long Short-Term Memory	62
6.1.3	Exponential Smoothing Recurrent Neural Network	64
6.2	Results	65
6.2.1	Validation Set Results	65
6.2.2	Test Set Results	66
6.3	Conclusion	76
7	Study Conclusion	77
7.1	Study Conclusion	77
7.2	Research Question	78
8	Conclusion & Discussion	79
8.1	Summary	79
8.2	Future Works	80
A	SARIMA coefficients	84
A.1	SARIMA - Coefficients - CO2 Dataset	84
A.2	SARIMA - Coefficients - Electricity Dataset	85
A.3	SARIMA - Coefficients - Iron Production Dataset	86
A.4	SARIMA - Coefficients - Daily min. temperature Dataset	87
B	Sensitivity Analysis	88
B.1	Multilayer Perceptron - CO2 Dataset	88
B.2	Multilayer Perceptron - Electricity Dataset	89
B.3	Multilayer Perceptron - Iron Production Dataset	90
B.4	Multilayer Perceptron - Daily min. temperature Dataset	91
B.5	Long Short-Term Memory - CO2 Dataset	92
B.6	Long Short-Term Memory - Electricity Dataset	93
B.7	Long Short-Term Memory - Iron Production Dataset	94
B.8	Exponential Smoothing Recurrent Neural Network - CO2 Dataset	95
B.9	Exponential Smoothing Recurrent Neural Network - Electricity Dataset	96
B.10	Exponential Smoothing Recurrent Neural Network - Iron Production Dataset	97
B.11	Exponential Smoothing Recurrent Neural Network - Daily min. temperature Dataset	98

Chapter 1

Introduction

1.1 Introduction

The field of forecasting, largely driven by the increasing accessibility of data, computing power, and advanced modelling techniques, has come a long way since its first attempts in the 1950s. In particular, as more and more companies have adopted data-driven strategies, time series forecasting has become increasingly important to improve their performance and, therefore, their long-term survival.

Although practitioners and researchers have contributed greatly to the development of the field, another important contribution has come from forecasting competitions. By providing a collaborative platform for researchers, practitioners, and data enthusiasts to benchmark, evaluate, and improve forecasting models, these competitions have revolutionized the field and paved the way for the development of new techniques and methodologies.

In addition, by bringing together a wide range of participants with different backgrounds and expertise, these competitions encourage knowledge sharing and collaboration. Participants are exposed to a variety of forecasting techniques, novel approaches, and real-world forecasting challenges. The exchange of ideas and insights stimulates innovation and contributes to the development of best practices in the field.

In the early days of forecasting, there was a strong belief that simpler methods were better. Over time, however, forecasting competitions have helped to change this, proving that complex methods can be useful and efficient for both theoretical and practical applications. The results and lessons learned from these competitions have not only advanced the understanding of forecasting methods, but also guided the selection and implementation of appropriate models in practice.

This thesis mainly discusses forecasting competitions and how they contribute to the field of forecasting. It also discusses the winner of the M4 competition, a hybrid model that blends statistical and deep learning concepts.

The model, called Exponential Smoothing Recurrent Neural Network (ES-RNN), was proposed by Slawek Smyl, a data scientist at Uber (Redd et al., 2019; Smyl, 2020). This hybrid model outperformed all other models in the M4 competition, pointing the way for further development and research into more complex methods (Makridakis & Petropoulos, 2020; Makridakis et al., 2020).

1.2 Structure of the Thesis

Literature Review on Time Series Forecasting. This chapter covers the history of forecasting from its origins. From the first attempts in the 1950s to 2020, the chapter aims to provide a broad overview of the forecasting literature and, in the specific context of this thesis, of forecasting competitions. By the end of the chapter, the reader will have a clear overview of the forecasting competitions that have taken place up to 2020 and the contributions they have made to the field of forecasting.

Purpose of the Thesis. Following on from the previous chapter, this chapter specifically discusses forecasting competitions and the reasons for writing this thesis. By the end of this chapter, the reader should have a clear understanding of where the thesis is going and how it will get there.

Theoretical Background. While the previous two chapters were mainly literature reviews, comments, and observations, this chapter focuses more on the theoretical aspects of the thesis. All the technical concepts necessary to fully understand the study presented later are provided in this chapter. From the models to their training, validation, and testing, this chapter provides the reader with all the necessary components to fully appreciate and understand the results discussed in the following chapters.

Experimental Settings. This chapter describes all the settings and components (libraries, datasets, benchmarks, etc.) used to perform and obtain the results discussed in the following chapter. It also allows the reader to have a clear understanding of the context in which the study was carried out.

Study Results. This section presents all the results observed during the study. From the benchmarks to the deep learning and hybrid models, this section discusses all the results and analyses observed and performed in the study.

Study Conclusion. The final chapter discusses the results of the models and whether or not the study of the thesis is consistent with what has been written in the literature. In addition, the chapter concludes by addressing and answering the research questions that we have defined.

Chapter 2

Literature Review on Time Series Forecasting

Although the first attempts at forecasting were made at the end of the nineteenth century, the field of time series forecasting experienced a major revival with the publication of the Single Exponential Smoothing (SES) method in the early 1950s (R. Hyndman & Athanasopoulos, 2018; Majid & Mir, 2018). This method, proposed by Robert Goodell Brown in 1956, attracted attention for its simplicity, minimal computational requirements, and effectiveness in forecasting time series (R. Hyndman & Athanasopoulos, 2018; Majid & Mir, 2018). However, it quickly became apparent that the SES method had limitations in dealing with complex and diverse datasets, partly due to the increasing and varying complexity of the data.

In response to these limitations, more complex methods have been developed, including the two-parameter approach of Charles C. Holt in 1957 (R. Hyndman & Athanasopoulos, 2018; Majid & Mir, 2018). This method extended the SES method by incorporating an additional smoothing parameter for the slope, allowing the method to account for data with trends. Soon after, in 1960, the Holt-Winters (HW) approach was published and adopted by practitioners (R. Hyndman & Athanasopoulos, 2018; Majid & Mir, 2018). This model assumes the presence of an additive or multiplicative seasonal component in the data in conjunction with the trend smoothing parameter of the two-parameter approach (R. Hyndman & Athanasopoulos, 2018).

Despite the considerable advances made by the Holt's two-parameter model in 1957 and the Holt-Winters model in 1960, there was a need for more comprehensive and flexible time series forecasting techniques that could accommodate a variety of data structure models. This concern led to the introduction of the Box-Jenkins method in the 1970s (Box et al., 2015).

The Box-Jenkins method, better known as the Autoregressive Integrated Moving Average (ARIMA) method, was designed to overcome the limitations of previous forecasting techniques by providing a more systematic and robust procedure for model identification, estimation, and validation (Box et al., 2015; R. Hyndman & Athanasopoulos, 2018).

The Box-Jenkins methodology incorporates several key innovations (Box et al., 2015). These innovations include a means of identifying the model by examining the autocorrelation and partial autocorrelation structures of the data, an iterative process to ensure that the final model is parsimonious and accurately represents the underlying process of data generation, and residual analysis to detect residual patterns or dependencies not accounted for by the model (Box et al., 2015).

In addition, it became imperative to deal more effectively with seasonal patterns in the data using appropriate methods, as an increasing number of time series exhibited seasonal components. The Seasonal Autoregressive Integrated Moving Average (SARIMA) model, therefore, seemed a logical development to meet this need (Box et al., 2015; Brockwell & Davis, 2016). The SARIMA framework extends the original ARIMA framework by incorporating seasonal components, thereby enhancing its ability to capture and forecast time series with seasonal variations more accurately (Box et al., 2015; Brockwell & Davis, 2016; R. Hyndman & Athanasopoulos, 2018).

The SARIMA model combines non-seasonal autoregressive (AR), differencing (I), and moving average (MA) components with their seasonal (S) parts, resulting in a comprehensive and versatile forecasting technique (R. Hyndman & Athanasopoulos, 2018). By incorporating seasonal components, the SARIMA allows time series data to be decomposed into non-seasonal and seasonal elements which can be modelled separately and then recombined to produce accurate forecasts (Brockwell & Davis, 2016; R. Hyndman & Athanasopoulos, 2018). This additional layer of complexity provided by the SARIMA model enhances its ability to deal with a wider range of time series problems, particularly those that exhibit seasonality (Brockwell & Davis, 2016; R. Hyndman & Athanasopoulos, 2018).

During the late 1970s and 1980s, the field of time series forecasting experienced substantial progress in both theoretical and practical aspects (De Gooijer & Hyndman, 2006). A variety of new methods and techniques were introduced to overcome the limitations of earlier forecasting methods and to cope with the increasing complexity and diversity of datasets. For example, methods such as the Kalman Filter (KF) and Vector Autoregression (VAR) were developed during this period (De Gooijer & Hyndman, 2006).

At the same time, time series forecasting competitions, which were not so widely organized in the past, have grown as a means of assessing and comparing the performance of different forecasting methods on real datasets (De Gooijer & Hyndman, 2006). These competitions allow participants to receive valuable feedback on their forecasting methods, identify areas for improvement, and encourage the development of new and innovative forecasting techniques (De Gooijer & Hyndman, 2006).

Famous forecasting competitions such as the M competitions initiated by Spyros Makridakis, the Neural Network (NN) or the Tourism Forecasting Competition have served as platforms for researchers and practitioners to evaluate and compare the effectiveness of forecasting methods (De Gooijer & Hyndman, 2006).

Throughout the various forecasting competitions, several key observations and insights have been made that have shaped the understanding and development of time series forecasting methods (De Gooijer & Hyndman, 2006). Some of the observations made from the first M-competition in the early 1980s to the most recent include the following:

- M1 competition (1982): Based on 1001 time series from a wide range of fields, the results of the competition showed that simpler methods, such as exponential smoothing models, often performed as well or better than statistically sophisticated or complex approaches, such as the Box-Jenkins ARIMA model (De Gooijer & Hyndman, 2006). The competition highlighted the importance of model selection and underlined that greater sophistication does not always guarantee superior predictions (De Gooijer & Hyndman, 2006).
- M2 competition (1993): Although this second competition was based on only 29 time series, it confirmed the results of the first: simple methods remained accurate predictors (De Gooijer & Hyndman, 2006). It also showed that combining forecasts from several models could improve forecast accuracy, paving the way for the development of ensemble methods (De Gooijer & Hyndman, 2006).
- M3 competition (2000): The M3 competition used a larger and more diverse dataset (3003 time series) than the previous two competitions to evaluate the accuracy of different forecasting methods. The results of the competition confirmed what the previous competitions had already shown, namely the effectiveness of simpler methods and the power of combining forecasts through ensemble models (Makridakis & Hibon, 2000; De Gooijer & Hyndman, 2006). It also demonstrated that domain knowledge and expert judgment can improve predictions when applied carefully (Makridakis & Hibon, 2000).

In the early 2000s, forecasting theorists and practitioners began to embrace machine and deep learning forecasting techniques. As computing power became more affordable, these methods, which had been left out of the race until technological advances, regained interest in the forecasting community. They were then included in forecasting competitions to see if they could produce better forecasts than the methods commonly used at the time (Makridakis & Hibon, 2000; Crone et al., 2011).

Although still new to the field of forecasting, promising results for sophisticated models were observed and highlighted during the NN3 (Neural Network 3) competition in 2006 (Crone et al., 2011). While the results of the M competitions showed that well-established statistical methods generally outperformed sophisticated ones (Makridakis & Hibon, 2000), the conclusions of the NN3 competition, which was a replication of the M3 competition with an extension to neural network and computational intelligence models, were slightly different. Indeed, one of the results of the NN3 competition showed that neural network and CI methods can compete with, but not outperform, established statistical methods in time series forecasting (Crone et al., 2011). This observation, which was the first of many, paved the way for the use of neural network models in time series forecasting (Crone et al., 2011).

Eleven years later, in 2017, the M4 competition took place. It is still the largest forecasting competition to date. While previous M competitions did not introduce much high-frequency data: weekly, daily, and hourly, the M4 competition combined both high-frequency and low-frequency data: annual, quarterly, and monthly to provide a wide range of problems on which forecasting methods can be evaluated (Makridakis et al., 2020).

The M4 competition was also the first one of the M contests to include a Prediction Interval (PI) evaluation part (Makridakis et al., 2020). Unlike previous M competitions, the M4 competition consisted of much longer series, providing more opportunities for more complicated methods that require large amounts of data to be well trained (Makridakis et al., 2020).

During the competition, more than 61 forecast methods were proposed by 248 contestants from 46 countries for both the Point Forecast (PF) and Prediction Interval (PI) challenges. Only 49 of the 248 contestants submitted valid PFs for all 100,000 time series. For the PI challenge, only 20 forecasters submitted valid PIs for all 100,000 time series along with their PFs (Makridakis et al., 2020).

In previous competitions, combination methods have been discussed as a way to improve prediction accuracy, and the results of the M4 competition provided further insight into such methods. Indeed, all the best-performing methods, both in terms of PFs and PIs, were either pure statistical models or combinations of mostly statistical models (Makridakis & Petropoulos, 2020; Makridakis et al., 2020).

The conclusions of the competition showed that no single method can effectively capture time series patterns, while a combination of methods, each capturing a different component of such patterns, is more accurate because the errors of the individual models are canceled out by averaging (Makridakis & Petropoulos, 2020; Makridakis et al., 2020). In addition, unlike previous competitions, the M4 competition showed that more complex prediction methods lead to better performance. Previous competitions had suggested the opposite (Makridakis et al., 2020).

Surprisingly, the winner of the M4 competition was a hybrid model that used both statistical and machine learning features. The model, based on a combination of the Exponential Smoothing (ES) and Recurrent Neural Network (RNN) models, was presented by Slawek Smyl, a senior data scientist at Uber Technologies (Smyl, 2020). The model was declared the winner of the competition because it was 9.4% (sMAPE, see point 4.2.4.6) more accurate than the benchmark used in the M4 competition, as shown in Figure 2.1. The benchmark, which was a combination model, was a simple arithmetic mean of the Single, Holt, and Damped Exponential Smoothing models (Makridakis et al., 2020).

The performances of the 17 methods submitted to M4 that had OWAs at least as good as that of the Comb.

Type	Author(s)	Affiliation	sMAPE ^c				OWA ^d				Rank ^d				% improvement of method over the benchmark
			Yearly (23k)	Quarterly (24k)	Monthly (48k)	Others (5k)	Average (100k)	Yearly (23k)	Quarterly (24k)	Monthly (48k)	Others (5k)	Average (100k)	19	sMAPE	OWA
Benchmark for methods ^b															
Hybrid Combination	Smyl, S.	Uber Technologies	13.176	9.679	12.126	4.014	11.374	0.867	0.890	0.920	1.039	0.898	1	9.4%	8.6%
	Montero-Manso, P., Talagala, T., Hyndman, R. J. & Athanasopoulos, G.	University of A Coruña & Monash University	13.528	9.733	12.639	4.118	11.720	0.778	0.847	0.836	0.920	0.821	1	9.4%	8.6%
Combination	Pawlikowski, M., Chorowska, A. & Yanchuk, O.	ProLogistica Soft	13.943	9.796	12.747	3.365	11.845	0.820	0.855	0.867	0.742	0.841	3	5.7%	6.3%
	Jaganathan, S. & Prakash, P.	Individual	13.712	9.809	12.487	3.879	11.695	0.813	0.859	0.854	0.882	0.842	4	6.8%	6.2%
Combination	Fiorucci, J. A. & Louzada, F.	University of Brasilia & University of São Paulo	13.673	9.816	12.737	4.432	11.836	0.802	0.855	0.868	0.935	0.843	5	5.7%	6.1%
	Petropoulos, F. & Svetunkov, I.	University of Bath & Lancaster University	13.669	9.800	12.888	4.105	11.887	0.806	0.853	0.876	0.906	0.848	6	5.3%	5.6%
Combination	Shaub, D.	Harvard Extension School	13.679	10.378	12.839	4.400	12.020	0.801	0.908	0.882	0.978	0.860	7	4.3%	4.2%
	Legaki, N. Z. & Koutsouri, K.	National Technical University of Athens	13.366	10.155	13.002	4.682	11.986	0.788	0.898	0.905	0.989	0.861	8	4.5%	4.1%
Combination	Doornik, J., Castle, J. & Hendry, D.	University of Oxford	13.910	10.000	12.780	3.802	11.924	0.836	0.878	0.881	0.874	0.865	9	5.0%	3.7%
	Pedregal, D.J., Trapero, J. R., Villegas, M. A. & Madrigal, J. J.	University of Castilla-La Mancha	13.821	10.093	13.151	4.012	12.114	0.824	0.883	0.899	0.895	0.869	10	3.5%	3.2%
Statistical	Spiliotis, E. & Assimakopoulos, V.	National Technical University of Athens	13.804	10.128	13.142	4.575	12.148	0.823	0.889	0.907	0.975	0.874	11	3.2%	2.7%
	Roubinchtein, A.	Washington State Employment Security Department	14.445	10.172	12.911	4.436	12.183	0.850	0.885	0.881	0.992	0.876	12	3.0%	2.4%
Other Combination	Ibrahim, M.	Georgia Institute of Technology	13.677	10.089	13.321	4.747	12.198	0.805	0.890	0.921	1.079	0.880	13	2.8%	2.0%
	Kull, M., et al.	University of Tartu	14.096	11.109	13.290	4.169	12.496	0.820	0.960	0.932	0.883	0.888	14	0.5%	1.1%
Combination	Waheeb, W.	Universiti Tun Hussein Onn Malaysia	14.783	10.059	12.770	4.039	12.146	0.880	0.880	0.927	0.904	0.894	15	3.3%	0.4%
	Darin, S. & Stellwagen, E.	Business Forecast Systems (Forecast Pro)	14.663	10.155	13.058	4.041	12.279	0.877	0.887	0.887	1.011	0.895	16	2.2%	0.3%
Combination	Dantas, T. & Oliveira, F.	Pontifical Catholic University of Rio de Janeiro	14.746	10.254	13.462	4.783	12.553	0.866	0.892	0.914	1.011	0.896	17	0.0%	0.2%
	Trotta, B.	Individual	14.397	11.031	13.973	4.566	12.894	0.859	0.939	0.941	0.991	0.915	23	-2.7%	-1.9%
2nd Best ML	Bontempi, G.	Université Libre de Bruxelles	16.613	11.786	14.800	4.734	13.990	1.050	1.072	1.007	1.051	1.045	37	-11.4%	-16.4%

^a Rank: rank calculated considering both the submitted methods (50) and the set of benchmarks (10).

^b Com: the benchmark is the average (combination) of Simple, Holt and Damped exponential smoothing.

^c sMAPE: symmetric mean absolute percentage error.

^d OWA: overall weighted average of the relative sMAPE and the relative MASE.

Figure 2.1: Adapted from "The M4 Competition: 100,000 time series and 61 forecasting methods" by S. Makridakis, E. Spiliotis, and V. Assimakopoulos, 2018, International Journal of Forecasting, 36(1), p. 804. Copyright 2018 by Elsevier B.V.

The M4 competition provided empirical evidence of progress in the use of machine learning and deep learning techniques to improve forecasting (Gilliland, 2020). While the previous competitions suggested slightly different conclusions, the M4 competition, thanks to its variety of domain-specific time series, confirmed that combinations of methods and hybrid models were, and still are, the next prospects to be investigated to improve forecast accuracy (Makridakis et al., 2020).

Chapter 3

Purpose of the Thesis

3.1 Critics of Forecasting Competitions

Although the M4 competition and forecasting competitions, in general, have led to advances in the field of forecasting, particularly in the use of machine learning and deep learning techniques to improve forecasting accuracy, there have been and continue to be some concerns about the use of datasets from forecasting competitions as benchmarks for evaluating established forecasting models (Spiliotis et al., 2020).

The continued use of such datasets has led to a general belief that they represent the real world and lead to reliable conclusions that will hold up under real-world conditions (Spiliotis et al., 2020). In addition, their use as benchmarks for evaluating forecasting accuracy models has led current forecasting research to focus on improving, and consequently over-fitting, methods to specific data samples rather than solving real-world problems (Spiliotis et al., 2020).

Indeed, since forecasting competitions essentially judge forecasting models on a selected set of time series data, the models are fitted, and unfortunately over-fitted, to that specific set of data (Makridakis & Petropoulos, 2020). This setup limits the ability to extend the conclusions of such competitions to other situations or environments (Makridakis & Petropoulos, 2020).

In addition, because these competitions use carefully selected metrics to evaluate forecasting models, the evaluations are unlikely to capture all aspects of the models (Makridakis et al., 2020; Koutsandreas et al., 2021). These competitions neglect relevant considerations such as model interpretability, computational efficiency or robustness to different types of time series in their evaluations (Makridakis et al., 2000; Makridakis & Hibon, 2000; Crone et al., 2011; Makridakis et al., 2020). By omitting these criteria, forecasting competitions simply provide an answer to the question of which model is best in a given setting, without necessarily providing advanced arguments to support this claim (Makridakis & Hibon, 2000; Crone et al., 2011; Makridakis & Petropoulos, 2020; Makridakis et al., 2020).

Finally, as each forecasting competition has its own specific benchmark datasets, the winners of different competitions cannot be directly compared, as they were evaluated on different benchmarks (Makridakis & Hibon, 2000; Crone et al., 2011; Makridakis & Petropoulos, 2020).

While forecasting competitions have contributed and continue to contribute to the development of the field of forecasting by encouraging the development of new forecasting techniques, they lack advanced analysis of how the best models perform in different environments, settings, and domain applications (Spiliotis et al., 2020).

3.2 Research Question

Since the performance of the Exponential Smoothing Recurrent Neural Network (ES-RNN) model over the 100,000 time series used as benchmarks for the M4 competition was the best of all the competing models, the ES-RNN was declared the winner of the competition. Does this mean that it beats all forecasting models in all circumstances? Not necessarily (Makridakis & Petropoulos, 2020; Makridakis et al., 2020).

The M4 competition, by averaging forecast scores from 100,000 different time series without an in-depth analysis of the forecasts provided, neglected relevant information potentially useful for real-world applications and forecasting research: Why does the ES-RNN outperform all other models? Does the ES-RNN perform well in all situations? What is the accuracy of the ES-RNN model on low-frequency time series? on high-frequency time series? All these questions, which are relevant for forecasting research and practice, were not discussed much in the M4 competition results. Only general conclusions, findings, and results noted during the competition were highlighted (Makridakis et al., 2018, 2020).

This thesis aims to fill a gap in forecasting research by investigating the performance of Smyl's Exponential Smoothing Recurrent Neural Network (ES-RNN) model specifically on trend and/or seasonal time series data, which has received limited attention in existing studies. Using four selected time series with different characteristics, the thesis aims to investigate two main aspects:

- Identify factors that contribute to the superior performance of the Exponential Smoothing Recurrent Neural Network (ES-RNN) model over other models.
- Determine whether the Exponential Smoothing Recurrent Neural Network (ES-RNN) model emerges as the optimal choice for each defined environment.

By evaluating its performance across the four time series, the study aims to determine whether the ES-RNN model consistently outperforms other models and can be considered the best solution for the unique characteristics of the four series.

Ultimately, the primary objective of this thesis is to develop a holistic understanding of the performance of the ES-RNN model relative to other models when applied to the selected time series. This thesis aims to elucidate the reasons for the superiority of the ES-RNN model and, through careful comparisons and analyses, determine its suitability as the best choice of model for each of the selected time series.

Chapter 4

Theoretical Background

This chapter is divided into two sections. The section 4.1, called Models, is an introduction to the various models that will be used later in our experimental study. This section discusses how these models work, their advantages and disadvantages, and what we expect from them when dealing with time series data.

The section 4.2, entitled Theoretical Foundations, deals with the technical aspects associated with these models. From representing time series data as tabular datasets to testing, this section provides a comprehensive understanding of the process of building, training, validating and testing the models that we will later use in our experimental study.

Note that since the main topic of this thesis is to evaluate the predictive performance of hybrid and deep learning models using statistical models as benchmarks, we will not review the statistical models much in this thesis. We assume that the reader has some basic knowledge of time series models and that we do not need to review these models in detail. There are already many articles and publications on the subject and we invite the reader to have a look at them.

4.1 Models

4.1.1 Seasonal Autoregressive Integrated Moving Average (SARIMA)

Since the Seasonal Autoregressive Integrated Moving Average (SARIMA) model is one of the selected benchmarks for evaluating the machine learning, deep learning and hybrid models, it is worth discussing it a little so that we can remember later how to understand its coefficients.

SARIMA stands for Seasonal Autoregressive Integrated Moving Average and is an extension of the Autoregressive integrated Moving Average (ARIMA) model, a popular time series model for non-seasonal time series data. The SARIMA model extends the ARIMA model by incorporating seasonal components. These seasonal components allow the seasonal patterns in the data to be taken into account in the modelling.

To do this, the SARIMA model is based on seven parameters: p , d , q , P , D , Q , and m . The lower case parameters (p , d , q) represent the non-seasonal components, while the upper case parameters (P , D , Q) represent the seasonal components. The parameter m represents the number of time steps in each seasonal cycle.

Here is a more detailed breakdown of the SARIMA components:

- Autoregressive (AR) component: This component models the relationship between an observation and a series of lagged observations. The parameter p represents the order of the autoregressive component, i.e. the number of lagged observations.
- Integrated (I) component: This component represents the degree of differencing required to make the time series data stationary. The degree of differencing is represented by the parameter d .
- Moving Average (MA) component: This component models the error term as a linear combination of past error terms. The parameter q represents the order of the moving average component, i.e. the number of past error terms considered.
- Seasonal Autoregressive (SAR) component: This component models the relationship between an observation and a series of seasonally lagged observations. The parameter P represents the order of the seasonal autoregressive component, i.e. the number of seasonally lagged observations.
- Seasonal Integrated (SI) component: This component represents the degree of seasonal differencing required to make the time series data stationary. The degree of seasonal differencing is represented by the parameter D .
- Seasonal Moving Average (SMA) component: This component models the error term as a linear combination of past seasonal error terms. The parameter Q represents the order of the seasonal moving average component, i.e. the number of past seasonal error terms.
- Seasonal period (m): This parameter represents the number of time steps in each seasonal cycle, e.g. 12 for monthly data or 4 for quarterly data.

By choosing appropriate values for these parameters, the SARIMA model can capture complex relationships, patterns and trends present in the time series data and then be used to generate forecasts.

4.1.2 Multilayer Perceptron (MLP)

4.1.2.1 Perceptron

To better understand the Multilayer Perceptron (MLP) and how it works, we first need to look at its predecessor, which is also one of its main components, the Perceptron model.

Developed by Rosenblatt in 1957, the Perceptron model was mainly proposed to solve restrictive regression and classification problems (Rosenblatt, 1958). Inspired by nature, and more specifically biology, a perceptron is a mathematical abstraction of a neuron cell (Denuit et al., 2019). A neuron consists of three parts: dendrites, a cell body and an axon. Input signals are received by the dendrites through a biochemical process that weights the information according to its relevance, then the cell body sums up the incoming signal and finally, if a threshold is reached, the cell reacts and sends an output message down the axon (Denuit et al., 2019).

Similarly, in a perceptron, features are weighted before being sent to an activation function ϕ , which in biological terms is the cell body. The numerical output provided by the cell body is then used as input information for further steps (Denuit et al., 2019).

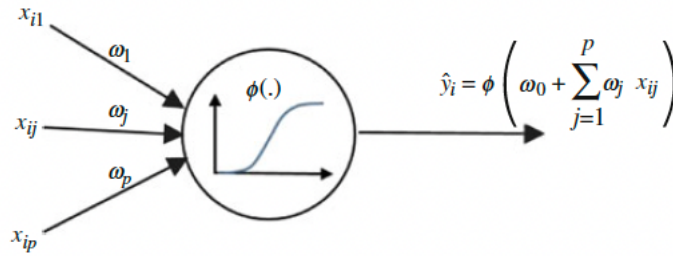


Figure 4.1: Perceptron model with three inputs and an activation function ϕ . From *Effective Statistical Learning Methods for Actuaries III: Neural Networks and Extensions* (p. 3) by Denuit, et al., 2019, Springer Actuarial. Copyright 2019 by Springer.

To give a mathematical example of how the Perceptron works, suppose we want to build an autoregressive model with P lagged observations and the unknown activation function ϕ applied to their sum. In addition, assume that the observations are univariate. The problem we want to solve with the Perceptron model is then given by:

$$\hat{y}_t = \phi(b_0 + \sum_{p=1}^P w_p x_{t-p}) \quad (4.1)$$

where the w 's are the parameters to learn, b_0 is the bias, x 's are the true observations, and $\hat{y}_t$ is the prediction provided by the Perceptron model.

Note that with a suitable optimizer and loss function, the parameters of the Perceptron model can be tuned to minimize the errors between predictions and true observations. Note also that this model is reduced to a linear regression with P lagged observations as inputs when ϕ is the identity function.

4.1.2.2 Conclusion

Although the Perceptron model can be useful for simple tasks, extensions to this model have been developed to deal with increasing data complexity (LeCun et al., 2015; Goodfellow et al., 2016). One such extension is the Multilayer Perceptron model (MLP), which will be discussed in the next section (LeCun et al., 2015).

4.1.2.3 Multilayer Perceptron

Multilayer Perceptron (MLP) models are sets of connected neurons (Goodfellow et al., 2016). There is an infinite number of possible connections. These neurons are organized into layers, which are connected to each other by links, or equivalently, weights.

In a feed-forward MLP model, information flows in one direction, from the input to the output nodes (Goodfellow et al., 2016). There is no circular relationship between the layers, nor between the nodes. The intermediate layers of neurons between the input neurons, also known as the input layer, and the output neurons, also known as the output layer, are called hidden layers (LeCun et al., 2015; Goodfellow et al., 2016).

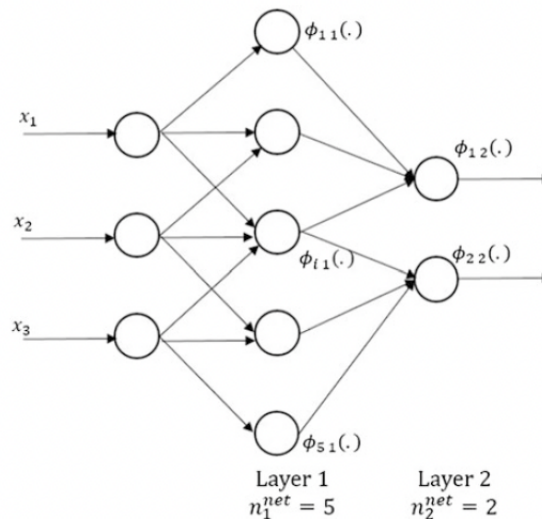


Figure 4.2: Multilayer perceptron (feed-forward neural network) with only one hidden layer of 5 neurons. From *Effective Statistical Learning Methods for Actuaries III: Neural Networks and Extensions* (p. 5) by Denuit, et al., 2019, Springer Actuarial. Copyright 2019 by Springer.

Since there is no feedback during the information flow, a backpropagation algorithm is used in a second step to provide feedback to the neurons about the predictions to adjust the weights and biases (Werbos, 1990; Goodfellow et al., 2016; Denuit et al., 2019). This procedure is widely known in the machine learning and deep learning literature as backpropagation (LeCun et al., 2015; Goodfellow et al., 2016).

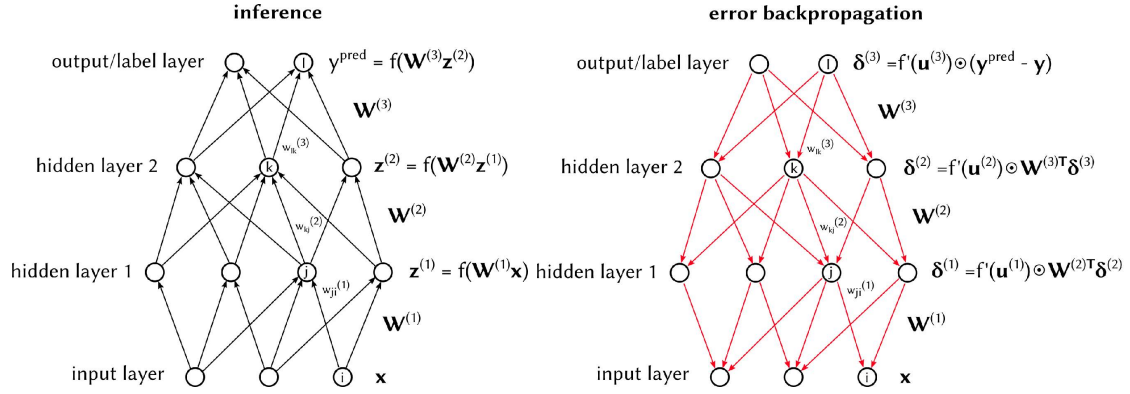


Figure 4.3: Feedforward (left) and backpropagation (right). Kungl, A. F. (2020). Robust learning algorithms for spiking and rate-based neural networks. (Doctoral dissertation). Heidelberg University, Germany.

This two-step process, information feedforward - error feedback, is repeated until convergence is reached.

To see how the MLP can be used in practice to solve time series problems, let us return to the example of an autoregressive model with P lagged observations.

Assume that the architecture consists of an input layer with P inputs, a hidden layer with Q neurons, and an output layer with a single neuron. Also assume that the function ϕ in the hidden layer is unknown, while that of the output layer is the identity function $f(x) = x$. Finally, assume that $\mathbf{x}_t$ is a univariate sequence of P lagged observations. In the example given, a one-step-ahead forecast is provided.

Hidden Layer: The input to the hidden layer can be denoted as $\mathbf{x}_t$, a P -dimensional input vector. The hidden layer consists of Q neurons, and each neuron has its own bias (b) and its own set of weights ($\mathbf{w}$). The activation function ϕ , which is unknown in the given context, is applied element by element to the outputs of the hidden layer. The equation for the hidden layer is given by:

$$\mathbf{h}_t = \phi(\mathbf{W}_h \cdot \mathbf{x}_t + \mathbf{b}_h) \quad (4.2)$$

where

- $\mathbf{h}_t$ is the output of the hidden layer, a Q -dimensional vector.
- $\mathbf{b}_h$ is the bias vector of shape $(Q, 1)$.
- $\mathbf{W}_h$ is the weight matrix of shape (Q, P) for the hidden layer.

Output Layer: The output layer is a single neuron with the identity activation function $f(x) = x$. The output vector of the hidden layer, $\mathbf{h}_t$, serves as the input to the output layer. The output layer also has its own bias (b) and set of weights ($\mathbf{w}$). The equation for the output layer is given by:

$$\hat{y}_t = \phi(\mathbf{W}_o \cdot \mathbf{h}_t + b_o) = \mathbf{W}_o \cdot \mathbf{h}_t + b_o \quad (4.3)$$

where

- $\hat{y}_t$ is the output of the MLP, a scalar value.
- b_o is the bias of the output layer, a scalar value.
- $\mathbf{W}_o$ is the weight matrix of shape (1, Q) for the output layer.

During the model training, the weights and biases of the model are learned. The problem to solve is as follows, assuming that $\boldsymbol{\theta}$ contains all the parameters of the model and $\mathcal{L}$ is a loss-definite function:

$$\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta}} \sum_{t=P+1}^T \mathcal{L}(y_t, \hat{y}_t | \boldsymbol{\theta}) \quad (4.4)$$

Again, with a suitable optimizer and loss function, the parameters of the model can be learned so that the errors between the predictions and the actual observations are minimal.

4.1.2.4 Conclusion

Although Multilayer Perceptron (MLP) models are widely used for traditional regression and classification tasks, they are less commonly used for dealing with sequential data (LeCun et al., 2015; Goodfellow et al., 2016). By treating all inputs as independent, MLPs often fail to capture the temporal structure inherent in time series data (Goodfellow et al., 2016; Makridakis et al., 2020).

4.1.3 Recurrent Neural Network (RNN)

4.1.3.1 Recurrent Neural Network

As mentioned in the subsection 4.1.2.3, Multilayer Perceptron (MLP) models have no feedback connections. The outputs of the model are not fed back into itself. In addition, the input vectors are processed in an isolated and independent manner (Goodfellow et al., 2016). There is no exchange of information between the input vectors, even if they are from the same series (Brownlee, 2017).

To address these limitations, especially in scenarios involving sequential data where observations are time-dependent, alternative classes of neural networks have been introduced (LeCun et al., 2015). Among these classes, Recurrent Neural Networks (RNNs) stand out as a specialized type of artificial neuron tailored for processing sequential data (Goodfellow et al., 2016).

To move from MLP to RNN, one of the early ideas in machine learning and statistical models was to share parameters between parts of the model (Goodfellow et al., 2016). Sharing parameters makes it possible to extend and apply a model to examples of different structures (e.g. different lengths) and to generalize across them (Goodfellow et al., 2016).

In a traditional fully connected feed-forward network, the network would have separate parameters for each input feature, meaning that the network would have to learn all the rules at each position in the series (LeCun et al., 2015). In contrast, a recurrent neural network uses the same weights across multiple time steps. Each time step prediction is a function of the previous time step predictions, resulting in the sharing of parameters over a very deep computational graph (Goodfellow et al., 2016).

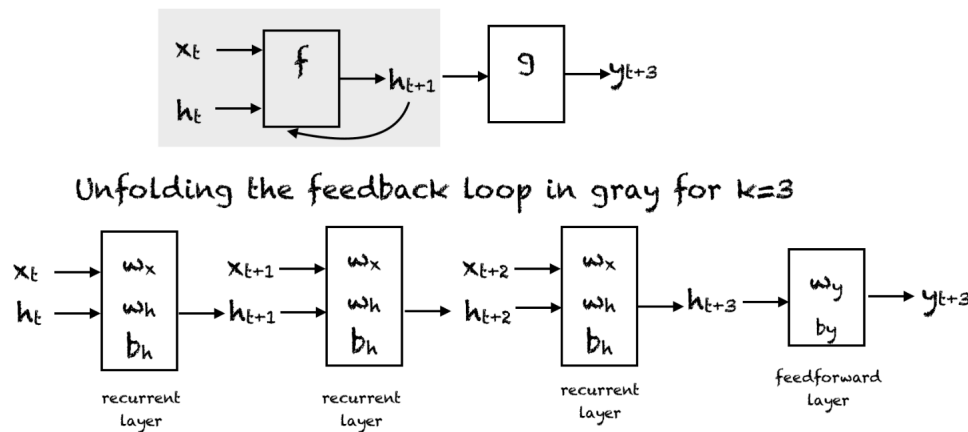


Figure 4.4: The parameter k indicates the number of previous time steps that the RNN uses to generate a prediction. Brownlee, J. (2017). Unrolled recurrent neural network with input. [Figure]. Machine Learning Mastery. Retrieved from <https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/>

Note to the reader: To explain the figure 4.4 and more generally how RNNs work, assume that $k = 3$ previous time steps are used to predict the next time step (e.g.: the sequence $[x_1, x_2, x_3]$ is used to predict x_4). Let's also assume that we are in a univariate context, so the inputs are scalar.

Since RNNs process sequences one step at a time, first x_1 is fed to the model, then x_2 , and finally x_3 to get the prediction for x_4 .

RNNs have feedback loops and process input sequences one time step at a time, maintaining in their hidden units a state vector (or scalar value, depending on the neural network architecture) $\mathbf{h}$ that implicitly contains information about the previously processed time steps in the series (LeCun et al., 2015).

In fact, RNNs are information-looping, meaning that they consider both the current time step and what they have learned from previous ones when making a prediction (Olah, 2015; Goodfellow et al., 2016). This way of working is important because past information may contain relevant information about what is coming next.

Although RNNs are trained using the same principle as MLPs, there is a small difference between the two training processes (Werbos, 1990). Because RNNs share parameters over time, the algorithm used to adjust the weights and biases in these artificial neural networks is called backpropagation through time (BPTT) (Werbos, 1990).

In BPTT, the errors that the model makes in its predictions are propagated backwards from the last time step to the first time step, with all time steps being unrolled. This type of procedure makes it possible to calculate the error made for each time step and thus to update the weights and biases accordingly (Werbos, 1990; Goodfellow et al., 2016)¹.

While BPTT is theoretically suitable for training RNNs, in practice it often suffers from two major problems: the exploding gradient and the vanishing gradient. BPTT can cause the gradient to explode or, conversely, become too small. This makes learning unstable and long-term dependencies difficult to learn (LeCun et al., 2015; Goodfellow et al., 2016).

4.1.3.2 Conclusion

Although traditional RNNs are able to learn short-term dependencies efficiently, they suffer when it comes to learning long-term dependencies and providing stable learning (LeCun et al., 2015). To overcome these problems, a variant of recurrent neural networks called Long Short-Term Memory (LSTM) was developed in the late 1990s (Hochreiter & Schmidhuber, 1997).

¹Backpropagation through time (BPTT) is not explained in this report. There is already a large body of literature on the subject, and we invite the reader to consult it if he or she requires an advanced understanding of it.

4.1.4 Long Short-Term Memory (LSTM)

4.1.4.1 Long Short-Term Memory

Developed at the end of the 1990s by Sepp Hochreiter and Jürgen Schmidhuber (Hochreiter & Schmidhuber, 1997), Long Short-Term Memory - also known as LSTM - is a special type of recurrent neural network capable of learning long-term dependencies and avoiding unstable gradients (Hochreiter & Schmidhuber, 1997; LeCun et al., 2015). Unlike a traditional Recurrent Neural Network (RNN), a Long Short-Term Memory (LSTM) model does not use classical layers to propagate information from input to output. Instead, it uses special types of layers, called LSTM layers, which contain LSTM cells, also called hidden units (Hochreiter & Schmidhuber, 1997; Goodfellow et al., 2016).

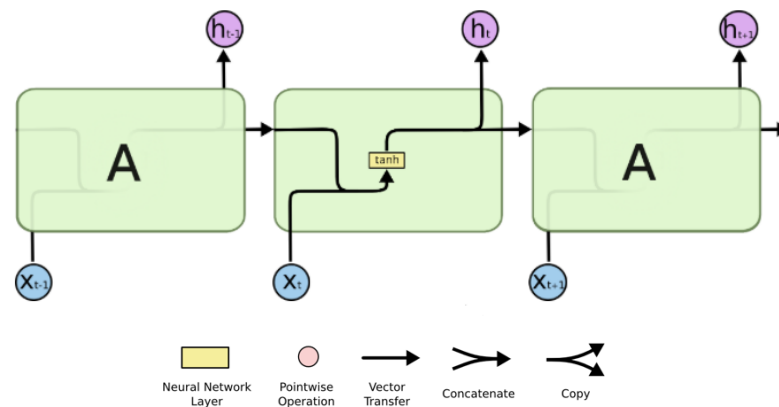


Figure 4.5: Simple RNN architecture. The repeating structure contains only one layer (tanh). Olah, C. (2015). Recurrent layer. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

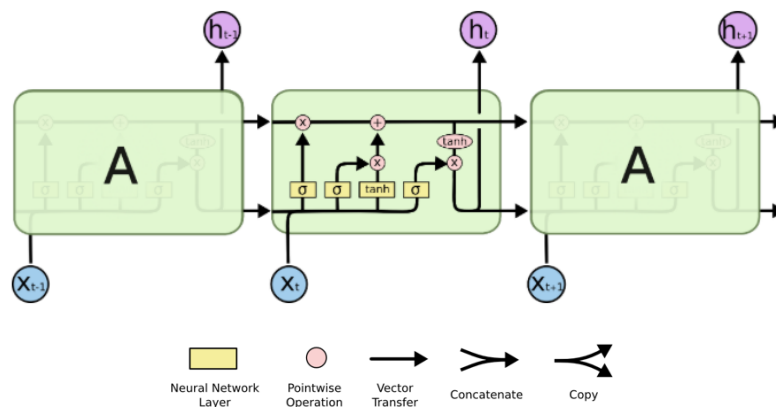


Figure 4.6: Simple LSTM architecture. The repeating structure contains four layers (three sigmoid layers and one tanh layer), each of which interacts with the others in a special way. Olah, C. (2015). LSTM cell. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Conceptually, the LSTM layer consists of three basic elements: a cell state vector $\mathbf{C}$, a hidden state vector $\mathbf{h}$, which has already been introduced in the section 4.1.3, and a set of gates that regulate how information flows through the network (Hochreiter & Schmidhuber, 1997; Goodfellow et al., 2016).

The main innovation of LSTMs compared to traditional RNNs is the cell state vector $\mathbf{C}$ (Hochreiter & Schmidhuber, 1997; Olah, 2015; Goodfellow et al., 2016). Long-term information is stored in this vector (Hochreiter & Schmidhuber, 1997). LSTMs make adjustments to the information contained in this vector as they learn to correctly incorporate long-term dependencies into their predictions (Hochreiter & Schmidhuber, 1997; Olah, 2015; Goodfellow et al., 2016). Gates are used to making these adjustments. Gates are the means by which LSTMs regulate how information flows through the network (Olah, 2015; Goodfellow et al., 2016).

To get a better understanding of how LSTMs deal with long-term dependencies in practice, let's take a look at what happens inside an LSTM cell, and therefore an LSTM layer.

4.1.4.2 Process

First, let us look at the variables that will be used later to explain the Long Short-Term Memory (LSTM):

- $x(t)$: Input vector at time step t . It typically has a dimension of $p \times 1$, where p is the number of lagged observations.
- $h(t)$: Hidden state vector at time step t . It also has a dimension of $m \times 1$, where m is the number of hidden units. This corresponds to the number of LSTM cells in the LSTM layer.
- $C(t)$: Cell state vector at time step t . It has the same dimension as the hidden state vector, $m \times 1$.
- W_i, W_f, W_o, W_c : Weight matrices for the input, forget, output, and cell state, respectively. These matrices have the dimensions $m \times (m + p)$.
- b_i, b_f, b_o, b_c : Bias vectors for the input, forget, output, and cell state, respectively. They have dimensions $m \times 1$.
- $i(t), f(t), o(t)$: Gate vectors at time step t . They have the same dimension as the hidden state and cell state vectors, $m \times 1$.
- $[h(t - 1), x(t)]$: Concatenation of the hidden state and input vectors. It has the dimensions $(m + p) \times 1$.

Finally, note that m represents the number of hidden units, or equivalently, the number of memory cells in the LSTM layer (Olah, 2015). A higher m captures better long-term dependencies. However, if m is high, the model is more complex

and therefore more computationally expensive to train (Olah, 2015).

Note to the reader: Note that in the following explanations, we assume an LSTM layer with m hidden units. Each hidden unit will then correspond to an LSTM cell.

As shown in the figure 4.7, the first step in an LSTM layer is to decide which information from the previous cell state vector $\mathbf{C}_{t-1}$ should be kept and which should be discarded. This decision is made using a sigmoid layer called the **Forget Gate layer** (Goodfellow et al., 2016).

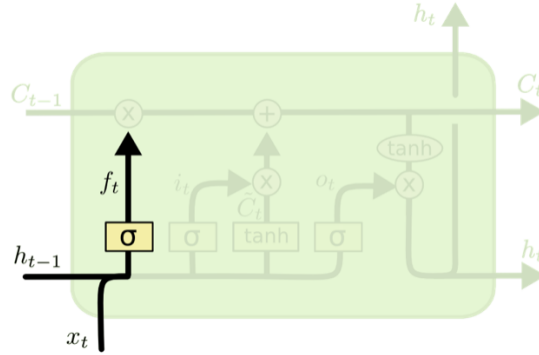


Figure 4.7: Forget gate layer. Olah, C. (2015). The forget gate layer. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

The **Forget Gate layer** examines the inputs x_t and $\mathbf{h}_{t-1}$ and generates a vector of values between 0 and 1 for each component of $\mathbf{C}_{t-1}$. A value of 1 for a given component means that all the information related to that component should be kept, whereas a value of 0 indicates that all the information related to that component should be discarded (Olah, 2015):

$$f(t) = \sigma(W_f \cdot [h(t-1), x(t)] + b_f) \quad (4.5)$$

The result of this operation is a vector $\mathbf{f}_t$, which is then point-wise multiplied with $\mathbf{C}_{t-1}$ to produce a vector that quantifies how much of each component of the vector $\mathbf{C}_{t-1}$ should be retained in $\mathbf{C}_t$ (Hochreiter & Schmidhuber, 1997; Olah, 2015; Goodfellow et al., 2016).

At the same time, another process determines what new information should be stored in $\mathbf{C}_t$. This is achieved by two simultaneous operations, as shown in the figure 4.8:

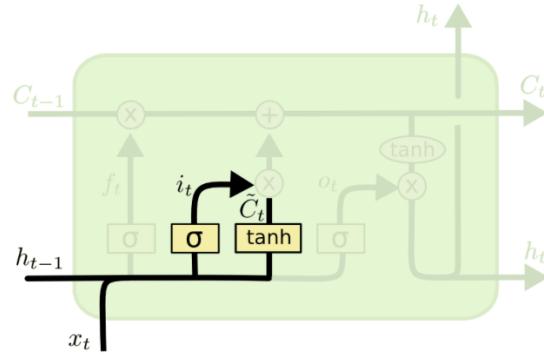


Figure 4.8: Input gate layer. Olah, C. (2015). The input gate layer. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

First, a sigmoid layer, commonly referred to as the **Input Gate layer** in the literature, determines which values of the cell state vector $\mathbf{C}_t$ should be updated (Olah, 2015):

$$i(t) = \sigma(W_i \cdot [h(t-1), x(t)] + b_i) \quad (4.6)$$

Second, a tanh layer generates a fresh cell state vector denoted $\tilde{\mathbf{C}}_t$ (Olah, 2015):

$$\tilde{C}(t) = \tanh(W_c \cdot [h(t-1), x(t)] + b_c) \quad (4.7)$$

The two vectors $\mathbf{i}_t$ and $\tilde{\mathbf{C}}_t$ are then multiplied together, and the result is added point by point to the vector $\mathbf{f}_t \cdot \mathbf{C}_{t-1}$ to generate the cell state $\mathbf{C}_t$ (Olah, 2015; Goodfellow et al., 2016):

$$C(t) = f(t)C(t-1) + i(t)\tilde{C}(t) \quad (4.8)$$

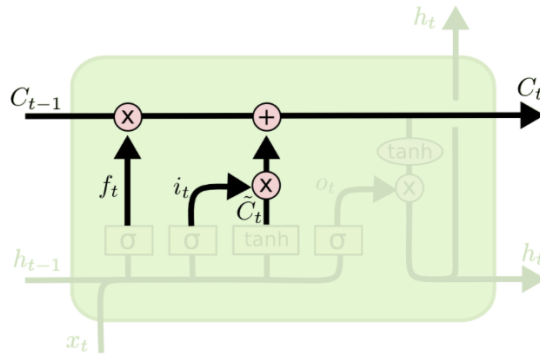


Figure 4.9: The outputs of the forget and input layers are used to obtain $\mathbf{C}_t$. Olah, C. (2015). LSTM cell. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

To get the vector $\mathbf{o}_t$ another sigmoid layer is used, the **Output Gate layer** (Olah, 2015; Goodfellow et al., 2016). This layer, just like the **Forget Gate layer**, examines the inputs x_t and h_{t-1} and outputs a vector containing values between 0 and 1 (Olah, 2015):

$$o(t) = \sigma(W_o \cdot [h(t-1), x(t)] + b_o) \quad (4.9)$$

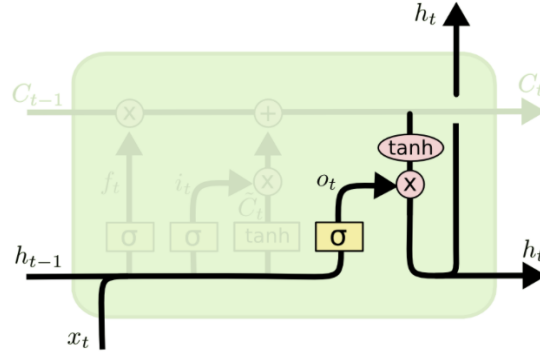


Figure 4.10: Output gate layer. Olah, C. (2015). The output gate layer. [Figure]. colah's blog. Retrieved from <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Finally, to obtain the hidden state vector $\mathbf{h}_t$, the vector $\mathbf{o}_t$ is multiplied point by point with $\mathbf{C}_t$, which is first passed through a tanh layer:

$$h(t) = o(t) \cdot \tanh(C(t)) \quad (4.10)$$

The hidden state vector $\mathbf{h}_t$ is then passed to the next time step to be processed. The process is repeated as many times as there are time steps in the series to be processed.

It is important to note that the LSTM cells within an LSTM layer operate independently. Only the outputs of these cells, which are hidden states, are connected by weights, as in MLPs with neurons (Olah, 2015; Goodfellow et al., 2016).

4.1.4.3 Remarks

In the previous section we have mainly discussed the architecture of traditional LSTM layers and hence traditional LSTM cells. Note, however, that there are other LSTM cell architectures available (Olah, 2015).

Depending on the requirements, other LSTM cell architectures may be preferred, although the differences with the classical one are minor (Olah, 2015). Among these other architectures we can mention the Bidirectional LSTM cell (Schuster & Paliwal, 1997), the Gated Recurrent Unit cell (Cho et al., 2014) or the Depth Gated Recurrent cell (Yao et al., 2015).

4.1.4.4 Conclusion

The key difference between LSTMs and traditional RNNs is their ability to process and remember information over long periods of time (Olah, 2015; Goodfellow et al., 2016). While in theory RNNs can handle long sequence data, in practice they struggle due to a number of technical limitations.

LSTMs, on the other hand, are specifically designed to overcome these limitations (Hochreiter & Schmidhuber, 1997; Olah, 2015). By incorporating gate mechanisms, LSTMs can control the flow of information and therefore effectively remember or forget information over long temporal horizons (Hochreiter & Schmidhuber, 1997; Olah, 2015). It is this crucial feature that makes LSTMs more suitable than RNNs for complex tasks involving long-term dependencies (Hochreiter & Schmidhuber, 1997; LeCun et al., 2015).

Unfortunately, the advantages of LSTMs come at a price. Training LSTMs is more complex and computationally expensive than training traditional RNNs (Hochreiter & Schmidhuber, 1997; Olah, 2015). They also require larger datasets to generalize effectively, and can be prone to overfitting if not trained appropriately (Pascanu et al., 2013).

Thus, although LSTMs offer a solution to the limitations of traditional RNNs, the choice between the two should be driven by the specific requirements of the task at hand. For tasks involving short sequences or limited data, RNNs may be sufficient (LeCun et al., 2015; Goodfellow et al., 2016). However, for tasks that require the processing of long sequence data and the learning of long-term dependencies, LSTMs are often the better choice (Hochreiter & Schmidhuber, 1997; LeCun et al., 2015; Olah, 2015).

4.1.5 Exponential Smoothing Recurrent Neural Network (ES-RNN)

4.1.5.1 Exponential Smoothing Recurrent Neural Network

The Exponential Smoothing Recurrent Neural Network (ES-RNN) was presented by Slawek Smyl in 2018 during the M4 competition. It was the winning model of the competition (Makridakis et al., 2020; Smyl, 2020). The model was found to be 9.4% more accurate than the benchmark model used during the competition as can be seen in the table 2.1.

The ES-RNN model is a hybrid model that blends statistical and machine learning concepts. By mixing the two paradigms, it leverages the strengths of both to provide more robust and accurate predictions than pure statistical or machine learning models (Smyl, 2020).

The model is designed to work in two steps (Smyl, 2020). First, a Holt-Winters (HW) model is used, and then a Long-Short Term Memory (LSTM) network (Smyl, 2020). The HW model is used as a pre-processing layer, while the LSTM updates the coefficients of the HW model per series and learns the global patterns of the series (Redd et al., 2019; Smyl, 2020).

4.1.5.2 Holt-Winters (HW) - Summary

Since the Exponential Smoothing Recurrent Neural Network (ES-RNN) model incorporates slightly adapted Holt-Winters (HW) models into its architectures, let's first review the classic HW model and its variants to fully understand how these models are integrated and used in the ES-RNN architecture model.

The Holt-Winters (HW) model is a forecasting model that incorporates both the trend and the seasonality of the data in its modelling. The model can be structured in different ways, depending on whether the seasonality in the data is additive or multiplicative (R. Hyndman & Athanasopoulos, 2018).

However, all Holt-Winters models consist of three basic elements:

- Level (l_t): The estimated level at time step t .
- Trend (b_t): The estimated rate of increase or decrease at time step t .
- Seasonality (s_t): The estimated seasonal component at time step t .

Holt-Winters Additive

The additive model assumes that seasonality is constant over the series (R. Hyndman & Athanasopoulos, 2018).

The following equations summarise the Holt-Winters additive model:

$$l_t = \alpha \times (y_t - s_{t-L}) + (1 - \alpha) \times (l_{t-1} + b_{t-1}) \quad (4.11)$$

$$b_t = \beta \times (l_t - l_{t-1}) + (1 - \beta) \times b_{t-1} \quad (4.12)$$

$$s_t = \gamma \times (y_t - l_t) + (1 - \gamma) \times s_{t-L} \quad (4.13)$$

where:

- y_t is the actual value at time step t .
- L is the length of the seasonal pattern. For example, $L = 12$ if the seasonal pattern is annual.
- α , β , and γ are the smoothing coefficients that control the weighting of level, trend, and seasonality. These three parameters have values in the interval $[0, 1]$.

The model is used iteratively to produce forecasts. The level, trend, and seasonality estimates are updated on the basis of observations and forecasts (R. Hyndman & Athanasopoulos, 2018):

$$\hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-L(k+1)} \quad (4.14)$$

with k being the integer part of $\frac{(h-1)}{L}$.

Holt-Winters Multiplicative

The multiplicative model assumes that the seasonal component of the time series is proportional to the level:

$$l_t = \alpha \times (y_t/s_{t-L}) + (1 - \alpha) \times (l_{t-1} + b_{t-1}) \quad (4.15)$$

$$b_t = \beta \times (l_t - l_{t-1}) + (1 - \beta) \times b_{t-1} \quad (4.16)$$

$$s_t = \gamma \times (y_t/l_t) + (1 - \gamma) \times s_{t-L} \quad (4.17)$$

The main differences with the additive model are the fractional terms in the level and seasonality equations.

The following equation is used to make predictions:

$$\hat{y}_{t+h|t} = (l_t + hb_t)s_{t+h-L(k+1)} \quad (4.18)$$

Both the additive and multiplicative models can be used for forecasting. The choice between the two depends on the nature of the seasonality in the data. If the amplitude of the seasonality remains constant as the level of the series changes, the additive model is appropriate. Conversely, if the amplitude of the seasonality is proportional to the level of the series, the multiplicative model is more appropriate (R. Hyndman & Athanasopoulos, 2018).

4.1.5.3 Preprocessing Layer - Exponential Triple Smoothing (ETS)

To pre-process the series, the ES-RNN uses Holt-Winters (HW) models to estimate the local parameters of each series (Redd et al., 2019; Smyl, 2020). For each series, the smoothing coefficients are estimated and then used to normalize and deseasonalize the series and observations (Smyl, 2020).

To achieve this, each series is first fitted with a specific HW model that fits its specific characteristics (Smyl, 2020):

Non-seasonal model

$$l_t = \alpha y_t + (1 - \alpha)l_{t-1} \quad (4.19)$$

Single seasonality model

$$l_t = \alpha\left(\frac{y_t}{s_t}\right) + (1 - \alpha)l_{t-1} \quad (4.20)$$

$$s_{t+L} = \beta\left(\frac{y_t}{l_t}\right) + (1 - \beta)s_t \quad (4.21)$$

Double seasonality model

$$l_t = \alpha\left(\frac{y_t}{s_t u_t}\right) + (1 - \alpha)l_{t-1} \quad (4.22)$$

$$s_{t+L} = \beta\left(\frac{y_t}{l_t u_t}\right) + (1 - \beta)s_t \quad (4.23)$$

$$u_{t+K} = \gamma\left(\frac{y_t}{l_t s_t}\right) + (1 - \gamma)u_t \quad (4.24)$$

where l is a level state variable, s is a seasonal state variable, and u is a second seasonal state variable. K is the length of the second seasonal pattern.

Once the per-series smoothing coefficients of each series have been found, the series are normalized and deseasonalized (Redd et al., 2019):

Non-seasonal model

$$x_t = \frac{y_t}{l_t} \quad (4.25)$$

Single seasonality model

$$x_t = \frac{y_t}{l_t s_t} \quad (4.26)$$

Double seasonality model

$$x_t = \frac{y_t}{l_t s_t u_t} \quad (4.27)$$

This process is performed for each time series of the dataset (Redd et al., 2019; Smyl, 2020).

By using the HW model as a pre-processing layer, the ES-RNN model can be adapted to any series. This makes it highly adaptable to a wide range of problems and domains (Redd et al., 2019; Smyl, 2020).

4.1.5.4 Deep Learning Layer - Dilated Long Short-Term Memory (DLSTM)

Once the series have been normalized and deseasonalized, they are fed series by series into a Long Short-Term Memory (LSTM) network (Redd et al., 2019; Smyl, 2020). A single neural network is trained on all the series.

Depending on the periodicity of the series to be treated: hourly, daily, monthly, etc., different LSTM network architectures can be used (Smyl, 2020). However, the basic LSTM neural network used in the ES-RNN model is a Dilated LSTM (DLSTM) neural network (Redd et al., 2019). A DLSTM neural network is an LSTM neural network with some of its connections skipped (Chang et al., 2017; Redd et al., 2019; Smyl, 2020).

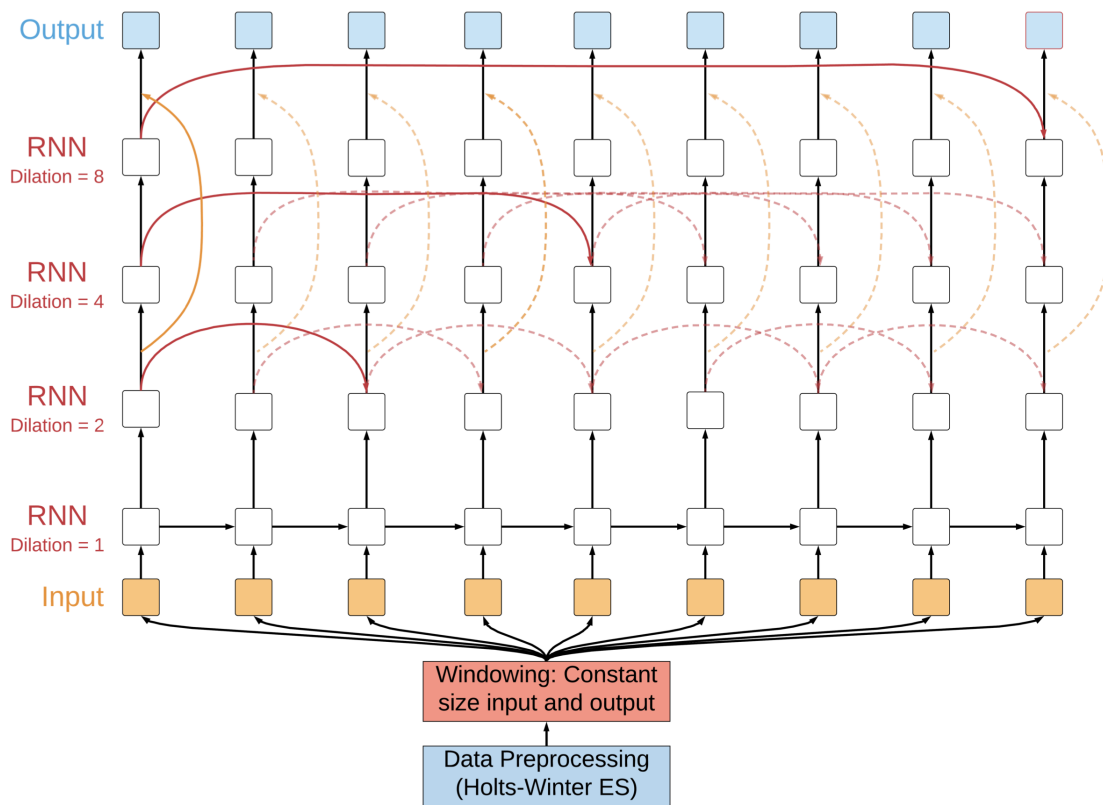


Figure 4.11: NN Architecture (quarterly seasonality) by Smyl et al. (2018).
Reprinted from (Redd et al., 2019).

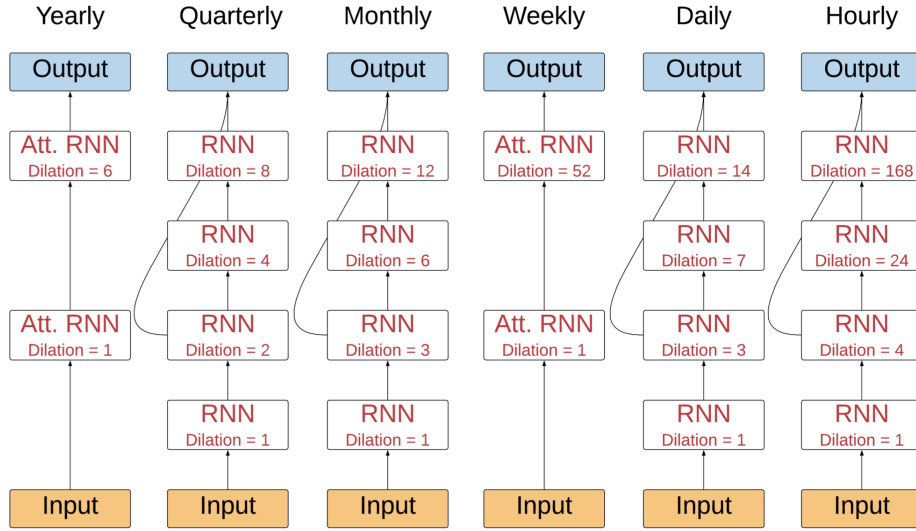


Figure 4.12: NN Architectures by Smyl et al. (2018).
Reprinted from (Redd et al., 2019).

In contrast to a fully connected LSTM neural network, a DLSTM neural network significantly increases computational efficiency and allows the neural network to better remember information from previous input instances (Redd et al., 2019; Smyl, 2020). Omitting some connections in the network allows information to be transferred more quickly to the other LSTM cells and to the output layer while avoiding noise and loss of information (Redd et al., 2019; Smyl, 2020).

4.1.5.5 Loss Function - Quantile Loss

In the M4 competition, Smyl used a specific loss function, known as quantile loss, to learn the parameters of his Exponential Smoothing Recurrent Neural Network (ES-RNN) models (Smyl, 2020):

$$\mathcal{L}(y_t, \hat{y}_t) = \begin{cases} (y_t - \hat{y}_t) \times Q, & \text{if } \hat{y}_t \leq y_t \\ (\hat{y}_t - y_t) \times (1 - Q), & \text{if } \hat{y}_t > y_t \end{cases} \quad (4.28)$$

where Q is a quantile value.

To explain this decision, Slawek Smyl noticed that when the average of sMAPE and MASE (see points 4.2.4.6 and 4.2.4.7) was used as the loss function, which was the criterion used to evaluate the models during the M4 competition, the ES-RNN models tended to have a positive bias (Smyl, 2020). In addition, the loss function was not differentiable at some points, which was problematic for gradient-based optimisation algorithms.

To overcome these problems, Smyl decided to use a quantile loss function with a non-symmetric penalty. He specified that he used a value of Q equal to 0.48 to counteract these problems (Smyl, 2020).

Quantile Loss - Example

To better understand this choice and the nuances of the quantile loss function, let's take a look at an illustration.

The Q parameter is an important factor in the quantile loss function. It determines the penalty strategy of the model. The setting of Q depends on the model's propensity for positive, negative or neutral bias in its predictions. In essence, Q controls the amount of penalty for underestimation (negative bias) and overestimation (positive bias).

Consider a scenario where the model tends to overestimate (positive bias). To counteract this, a Q value of less than 0.5 is chosen. This deliberate choice increases the penalty for overestimation (positive bias), thereby reducing the positive bias. For the sake of illustration, let us assume that Q is equal to 0.40; under this premise, the following formula will take the lead as the intended loss function for the training:

$$\mathcal{L}(y_t, \hat{y}_t) = \begin{cases} (y_t - \hat{y}_t) \times 0.40, & \text{if } \hat{y}_t \leq y_t \\ (\hat{y}_t - y_t) \times 0.60, & \text{if } \hat{y}_t > y_t \end{cases} \quad (4.29)$$

If we choose $Q = 0.40$, we notice a remarkable property: the formula (4.29) imposes a higher penalty for overestimation (positive bias) than for underestimation (negative bias).

On the other hand, if the model tends to underestimate (negative bias), a value of Q greater than 0.5 is preferred to reduce the negative bias. As an illustrative example, let's assume $Q = 0.60$:

$$\mathcal{L}(y_t, \hat{y}_t) = \begin{cases} (y_t - \hat{y}_t) \times 0.60, & \text{if } \hat{y}_t \leq y_t \\ (\hat{y}_t - y_t) \times 0.40, & \text{if } \hat{y}_t > y_t \end{cases} \quad (4.30)$$

Finally, if no bias is found in the model, the value $Q = 0.5$ is used as the designated choice:

$$\mathcal{L}(y_t, \hat{y}_t) = \begin{cases} (y_t - \hat{y}_t) \times 0.50, & \text{if } \hat{y}_t \leq y_t \\ (\hat{y}_t - y_t) \times 0.50, & \text{if } \hat{y}_t > y_t \end{cases} \quad (4.31)$$

In such a scenario, both overestimation (positive bias) and underestimation (negative bias) will incur identical penalties. It should also be noted that in this scenario the approach is equivalent to calculating the Mean Absolute Error (MAE) loss function (see point 4.2.4.2).

All exponential smoothing and neural network model parameters are fitted simultaneously with the Gradient Descent (GD) algorithm (Smyl, 2020). This approach of fitting all parameters with a single loss function is the reason why the ES-RNN model was classified as a hybrid model rather than a combination or ensemble of methods (Smyl, 2020).

4.1.5.6 Forecasts

Since the neural network model has been trained on normalized time series data, the final operation to be performed to obtain the forecasts is to denormalize the neural network outputs. This is achieved by using the exponential smoothing parameters estimated for each series and later adjusted in the second step (Redd et al., 2019; Smyl, 2020):

Non-seasonal model

$$\hat{y}_{t+1\dots t+h} = \text{RNN}(x_t)l_t \quad (4.32)$$

Single seasonality model

$$\hat{y}_{t+1\dots t+h} = \text{RNN}(x_t)l_t s_{t+1\dots t+h} \quad (4.33)$$

Dual seasonality models

$$\hat{y}_{t+1\dots t+h} = \text{RNN}(x_t)l_t s_{t+1\dots t+h} u_{t+1\dots t+h} \quad (4.34)$$

Depending on the value of h , the time horizon, the model will produce one or more forecasts. Note also that the multiplication operations are point-wise. Finally, remember that the parameters l , s , and u are local and specific to each series, whereas the LSTM model is global.

4.1.5.7 Remarks

It should be emphasized that the section 4.1.5.1 describes the basic architecture of the Exponential Smoothing Recurrent Neural Network (ES-RNN) model.

However, it is important to note that the ESRNN package, which we will introduce later in this thesis, offers a degree of adaptability in the selection of the cells used within the Long Short-Term Memory (LSTM) network. The alternatives include the traditional Recurrent Neural Network (RNN), Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) cells. The GRU cells are simplified versions of the LSTM cells, mainly characterized by a reduced number of gates.

4.1.5.8 Conclusion

With his hybrid model, Slawek Smyl has opened a new door of interest for machine learning and deep learning models in forecasting. During the competition, Slawek's model outperformed all other competing models.

By mixing two models from different fields, statistics and deep learning, Slawek developed a model that can use the properties of both to make better predictions than usual (Makridakis et al., 2020; Smyl, 2020).

Since the HW model has interesting statistical capabilities and properties, the ES-RNN model also benefits from these capabilities by using it in the first step (Redd et al., 2019; Smyl, 2020).

Simple, fast, and easy to interpret: the ES-RNN model benefits from better and faster learning of local patterns. Furthermore, since the HW model is linear and can capture seasonal components, using it as a pre-processing layer allows the ES-RNN model to capture both local linear components and seasonal patterns more accurately for all series. By first pre-processing the time series data with the Holt-Winters model, the neural network part of the model only needs to focus on the remaining components that were not captured (Smyl, 2020).

With the linear and seasonal local patterns removed from the series, the second stage of the ES-RNN model, a deep learning layer, only has to deal with the remaining components of the series (Redd et al., 2019; Smyl, 2020). The deep learning layer, consisting of an LSTM neural network, learns global patterns from the series and combines both local and global patterns to provide reliable predictions (Smyl, 2020).

While previous M competitions have argued that pure single models cannot provide superior forecasts (Makridakis et al., 2000), Slawek Smyl proposed to mix these models into a hybrid model (Smyl, 2020). This avant-garde approach combines the advantages of both models while addressing their limitations, resulting in a forecasting framework that is more robust (Makridakis & Petropoulos, 2020).

By winning the M4 competition with his hybrid model, Smyl has demonstrated to all the promise of hybrid models for forecasting and paved the way for continued research and innovation in the field (Makridakis et al., 2018).

4.2 Theoretical Foundations

As the main focus of this thesis is on univariate time series, this section covers all the concepts necessary for a full understanding of the training, validation, and testing of the models introduced previously in a univariate context. It should be noted, however, that these concepts can also be generalized to the multivariate case.

4.2.1 Time Series Problem - Supervised Learning Problem

In the section 4.1, we mentioned that machine learning and deep learning algorithms are primarily based on tabular data. To effectively use time series data with machine learning and deep learning algorithms, it is crucial to model time series as tabular

data so that they can be used with supervised learning algorithms.

The problem of transforming a time series into a tabular dataset that can be solved by supervised learning algorithms has received considerable attention in the literature. It has been addressed by leading researchers in the field of time series forecasting. This interest has led to the emergence of a widely accepted methodology for addressing this challenge (Brownlee, 2017).

Consider a time series of length T , represented by the vector

$$\mathbf{X} = (X_1, X_2, X_3, \dots, X_T) \in \mathbb{R}^T$$

To transform this series into a tabular dataset, a two-step approach is commonly used in the literature (Brownlee, 2017). First, a fixed number of lagged observations is selected, denoted by the parameter P . Second, rolling windows of length $P+1$ are generated from the series. This two-step process produces a collection of rows that can be used to treat the series as a tabular dataset and thus as a supervised learning problem.

For mathematical convenience, let us assume that $P = 3$. Applying the above methodology to a univariate time series of length T will produce the following tabular dataset:

$$\mathbf{X} = \begin{pmatrix} X_1 & X_2 & X_3 & X_4 \\ X_2 & X_3 & X_4 & X_5 \\ X_3 & X_4 & X_5 & X_6 \\ \dots & \dots & \dots & \dots \\ X_{T-4} & X_{T-3} & X_{T-2} & X_{T-1} \\ X_{T-3} & X_{T-2} & X_{T-1} & X_T \end{pmatrix} \in \mathbb{R}^{T-P \times P+1}$$

Note that the resulting tabular dataset is affected by the parameter P . More specifically, depending on the value of P , a certain number of rows will be lost during the transformation. For example, if $P = 3$, the first three observations of the time series, X_1 , X_2 and X_3 , cannot be used for the prediction evaluation. This is because there are no three previous observations for any of them from which they could be estimated.

While the methodology discussed above has proven to be a useful approach to transforming time series into tabular datasets, it is important to carefully consider the choice of the value of P , as it will affect the structure of the resulting dataset, but also the results of the forecasting models (Brownlee, 2017).

4.2.2 Activation Functions

With the ability to model time series problems as supervised learning problems, the use of neural networks and machine learning algorithms to forecast time series has become increasingly common over time.

In the context of NNs, one of the most important components to be considered in the modelling process is the activation function. In mathematical terms, an activation function is a mathematical function that is applied to the output of a neuron in order to introduce non-linearity into the model. It allows the model to learn complex patterns in the data.

There are many activation functions that can be used with neural network models to make forecasts. Deciding which one to use can be task-specific and influenced by the characteristics of the data being analyzed (Goodfellow et al., 2016; Chollet, 2017).

Let us have a look at four of them that we are going to use in our experiments later on. For time series data and regression problems, these four are the most commonly used in practice.

4.2.2.1 Linear Function

The linear function, also called the identity function, is the simplest activation function of all. The function's output equals the input.

Mathematically speaking, the linear function is defined as

$$f: R \rightarrow R$$

$$f(\mathbf{x} \cdot \mathbf{w} + b) = \mathbf{x} \cdot \mathbf{w} + b \quad (4.35)$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ the vector of weights and b the bias.

Its derivative with respect to the variable $z = \mathbf{x} \cdot \mathbf{w} + b$ is given by

$$\frac{d}{dz}f(z) = 1 \quad (4.36)$$

In regression problems, the linear activation function is typically not used in hidden layers of a neural network since it does not provide non-linearity, which is essential for learning complex patterns in the data (Chollet, 2017). Nevertheless, this function is widely used in the output layer to provide predictions in regression tasks.

4.2.2.2 Sigmoid Function

The sigmoid function, also known as the logistic function, is a smooth, S-shaped function that maps any input value to the range $[0, 1]$. Bounded, continuous and monotonically increasing, this function keeps a neuron's output within a certain range, preventing it from becoming too large or too small during training.

Mathematically speaking, the sigmoid function is defined as

$$\sigma: R \rightarrow [0, 1]$$

$$\sigma(\mathbf{x} \cdot \mathbf{w} + b) = \frac{1}{1 + e^{-(\mathbf{x} \cdot \mathbf{w} + b)}} \quad (4.37)$$

and its derivative with respect to the variable $z = \mathbf{x} \cdot \mathbf{w} + b$ is given by

$$\frac{d}{dz}\sigma(z) = f(z)(1 - f(z)) \quad (4.38)$$

The sigmoid function is not commonly used in hidden layers because it suffers from several drawbacks, the most important being the vanishing gradient problem (Glorot & Bengio, 2010; Pascanu et al., 2012). In fact, the ReLU and hyperbolic tangent functions are generally preferred as activation functions for hidden layers because of their mathematical properties and simplicity (Glorot & Bengio, 2010).

4.2.2.3 Hyperbolic Tangent Function

Similar in many ways to the logistic function (continuous, monotonically increasing and bounded), the hyperbolic tangent function is another activation function widely used in deep learning architectures (Glorot & Bengio, 2010). Unlike the logistic function, the hyperbolic tangent function maps any input value to the range $[-1, 1]$.

Mathematically speaking, the hyperbolic tangent function is defined as

$$\begin{aligned} \tanh: \mathbb{R} &\rightarrow [-1, 1] \\ \tanh(z) &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \end{aligned} \quad (4.39)$$

and its derivative with respect to the variable $z = \mathbf{x} \cdot \mathbf{w} + b$ is given by

$$\frac{d}{dz}\tanh(z) = 1 - (\tanh(z))^2 \quad (4.40)$$

Its main advantage over the logistic function is that the output values are centered around 0 as $\tanh(z) = -\tanh(-z)$, which makes it preferable to the logistic function for hidden layers.

4.2.2.4 Rectifier Linear Unit (ReLU) Function

The ReLU function is a piece-wise continuous linear activation function widely used in deep neural network architectures. In simple terms, this function returns the value of the input if it is positive and 0 otherwise. Due to its simplicity, low computational requirements and mathematical properties, the ReLU function has been considered the standard choice of activation function for hidden layers of neural networks (Géron, 2017).

Mathematically speaking, the ReLU function is defined as

$$f: \mathbb{R} \rightarrow \mathbb{R}_+$$

$$f(z) = \max(0, z) \quad (4.41)$$

and its derivative with respect to the variable $z = \mathbf{x} \cdot \mathbf{w} + b$ is given by

$$\frac{d}{dz}f(z) = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases} \quad (4.42)$$

The ReLU function has long been one of the main activation functions used in deep learning architectures, as it can add non-linear components to models, reduce the risk of vanishing gradients, and is computationally efficient.

Unfortunately, the ReLU function is not perfect. The function suffers from a problem known as dying ReLU's (Géron, 2017). During training, due to the nature of the function, some neurons effectively "die". That is, they stop outputting anything other than 0. To alleviate this problem, variants of ReLU have been developed, such as the leaky ReLU, the randomized leaky ReLU (RReLU), the parametric leaky ReLU (PReLU), and the exponential linear unit (ELU) (Géron, 2017). One of these may be preferred to the original ReLU function, depending on the problem at hand.

Finally, note that although ReLU functions are commonly used for hidden layers, they are rarely used for output layers. This is because their output domain is positive and this may not be appropriate for some problems.

4.2.3 Optimization Algorithms

Choosing an optimizer is another critical component in implementing a neural network. The optimizer trains the parameters of neural network models to minimise the differences between predicted and actual values. There are several optimizers that are commonly used to train neural networks. These include Adam, Gradient Descent (GD) and its variants: Stochastic Gradient Descent (SGD), Mini-Batch Gradient Descent (MGD) (Géron, 2017).

The choice of the optimizer can depend on several factors, such as the size of the dataset, the characteristics of the data being analyzed, and the complexity of the model. For example, Adam may be more suitable for larger datasets with more complex models, while SGD may be more suitable for smaller datasets with simpler models (Glorot & Bengio, 2010; Kingma & Ba, 2017).

This section discusses two optimizers and their variants. These will be used in our experiments later on.

4.2.3.1 Gradient Descent

The Gradient Descent (GD) algorithm is a first-order optimization algorithm. It minimises a cost function $\mathcal{L}$ by iteratively updating model parameters as it learns. Each update uses the entire training set of observations to compute the gradient $\nabla_{\theta}\mathcal{L}(\theta_t)$ (Géron, 2017). The gradient descent update rule is given by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) + \beta(\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}) \quad (4.43)$$

where $\boldsymbol{\theta}_t$ represents the model parameters at iteration t , α the learning rate, β the momentum and $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)$ the gradient of the cost function $\mathcal{L}$ with respect to the parameters $\boldsymbol{\theta}_t$.

There are two hyperparameters that must be chosen carefully: α , the learning rate, which is generally small, and β , the momentum, a scalar value that stabilizes the update so that it does not become too large from one iteration to the next (Géron, 2017).

4.2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a variant of Gradient Descent (GD) that also updates model parameters iteratively, but uses the gradient of the cost function $\mathcal{L}$ of a single randomly selected training data point instead of the entire training set (Géron, 2017). At the cost of increased noise in the parameter updates, this results in faster convergence and the ability to escape local minima (Géron, 2017). The SGD algorithm is similar to the GD algorithm in its update rule, as it is only a small modification of the GD algorithm:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}_t) \quad (4.44)$$

where $\mathcal{L}_i(\boldsymbol{\theta}_t)$ is the cost function of a single data point i , and the remaining variables have the same meaning as described in the previous section.

Note that the same comments that were made about the GD optimizer can be applied to this variant as well.

4.2.3.3 Mini-Batch Gradient Descent

The Mini-batch Gradient Descent (MGD) algorithm is a compromise between the Gradient Descent (GD) and Stochastic Gradient Descent (SGD), where the model parameters are updated using the gradient of the cost function $\mathcal{L}$ of multiple subsets of observations from the training set, rather than the entire set (Géron, 2017).

The entire training set is divided into equal-sized subsets called batches. Each batch is used sequentially to update the model parameters. Instead of a single update based on the entire training set (Géron, 2017), several updates are performed during each epoch. This approach is a combination of the advantages of GD and SGD. It strikes a balance between convergence speed and update noise. The update rule for mini-batch gradient descent is:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \times \frac{1}{|B_k|} \sum_{i \in B_k} \nabla_{\boldsymbol{\theta}} \mathcal{L}_i(\boldsymbol{\theta}_t) \quad (4.45)$$

where B_k is the set of data points in the mini-batch k , $|B_k|$ is the size of the mini-batch k , and the remaining variables have the same meaning as described in the previous sections.

Again, the same comments that have been made about the GD and SGD algorithms also apply to the MGD, as it is a variant of the GD algorithm.

4.2.3.4 Adam

Adam, which stands for Adaptive Moment Estimation, is an optimizer that combines the advantages of both momentum-based and adaptive learning rate methods (Géron, 2017; Kingma & Ba, 2017; Denuit et al., 2019).

Adam keeps track of previous gradients when updating parameters, unlike Gradient Descent (GD) algorithms. To keep track of these gradients, Adam uses an exponentially decaying average of previous squared gradients (Géron, 2017; Denuit et al., 2019):

$$\mathbf{m}_t \leftarrow \beta_{1,t} \mathbf{m}_t + (1 - \beta_{1,t}) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) \quad (4.46)$$

$$\mathbf{s}_t \leftarrow \beta_{2,t} \mathbf{s}_t + (1 - \beta_{2,t}) \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) \otimes \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) \quad (4.47)$$

$$\hat{\mathbf{m}}_t \leftarrow \frac{\mathbf{m}_t}{1 - \beta_{1,t}^t} \quad (4.48)$$

$$\hat{\mathbf{s}}_t \leftarrow \frac{\mathbf{s}_t}{1 - \beta_{2,t}^t} \quad (4.49)$$

$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \eta \times \hat{\mathbf{m}}_t \oslash \sqrt{\hat{\mathbf{s}}_t + \boldsymbol{\varepsilon}} \quad (4.50)$$

Because Adam uses adaptive learning rates for each parameter, the algorithm converges to the optimum faster than gradient descent algorithms (Kingma & Ba, 2017; Denuit et al., 2019). In addition, Adam handles sparse data or noisy gradients better than gradient descent algorithms, and does so with a smaller memory footprint (Kingma & Ba, 2017).

4.2.3.5 Remarks

Optimizers play a key role in the performance of neural network models such as hybrid models, MLPs and LSTMs. By efficiently tuning model parameters, optimizers help to improve the accuracy and generalization of neural network models (Géron, 2017).

In order to train effective neural network models, it is crucial to understand and select the appropriate optimizer. From Gradient Descent (GD) and its variants: SGD, MGD to advanced optimization techniques such as Adam and its variants: Nadam,

AdaMax, which are not discussed in this section, researchers and practitioners have many options to choose from.

4.2.4 Accuracy Measures

Accuracy measures play a crucial role in assessing the performance of forecasting models in practice, research and forecasting competitions. They quantify the differences between the forecasts and the actual values observed in the time series, allow model parameters to be fine-tuned and can be used for model comparison and selection. Several accuracy measures exist in the literature, each with its own strengths, weaknesses, limitations and conclusions (Koutsandreas et al., 2021).

Since the focus of this thesis is on the evaluation of forecasting models using point forecast (PF) accuracy measures, the following section will only cover the most common point forecast accuracy measures that are used both in practice and in the literature (Koutsandreas et al., 2021). Some of these measures have been used to assess models during forecasting competitions (Makridakis & Hibon, 2000; Makridakis et al., 2020; Koutsandreas et al., 2021).

4.2.4.1 Quantile Error

See point 4.1.5.5 for more details.

4.2.4.2 Mean Absolute Error (MAE)

The MAE is calculated as the average absolute difference between forecasts and actual observations. It is a scale-dependent measure of accuracy (Koutsandreas et al., 2021) and also a simple and easy-to-interpret measure of forecast error, suitable for cases where all errors should be treated equally regardless of their size:

$$MAE = \frac{1}{T} \sum_t |y_t - \hat{y}_t| \quad (4.51)$$

4.2.4.3 Mean Square Error (MSE)

Also a scale-dependent measure of accuracy, the MSE is calculated as the average of the squared difference between the predictions and the actual observations. The MSE is more sensitive to outliers than the MAE because the errors are squared before being averaged, giving more weight to larger errors (Koutsandreas et al., 2021):

$$MSE = \frac{1}{T} \sum_t (y_t - \hat{y}_t)^2 \quad (4.52)$$

4.2.4.4 Root Mean Square Error (RMSE)

The RMSE is the square root of the Mean Squared Error. Because it is easier to interpret than the Mean Squared Error, and has the same units as the original data, this measure of accuracy is frequently used in the evaluation of forecasts (Koutsandreas et al., 2021):

$$RMSE = \sqrt{\frac{1}{T} \sum_t (y_t - \hat{y}_t)^2} \quad (4.53)$$

4.2.4.5 Mean Absolute Percentage Error (MAPE)

The MAPE is calculated as the mean absolute percentage difference between predictions and actual observations. MAPE is a scale-independent and easy-to-interpret measure. It is useful for evaluating and comparing the results of a model over multiple time series of different units (Koutsandreas et al., 2021):

$$MAPE = \frac{100}{T} \sum_t \frac{|y_t - \hat{y}_t|}{y_t} \quad (4.54)$$

4.2.4.6 Symmetric Mean Absolute Percentage Error (sMAPE)

The sMAPE is an improved version of the MAPE that addresses some of its shortcomings, such as being undefined when actual values are equal to zero (R. J. Hyndman & Koehler, 2006; Koutsandreas et al., 2021). The sMAPE measure is widely used in forecasting competitions to evaluate forecasting models as it is a more balanced and complete measure of forecast error than the MAPE measure (R. J. Hyndman & Koehler, 2006):

$$sMAPE = \frac{200}{T} \sum_t \frac{(|y_t - \hat{y}_t|)}{(y_t + \hat{y}_t)} \quad (4.55)$$

4.2.4.7 Mean Absolute Scaled Error (MASE)

The Mean Absolute Scaled Error is considered by many experts to be the most convenient measure of model performance in forecasting (R. J. Hyndman & Koehler, 2006). The MASE is calculated as the mean absolute error (MAE) of the residuals first scaled by the in-sample MAE of a naive forecast. It is therefore particularly useful for assessing the performance of a model against a benchmark (Koutsandreas et al., 2021). In addition, the MASE allows you to compare the performance of a model over multiple time series at different scales (Koutsandreas et al., 2021).

Due to its robustness and ease of interpretation, the MASE has been widely used in competitions to assess forecasting models since its introduction during the M3 competition (Makridakis & Hibon, 2000):

$$q_t = \frac{e_t}{\frac{1}{(n-1)} \sum_{i=2}^N |y_i - y_{i-1}|} \quad (4.56)$$

$$MASE = \text{mean}(|q_t|) \quad (4.57)$$

where e_t is the residual at time t and N the number of observations in the training set (in-sample).

In the one-step-ahead context, the numerator of (4.56) is the in-sample mean absolute error of a one-step-ahead naive forecasting method. In a one-step-ahead naive forecasting model, the one-step forecast is equal to the last observation: $\hat{y}_{t+1} = y_t$.

4.2.4.8 Remarks

Forecasting competitions, such as the M competitions (Makridakis & Hibon, 2000; Makridakis et al., 2020), combine these accuracy measures to evaluate and compare the performance of forecasting models. This provides a comprehensive understanding of where each model excels and where it falls short in different circumstances. For example, the M4 competition used the symmetric mean absolute percentage error (sMAPE) as the primary evaluation measure and the mean absolute scaled error (MASE) as a secondary measure (Makridakis et al., 2020).

In order to effectively evaluate and compare the performance of different models, selecting the appropriate accuracy measure(s) for a given forecasting problem is critical (R. J. Hyndman & Koehler, 2006). By taking into account the specific requirements and characteristics of the problem at hand, accuracy measures can help in the identification of the most appropriate forecasting models (R. J. Hyndman & Koehler, 2006; Koutsandreas et al., 2021).

4.2.5 Time Series Validation Procedures

In traditional regression tasks, the standard measure used to evaluate models is the Root Mean Square Error (RMSE), and the most common splitting technique used to evaluate models is k-fold cross-validation (Bergmeir & Benítez, 2012). While this is the method of choice for traditional regression tasks, typical cross-validation approaches are not well suited for the evaluation of model performance with time series data (Bergmeir & Benítez, 2012; Bergmeir et al., 2018). This is because the assumptions that are made about the observed data in time series problems are different from those in traditional regression problems: whereas traditional regression problems assume that the observations are independent, time-series data assume that the observations are dependent on time (Bergmeir et al., 2018).

As the observations are time-dependent, they cannot be split using traditional splitting techniques, as this would lead to data leakage (Bergmeir et al., 2018). Indeed, when using K-fold cross-validation or other traditional validation procedures, K disjoint random samples of equal size are selected independently from the whole data set without considering the inherent correlation structure in the data (Bergmeir et al., 2018). In addition, not only do these procedures neglect the temporal structure

of the data, but they also assume that the observations are uniformly distributed, which is an inaccurate assumption for time series data (Bergmeir et al., 2018). However, thanks to the research community, alternative validation methods have been developed to address these issues. This section discusses two techniques commonly used in the literature to evaluate forecasting models.

4.2.5.1 Out-of-Sample Evaluation

An out-of-sample evaluation starts by splitting the time series into two sets, a training set and a test set (Tashman, 2000). This division is performed in accordance with the temporal nature of the data, the two sets are consecutive in the time and disjoint. In addition, if some hyperparameters need to be tuned or found, the initial training set is again divided into two sets, a smaller training set and a validation set. The validation set is used to find the best hyperparameters for the model and data available. To find these best hyperparameters, the model is first trained on the first set of data and then calibrated on the second set. The goal of such a split is to find the combination of hyperparameters that best fits the validation set, without using the test set as it is considered unavailable during model building. Grid search, random search or other methods can be used to find these optimal hyperparameters (Michelucci, 2018).

Once the best hyperparameters have been found, the entire training set with the found hyperparameters is used to train a final model. This final model is then finally evaluated on the test set using another out-of-sample procedure (Bergmeir et al., 2018).



Figure 4.13: Figure 2. Out-of-sample evaluation (p. 9). From Bergmeir, H., & Hyndman, R. J. (2012). A note on the validity of cross-validation for evaluating time series prediction. *Journal of Machine Learning Research*, 13, 281-286.

Although this validation procedure takes into account the inherent structure of the data, this methodology may not work well if there is not a lot of data or the number of parameters to be estimated is large (Bergmeir et al., 2018). In addition, this approach only gives a single assessment of the quality of the forecasting model, which can be very problematic in some situations where the predictive performance of the model is fundamental and needs to be well and robustly assessed (Bergmeir et al., 2018).

4.2.5.2 Walk Forward Validation Approach

As mentioned in the previous subsection, the out-of-sample evaluation approach has some drawbacks, one of which is that we have only one measure of the quality of the forecasting model (Bergmeir et al., 2018). As this might not be reliable enough to judge the quality of a model in some contexts, new techniques have been proposed to deal with this limitation. One of these techniques is the walk-forward validation approach, which was first introduced in 1999 and later revised in 2007 (Hu et al., 1999).

In this approach, there is not a single test set, but several ones (Hu et al., 1999). Each test set contains only one observation, and for each of them, the training set consists of all the observations that precede the one in the test set (Hu et al., 1999; Bergmeir et al., 2018).

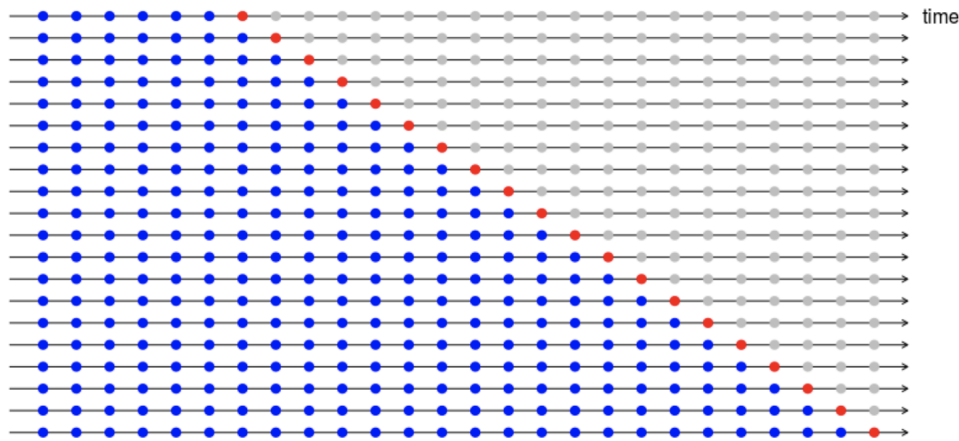


Figure 4.14: Walk-forward validation approach. Training set (blue) and test set (red). Reprinted from Hyndman, R. J., & Athanasopoulos, G. (2021). Cross-validation for time series. In *Forecasting: principles and practice* (3rd ed.). <https://otexts.com/fpp3/tscv.html>

This approach enables to have more than one measure of the predictive performance of the model (Bergmeir et al., 2018; R. Hyndman & Athanasopoulos, 2018). One model could be built using the first 10 observations, the last being the test set, then another using the first 11 observations, the last being the test set again, and so on until no test set observation remains. In the end, there are as many measures as there are test sets. This process also makes it possible to compare performance over different time horizons for one or more steps ahead (R. Hyndman & Athanasopoulos, 2018).

Finally, an average of these measures is taken to quantify the quality of the predictive model (Hu et al., 1999; Bergmeir et al., 2018). This average is based on more than one value, which ensures a lower variance and a more accurate measure of the quality of the model².

²Note that the approach can also be used for multi-step forecasting

If some hyperparameters need to be found, the training set is again divided into two subsets, with the most recent observations being the validation set and the previous ones being the training set (Bergmeir et al., 2018).

4.2.5.3 Remarks

In the literature, the walk-forward approach is a recommended method when the conditions are suitable to use it for time series data (Bergmeir et al., 2018; R. Hyndman & Athanasopoulos, 2018). The walk-forward approach is a consistent procedure that can provide multiple error measures and easy-to-interpret results (Bergmeir et al., 2018; Michelucci, 2018).

Therefore, it will be the method that we will use later in the study to validate and test the models.

4.3 Conclusion

In this chapter, we covered all the key technical concepts that will play an important role in the following chapters. From models and how they operate to how they are trained, validated and tested, this chapter has provided a broad overview of all the concepts that will be used to conduct the study in the following chapters.

The following chapters explore the research question in more depth. First, the experimental setting is presented in order to align the study with the research question. Then, the results of the empirical experiment are discussed in depth and contrasted with findings from the literature. Finally, a conclusion summarising the findings of the thesis and future work that could be explored will be provided to close the thesis.

Chapter 5

Experimental Settings

This chapter presents all the settings and decisions that were used and made to conduct the study and answer the research questions.

5.1 Libraries

The study was conducted using a number of libraries:

- Pandas, NumPy, Matplotlib and Seaborn were the main libraries used to manage, prepare and visualise the time series.
- The Statsmodels API library was used to implement and build the statistical methods used as benchmarks in the study.
- The Tensorflow (2.9.0) and Keras (2.9.0) libraries were used to implement and build the machine and deep learning models. While Tensorflow was mainly used to exploit the power of the GPU, the Keras library was used to implement the models.
- For the last model, the Exponential Smoothing Recurrent Neural Network (ES-RNN), the ESRNN library was used. The ESRNN library is a PyTorch implementation of the ES-RNN model proposed by Smyl during the M4 competition. According to the documentation and literature, this library is a reliable and trustworthy implementation of the model in Python (Redd et al., 2019).
- All scripts were written using Python 3.8.12.

5.2 Datasets

Since the purpose of the thesis is to study the performance of machine learning, deep learning, and hybrid models in different situations, four different time series were selected to achieve this purpose. These series are presented below.

For each time series, information is also provided on the sample size of the sets used for model training, validation, and testing.

5.2.1 Mauna Loa's CO₂ emissions between 1965 and 1980

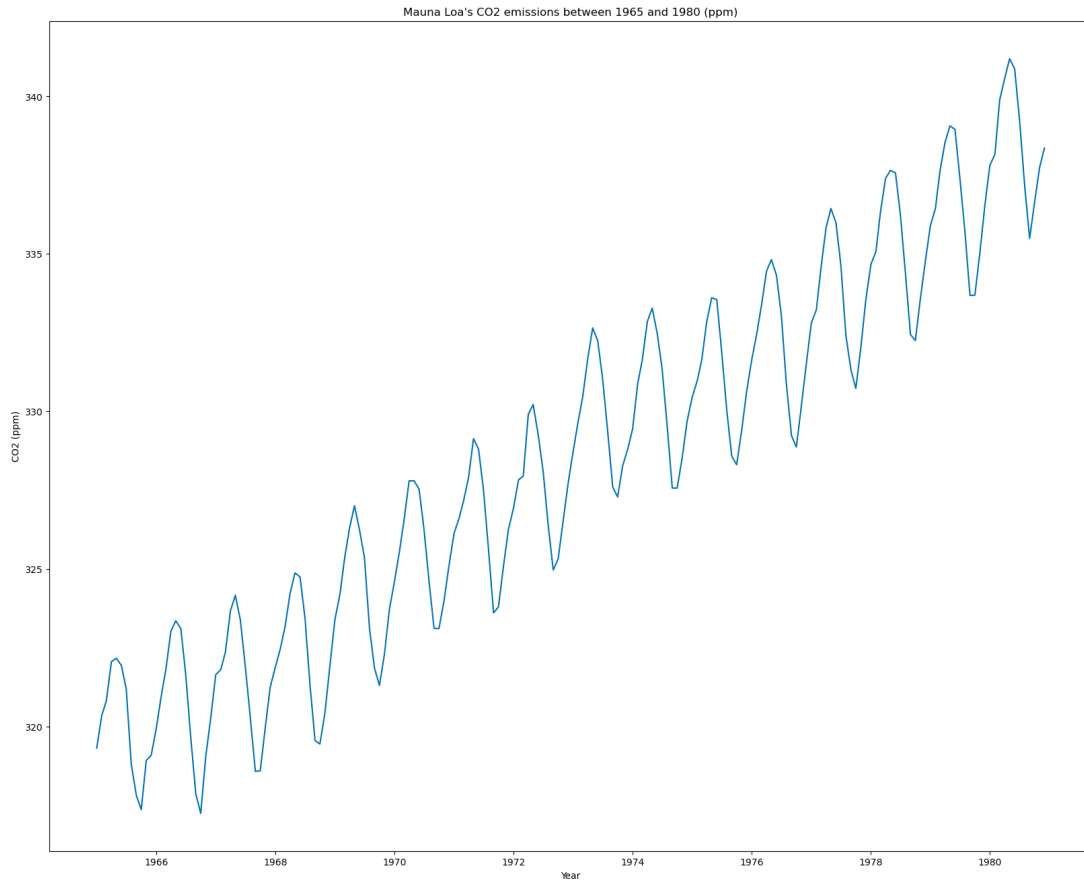


Figure 5.1: Mauna Loa's CO₂ emissions between 1965 and 1980 (ppm).

Characteristics

- Sample size: 192 observations.
- Seasonality period (m): 12.
- Sample size validation set: 24 observations (two cycles).
- Sample size test set: 24 observations (two cycles).

5.2.2 Electricity production in Germany between 1955 and 1980

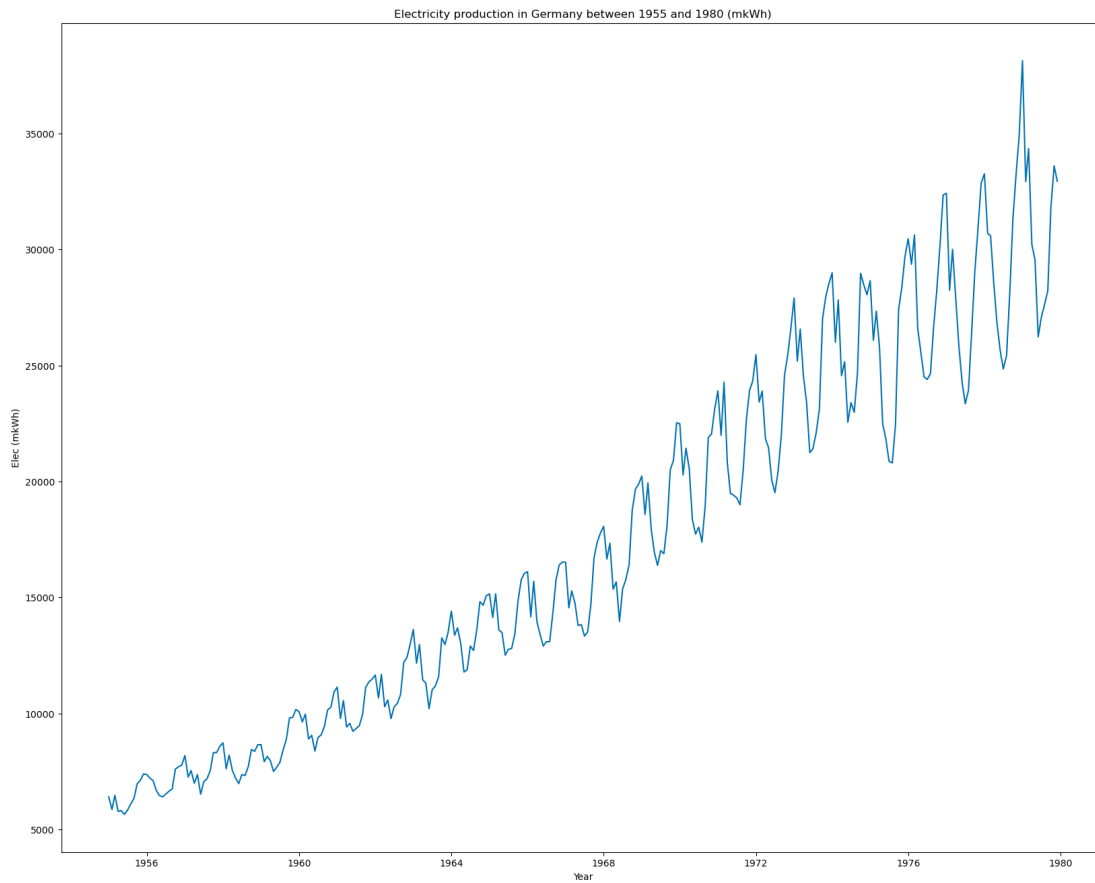


Figure 5.2: Electricity production in Germany between 1955 and 1980 (mkWh).

Characteristics

- Sample size: 300 observations.
- Seasonality period (m): 12.
- Sample size validation set: 24 observations (two cycles).
- Sample size test set: 24 observations (two cycles).

5.2.3 Iron production in Australia between March 1956 and September 1994

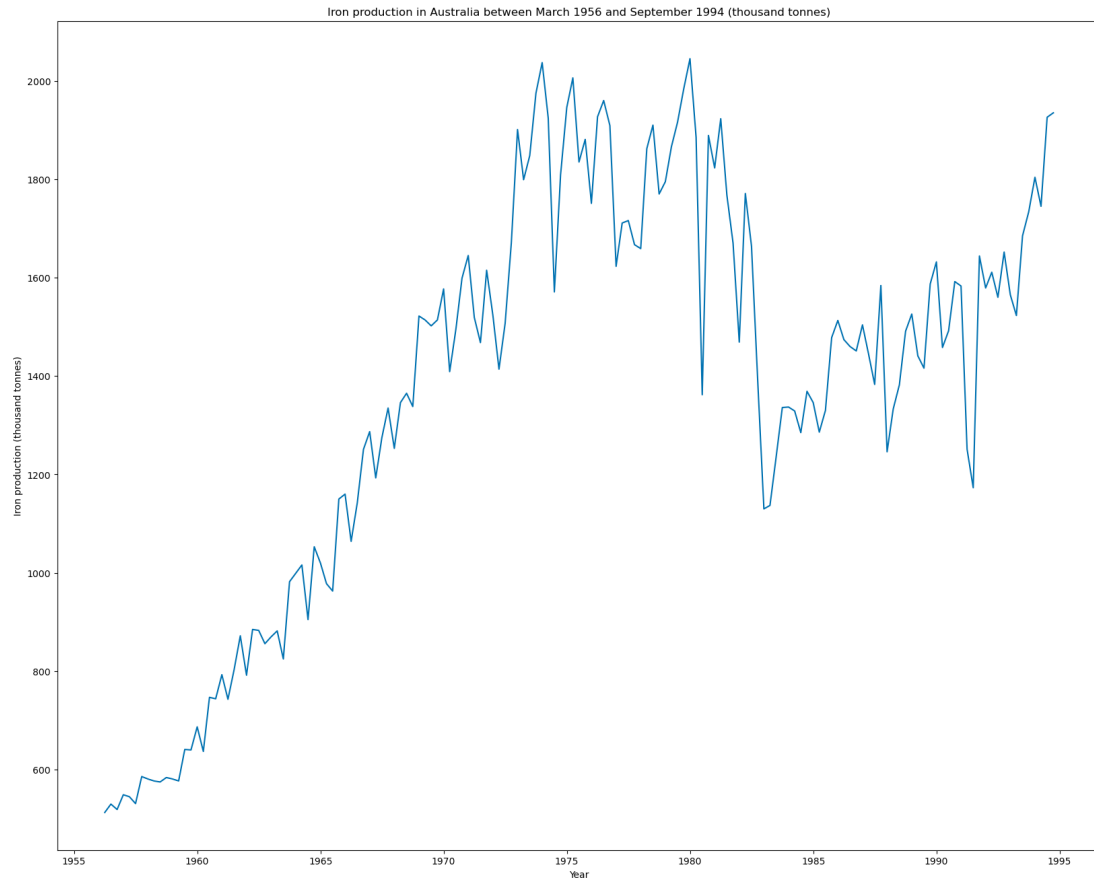


Figure 5.3: Iron production in Australia between March 1956 and September 1994 (thousand tonnes).

Characteristics

- Sample size: 155 observations.
- Seasonality period (m): 4.
- Sample size validation set: 16 observations (four cycles).
- Sample size test set: 16 observations (four cycles).

5.2.4 Minimum daily temperatures over 5 years (1986-1990) in the city of Melbourne, Australia

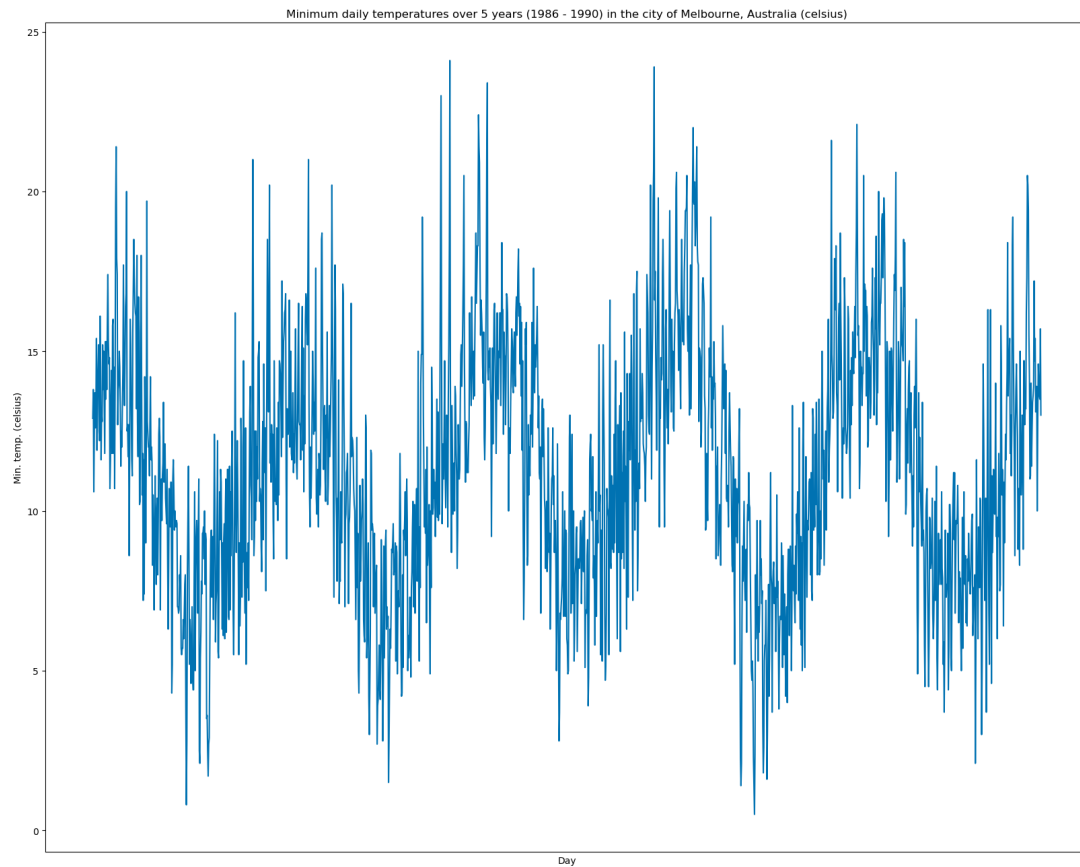


Figure 5.4: Minimum daily temperatures over 5 years (1986-1990) in the city of Melbourne, Australia (celsius).

Characteristics

- Sample size: 1825 observations.
- Seasonality period (m): 7.
- Sample size validation set: 365 observations (fifty-two cycles).
- Sample size test set: 365 observations (fifty-two cycles).

5.3 Training, Validation and Testing

The reader should note that the root mean square error (RMSE) (see point 4.2.4.4) was used to train, validate and test all the models. The only exception among the models is the Exponential Smoothing Recurrent Neural Network (ES-RNN) model. Unlike the others, and as discussed in the previous chapter, this model is not trained using the root mean square error (RMSE) (see point 4.1.5.5). Note, however, that like the others, it has been validated and tested using it.

In addition, as mentioned above (see point 4.2.5.2), a one-step walk-forward validation procedure was used to validate the models. The same procedure was used for the testing phase. This choice ensured that all the models used the same observations at each stage and that their results were then comparable.

Training, Validation, and Test Sets - A Reminder

Before we look at the benchmark results for the selected time series, let us take a moment to refresh our understanding of the critical roles played by the training, validation and test sets in data science. To illustrate their role, we will use the example of an autoregressive process of order p , commonly referred to as AR(p).

Modelling an AR(p) process involves a crucial decision even before fitting the model: determining the value of p . The choice of p defines the structure of the autoregressive process by influencing the number and values of the coefficients involved in the model. In essence, p acts as a hyperparameter for the AR(p) process, it must be set prior to fitting the model:

$$y_t = \mu + \sum_{p=1}^P a_p x_{t-p} + \varepsilon_t \quad (5.1)$$

In data science, hyperparameters are components of the model that require pre-configuration before the learning algorithm is applied (Goodfellow et al., 2016; Géron, 2017). They are like preconditions that must be set before you can learn the parameters of the model from the data (Goodfellow et al., 2016). For example, in neural network (NN) models, critical hyperparameters include the number of neurons, the number of hidden layers, and the learning rate (Goodfellow et al., 2016). On another note, seasonal autoregressive integrated moving average (SARIMA) models include hyperparameters such as p , d , q , P , D , Q , and m .

Since the training set is already used to obtain model parameters, such as coefficients in AR(p) or weights in neural network (NN) models, it is not convenient to use it again to identify optimal hyperparameter values. This leads to misleading assessments of model performance, as both coefficients and hyperparameters are fitted to the same set.

To overcome this dilemma, the series is partitioned into two subsets: a training set and a validation set. This partitioning is critical to effectively address the challenge and ensure that the performance of the model is evaluated objectively and reliably (Goodfellow et al., 2016; Géron, 2017).

The validation set plays an essential role in fine-tuning the hyperparameters. While the training set is used to learn the parameters of the models, the validation set is used to evaluate the performance of the model, taking into account both the parameters and the hyperparameters (Goodfellow et al., 2016). Once the optimal hyperparameter values have been determined based on the performance on the validation set, it is important to take the evaluation one step further (Goodfellow et al., 2016; Géron, 2017). This is where a third set, the test set, comes into play (Goodfellow et al., 2016; Géron, 2017).

During the model development and hyperparameter tuning phases, the test set remains completely untouched. It serves as a completely independent set that reflects real-world scenarios (Goodfellow et al., 2016; Géron, 2017). This separation ensures that the final evaluation of the model's performance is based on completely new and unseen data (Goodfellow et al., 2016; Géron, 2017). By evaluating the model on the test set, practitioners and researchers gain a realistic understanding of how well the model generalizes to novel situations, without any bias introduced by the training or validation processes (Goodfellow et al., 2016; Géron, 2017).

Note, however, that if the model does not include hyperparameters in its modelling, the series will only be divided into two sets: a training set and a test set (Goodfellow et al., 2016; Géron, 2017). As there are no hyperparameters to configure, there is no need to use a validation set, all parameters are learned on the training set and the model is tested on the test set (Géron, 2017).

Finally, a brief summary of the roles of the three sets:

- **Training Set:** This set is used for model fitting and parameter learning.
- **Validation Set:** This set is used to evaluate the performance of the model and to make informed decisions about the choice of hyperparameters.
- **Test Set:** This set remains untouched during training and validation and is used to evaluate the ability of the model to generalize to new, unseen data.

5.4 Benchmarks

Finally, we discuss how the benchmark models performed to see what the machine learning, deep learning, and hybrid models need to outperform to be considered better in the given experimental study context.

Two statistical models, Holt-Winters (HW) and Seasonal Autoregressive Integrated Moving Average (SARIMA), have been used as benchmarks in this study. While the next chapter, which is also the main chapter of this thesis, discusses and compares the results of all the models, this section only presents the results of the benchmarks.

The following optimal configurations were found for the Holt-Winters and SARIMA models:

Table 5.1: Holt-Winters (HW) Model Parameters

Dataset	Trend	Seasonal	Damped Trend	Boxcox	Bias
CO2 Dataset	Mul.	Mul.	False	False	True
Electricity Dataset	Add.	Mul.	True	False	False
Iron Production Dataset	Mul.	Add.	True	False	True
Daily min. temperature Dataset	Mul.	Add.	True	False	True

It is important to note that the statsmodels library offers a number of trend options, including additive, multiplicative and damped trends (Seabold & Perktold, 2010).

The choice of an additive trend incorporates a linear trend component into the model, whereas the choice of a multiplicative trend incorporates an exponential trend. In addition, the ability to include a damped trend, although not a requirement, allows the trend to be systematically damped over time, resulting in a more complex representation of the underlying temporal patterns within the series (Seabold & Perktold, 2010; R. Hyndman & Athanasopoulos, 2018).

Finally, the Boxcox setting is used to determine whether a Boxcox transformation is used before fitting the model (Seabold & Perktold, 2010; R. Hyndman & Athanasopoulos, 2018). On the other hand, the Bias setting is used to remove bias from the predicted and fitted values, ensuring that the mean residual is forced to zero (Seabold & Perktold, 2010).

Table 5.2: Seasonal Autoregressive Integrated Moving Average (SARIMA) Model Parameters

Dataset	p	d	q	P	D	Q
CO2 Dataset	1	2	2	1	1	0
Electricity Dataset (boxcox)	3	0	3	2	1	3
Iron Production Dataset	2	0	2	3	1	3
Daily min. temperature Dataset	1	1	2	1	0	0

Surprisingly, although we tested several values for the hyperparameters $d = \{0, 1, 2, 3\}$ and $D = \{0, 1, 2\}$, some of the optimal models we found in terms of minimizing the root mean square error (RMSE) did not set values greater than 0 for these parameters. In fact, although some of the series studied have strong seasonal trends, some of the optimal models we found did not require values of d and D greater than 0 to be the best models in terms of forecasting accuracy.

The results of the optimal models found are shown below:

Dataset	Holt-Winters		SARIMA	
	Validation set	Test set	Validation set	Test set
CO2 Dataset	0.2696	0.3922	0.2590	0.3878
Electricity Dataset	937.3489	1081.4712	933.6270	1045.1695
Iron Production Dataset	98.8693	147.1269	95.9448	143.3327
Daily min. temperature Dataset	2.3424	2.4967	2.1625	2.2706

Table 5.3: Results of the Benchmarks (RMSE)

Table 5.3 shows that the Seasonal Autoregressive Integrated Moving Average (SARIMA) model outperformed the Holt-Winters (HW) model on all datasets. Depending on the dataset, the difference between the two models is between 1% and 10%.

Based on the following equation:

$$\hat{e}_t = y_t - \hat{y}_t \quad (5.2)$$

In order to better characterize the quality of the forecasts made by the two models, several histograms have been generated. These histograms allow us to gain a better understanding of the models' residuals and their tendency to under- or over-estimate¹:

¹The reader should note that these histograms were generated using the predictions from both the validation and test sets. Since good generalization is always associated with a good initial fit, we thought it would be interesting to combine the two sets to assess both generalization and goodness of fit together in this chapter.

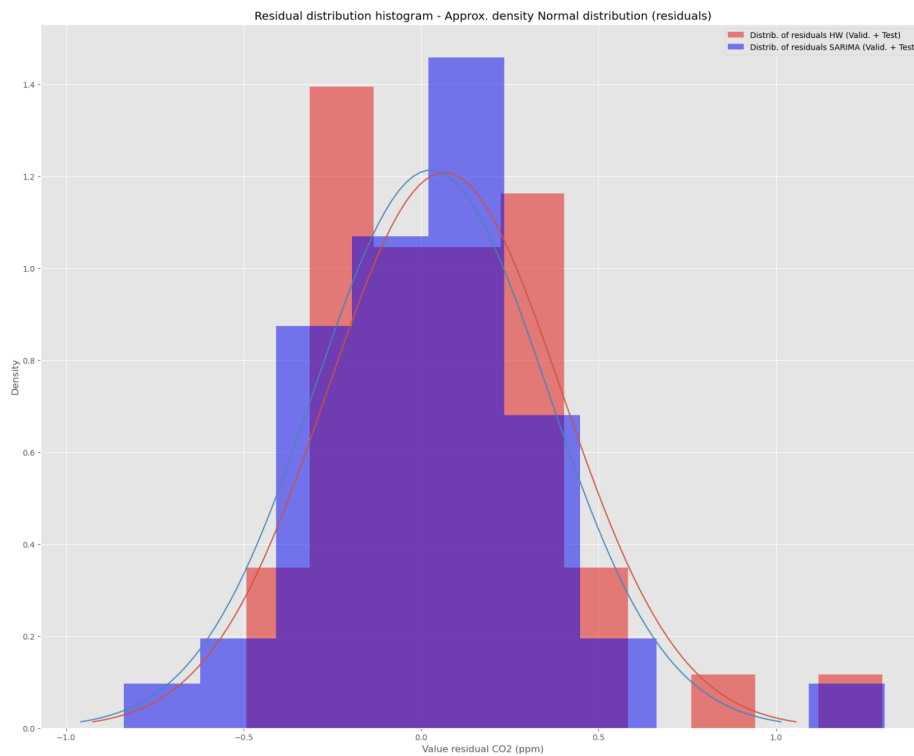


Figure 5.5: Histogram about SARIMA and Holt-Winters Models - CO2 - Validation and Test sets - Aggregated

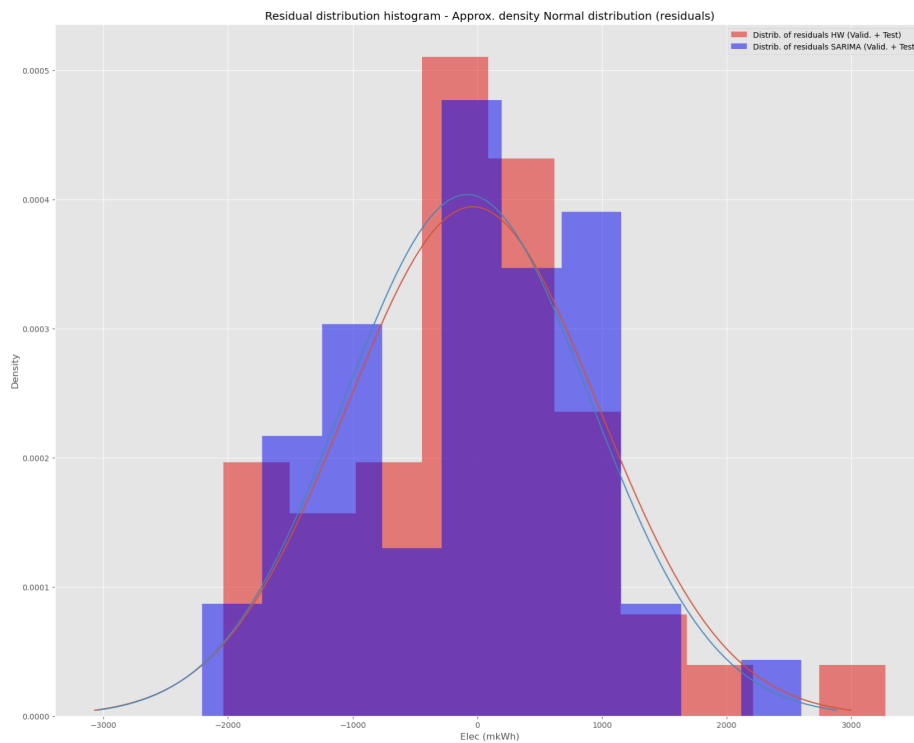


Figure 5.6: Histogram about SARIMA and Holt-Winters Models - Electricity - Validation and Test sets - Aggregated

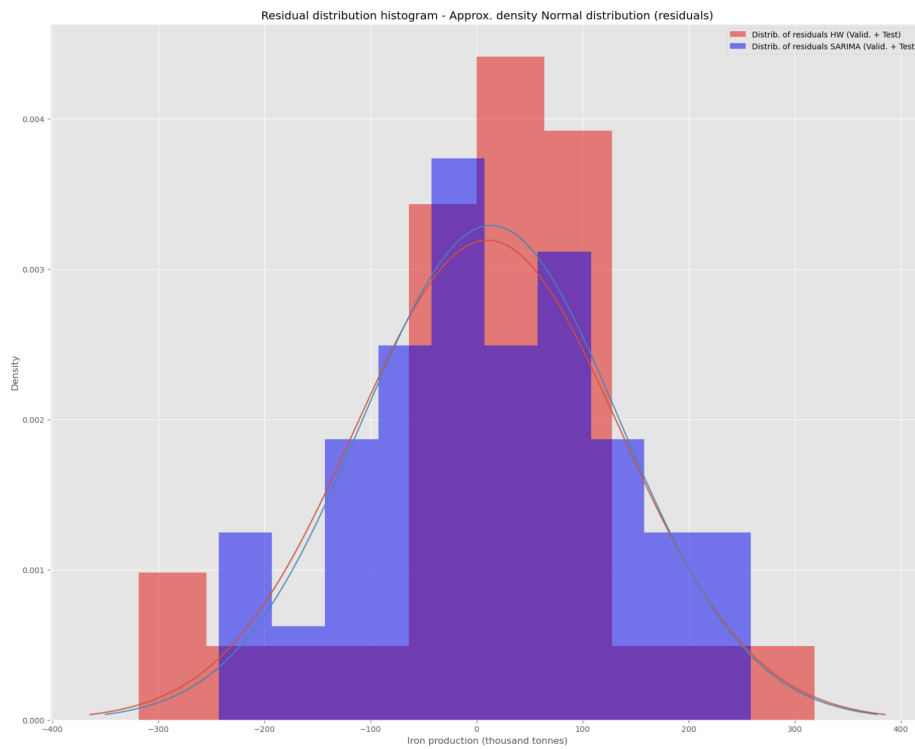


Figure 5.7: Histogram about SARIMA and Holt-Winters Models - Iron Production - Validation and Test sets - Aggregated

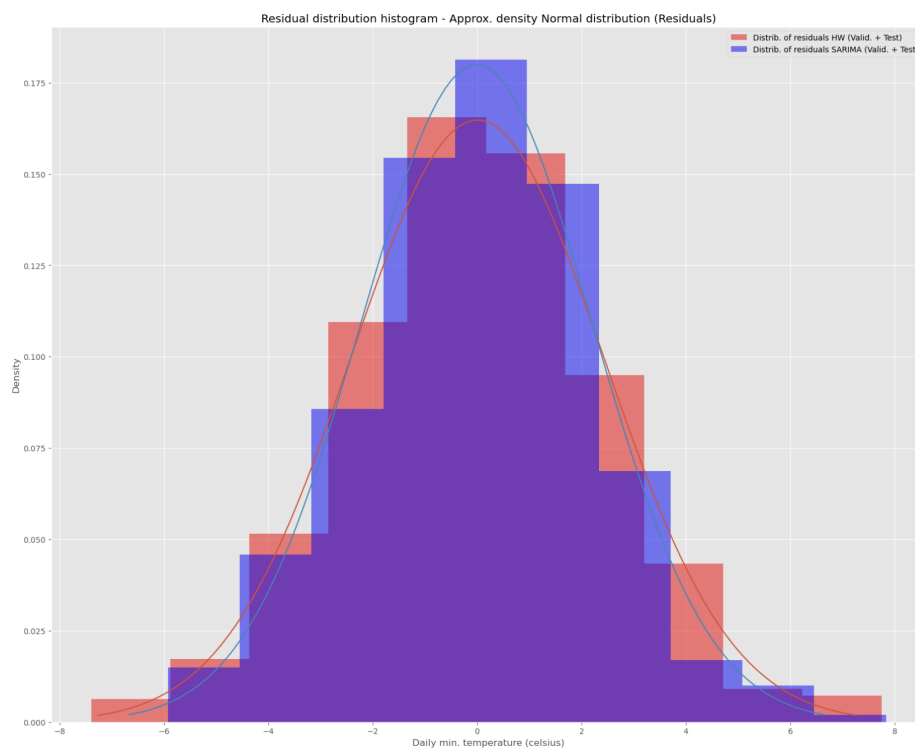


Figure 5.8: Histogram about SARIMA and Holt-Winters Models - Daily min. temperature - Validation and Test sets - Aggregated

In general, the Seasonal Autoregressive Integrated Moving Average (SARIMA) model outperformed the Holt-Winters (HW) model in terms of both forecast accuracy and consistency. In particular, the SARIMA forecasts showed less variability and smaller errors than those produced by the Holt-Winters model. In addition, the SARIMA forecasts were closer to 0 than those of the Holt-Winters forecasts, as its residual distributions are more concentrated around 0 than those of the Holt-Winters.

The histograms clearly illustrate a key factor in the success of the SARIMA: its ability to deal effectively with errors. While the Holt-Winters model had larger errors, which significantly affected its overall performance and resulted in it being ranked second in the comparison, the SARIMA model consistently demonstrated superior error management across all datasets.

It is important to emphasise that the results obtained are quite straightforward, as the time series studied have no evidence of breakpoints at their ends. In practical scenarios where there are no breakpoints, parametric approaches such as SARIMA often outperform non-parametric alternatives such as Holt-Winters.

In the end, the SARIMA model emerged as the best choice among the benchmarks for forecasting the selected datasets. Its superior performance and effective error management ensured consistent and accurate forecasts across the different datasets.

5.5 Conclusion

In this chapter we outlined all the settings we used to conduct the study, the results of which are presented in the following chapter. From the libraries and time series we used to the way we carried out the training, validation, and testing procedures, the chapter provided a clear overview of all the decisions and settings needed to understand and carry out the study. The chapter ended with some findings and results we noticed about the benchmark models.

In the next chapter, which is also the main chapter of this thesis, we examine the results of the machine learning, deep learning, and hybrid models and see if they were able to outperform the results of the benchmarks.

Chapter 6

Study Results

To address the challenges of tuning neural network models, the chapter is divided into two parts.

In the first part, entitled Models (see section 6.1), the machine learning, deep learning, and hybrid models are discussed separately. As sensitivity analyses were performed on these models, we also discuss some of the findings that emerged from these analyses.

In the second part, called Results (see section 6.2), once the best models have been identified for each dataset, a comparison of their performance is provided, dataset by dataset, on the validation and test sets.

By first identifying the best models for each dataset and then comparing these models, we ensure that an optimal model for each dataset can be identified from those we have been able to explore.

6.1 Models

To perform sensitivity analyses, we first explored approximately 800 different configurations for each combination of model and dataset. This was done to identify hyperparameters that were likely to be locally optimal for each model-dataset pair.

We then deliberately explored configurations close to these assumed local optimums to discover potential improvements.

In the sensitivity analyzes, the following hyperparameters were examined:

- The number of timesteps (inputs)
- The number of epochs
- The size of the batches

- The number of nodes in the hidden layers
- The degree of dilation (ES-RNN)

Note that for all neural network models, the activation functions used were the tangent function in the hidden layers and the linear function in the output layer.

The reader should also bear in mind that the results of neural network models are stochastic and may vary from one study to another. To ensure the validity and accessibility of the results presented here, we performed multiple runs.

In addition, given the inherent impossibility of finding global optima for neural network models, it is crucial to be aware that the results of neural networks are only local optima and not global optima.

Finally, it should be noted that the sensitivity analyses were carried out on the validation sets and the results can be found in the appendices. The only model for which we do not provide sensitivity analysis results is the LSTM-Daily pair. As the training and validation of this model proved too time-consuming, we have not been able to provide further analysis for this model-dataset combination.

6.1.1 Multilayer Perceptron

The following table shows the optimal configurations we found for the Multilayer Perceptron (MLP) model after the validation and sensitivity analysis steps. These configurations are those that gave the best results during these steps and were therefore those used during the testing phase:

Table 6.1: Optimal Local Configurations - Multilayer Perceptron - Hyperparameters

Hyperparameter	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Inputs	15	27	12	7
Nodes	32	64	512	64
Hidden Layers	3	1	1	1
Epochs	30	370	120	20
Batch Size	16	8	16	256
Trend Diff.	1	1	0	1
Seasonal Diff.	1	0	0	0
Optimizer	SGD	Adam	SGD	SGD
Learning Rate	0.001	0.0001	0.0001	0.01
Standardized	True	True	False	False
Log Transformation	False	False	True	False

Table 6.2: Multilayer Perceptron - Ranks and Scores - RMSE

Validation Set	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Rank	2nd	1st	1st	2nd
Score	0.2457	868.3411	74.1208	2.2240

Surprisingly, the MLP model produced some of the best results during the validation and sensitivity analysis steps. It was also the fastest of the neural network models to train, validate and evaluate.

In the sensitivity analyses, we found that the MLP was very sensitive to variations in the values of its hyperparameters. When we slightly changed the values of any of the hyperparameters studied: the number of inputs, nodes, epochs or the size of the batches, the results of the MLP varied considerably. Therefore, we declared the MLP as the most sensitive neural network during the sensitivity analysis procedures. Note that the hyperparameters that most influenced its results were the size of the batches and the number of nodes.

Based on our research and the results obtained during the validation phase, there is a belief that the MLP model tends to overfit some of the datasets. However, it is important to note that this assumption remains tentative as we have only examined the results on the validation sets in this section.

To confirm or refute this hypothesis, a thorough analysis of the model's performance on the test sets is required. The test set evaluation will provide a more conclusive assessment of whether or not the Multilayer Perceptron (MLP) model does indeed suffer from overfitting.

6.1.2 Long Short-Term Memory

Table 6.3: Optimal Local Configurations - Long Short-Term Memory - Hyperparameters

Hyperparameter	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Inputs	22	12	24	21
Nodes	256	64	16	64
Hidden Layers	1	1	3	2
Epochs	90	270	35	30
Batch Size	16	32	1	64
Trend Diff.	0	1	0	0
Seasonal Diff.	1	0	0	0
Optimizer	SGD	Adam	Adam	Adam
Learning Rate	0.01	0.0001	0.00001	0.001
Standardized	True	True	True	False
Log Transformation	False	True	True	False

Table 6.4: Long Short-Term Memory - Ranks and Scores - RMSE

Validation Set	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Rank	4th	5th	3rd	5th
Score	0.2645	999.0565	94.3165	2.4807

In contrast to the Multilayer Perceptron (MLP) model, the Long Short-Term Memory (LSTM) model took the longest to train, tune and test. The LSTM was also one of the worst performers in the validation and sensitivity analysis steps.

It should be noted that in the sensitivity analyses, the LSTM model did not respond as dramatically to hyperparameter variations as the MLP model.

Although the LSTM did not perform as well as the MLP, it proved to be more stable than the MLP. However, the way in which the variations affected the results of the LSTM was similar to that of the MLP. Only the magnitude of the responses was different. Furthermore, as with the MLP, the hyperparameters that most influenced the results of the LSTM were the number of nodes and the size of the batches.

During the research, it was observed that the LSTM model encountered a glass ceiling¹ in its ability to generate accurate forecasts and thus achieve high scores in the validation and sensitivity analyses. Specifically, for certain datasets, such as Electricity and Daily min. temperature, the LSTM model struggled to match the performance of the best-performing models. However, in the case of the Iron Production dataset, the LSTM model performed well and achieved a position within the top 3 performers.

Given the observation that the LSTM model did not fit most of the datasets well, we suspected that the LSTM, unlike the MLP and Exponential Smoothing Recurrent Neural Network (ES-RNN) models (as seen in section 6.1.3), tends to underfit datasets. This conclusion follows from the poor performance of the model during the validation and sensitivity analysis steps.

Later in this chapter, we will examine the results of the Long Short-Term Memory (LSTM) model on the test sets to see if there is indeed evidence of underfitting. The evaluation on the test sets will provide a more definitive assessment of the performance of the model and its ability to generalize to unseen data.

¹This phenomenon means that there is a threshold beyond which it becomes difficult to improve forecast accuracy. Essentially, it marks a point where the forecasting model has peaked and no significant improvements in accuracy can be made.

6.1.3 Exponential Smoothing Recurrent Neural Network

Table 6.5: Optimal Local Configurations - Exponential Smoothing Recurrent Neural Network - Hyperparameters

Hyperparameter	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Inputs	48	78	32	35
Nodes	48	66	32	16
Cell Type	RNN	GRU	LSTM	LSTM
Dilation	[1, 4, 8, 16]	[1, 4, 8, 16]	[1, 3, 6, 12]	[1]
Epochs	90	65	35	35
Learning Rate	0.001	0.01	0.1	0.1
Percentile	50	50	40	50
Normalization Layer	True	True	True	True
Log Transformation	False	True	False	False

Table 6.6: Exponential Smoothing Recurrent Neural Network - Ranks and Scores - RMSE

Validation Set	Dataset			
	CO2	Electricity	Iron Production	Daily min. temperature
Rank	1st	4th	2nd	4th
Score	0.2342	960.8040	88.8869	2.4497

Although the Exponential Smoothing Recurrent Neural Network (ES-RNN) model was one of the simpler models to train, tune and evaluate during the study, it did not fully meet the expectations we set for it in the context of this thesis.

Similar to the Long Short-Term Memory (LSTM) model, we found that the ES-RNN encountered a glass ceiling on one of the datasets, the Daily min. temperature dataset. Furthermore, where it did not, it often showed patterns of overfitting, e.g. the CO2 and Iron Production datasets, as its results were usually much better than those of the other models.

The study also showed that the ES-RNN behaved consistently throughout the sensitivity analyses, with little impact from parameter changes. Nevertheless, it is important to highlight that the results of the ES-RNN model showed explosive behavior for certain hyperparameter values studied.

Unfortunately, we suspect that the Exponential Smoothing Recurrent Neural (ES-RNN), like the Multilayer Perceptron (MLP), tends to overfit the datasets².

²It is important to note that the implementation of the Exponential Smoothing Recurrent Neural Network (ES-RNN) model in the Python ESRNN package had limitations that affected our ability to fully exploit all of its capabilities. In particular, the incomplete documentation on how to use certain features limited our ability to exploit its full potential. As a result, the quality of its predictions, and therefore its overall results, were limited.

6.2 Results

In this section, we will compare the results of the best models obtained for each dataset.

To ensure a comprehensive analysis, the section is divided into two parts. First, we will examine the results of the validation sets to understand the performance of the models during the validation phase. Second, we will look at the results of the test sets to evaluate the generalization and performance of the models on unseen data.

6.2.1 Validation Set Results

Table 6.7: Validation results (RMSE)

Models	CO2	ELECTRICITY	IRON PRODUCTION	DAILY MIN. TEMP
Holt-Winters	0.2696	937.3489	98.8693	2.3424
SARIMA	0.2590	933.6270	95.9448	2.1625
MLP	0.2457	868.3411	74.1208	2.2240
LSTM	0.2645	999.0565	94.3165	2.4807
ES-RNN	0.2342	960.8040	88.8869	2.4497

Based on the discussions in section 6.1 and the information presented in Table 6.7, it is already clear that there is no single model that can be universally regarded as the best for all datasets in the given study context. Looking at the validation results, it is clear that some models performed better on certain datasets than others.

As discussed in section 6.1.2, the Long Short-Term Memory (LSTM) model struggled to fit some of the datasets effectively, probably due to its complexity. There were exceptions to this pattern, as demonstrated by its competitive performance on the Iron Production dataset.

Conversely, the Multilayer Perceptron (MLP) model showed consistent performance, consistently ranking in the top three during the validation phase (as seen in section 6.1.1). Nevertheless, this remarkable performance raised concerns about the possibility of overfitting the datasets. While the ability to consistently achieve high scores on the validation sets is admirable, it is important to carefully examine the ability of the model to generalise to unseen data to address potential overfitting issues.

During the validation phase, the Exponential Smoothing Recurrent Neural Network (ES-RNN) model showed superior performance, particularly for the CO2 and Iron Production datasets. However, it is important to note that the overall performance of the ES-RNN during validation was mixed.

Although the ES-RNN showed strong and competitive performance on the CO2 and Iron Production datasets, its results on the Electricity and Daily min. temperature datasets raised concerns about its suitability for the specific study context.

To gain a full understanding of the performance of the Exponential Smoothing Recurrent Neural Network (ES-RNN) model, it is crucial to examine its results on the test datasets. Evaluating how well the model performs on previously unseen data will provide valuable insights into its ability to generalize and make accurate predictions in different scenarios.

6.2.2 Test Set Results

This section is in two parts. The first is an illustrative analysis of the models' predictions for each dataset. These visual representations provide valuable insights into the ability of the models to fit the data and capture inherent patterns.

The next section discusses the results of the test sets and concludes with concluding remarks. Examining the test set results provides a more robust assessment of the models' ability to generalize and their overall effectiveness in predicting unobserved data.

6.2.2.1 Model - Dataset Results

Before proceeding to a detailed analysis of the results of the models on the test sets, it is useful to make a visual comparison of the model predictions between the validation and test sets. This comparison will help to identify notable differences in the performance of the models when dealing with data encountered during validation as opposed to completely new data.

6.2.2.1.1 CO2 Dataset

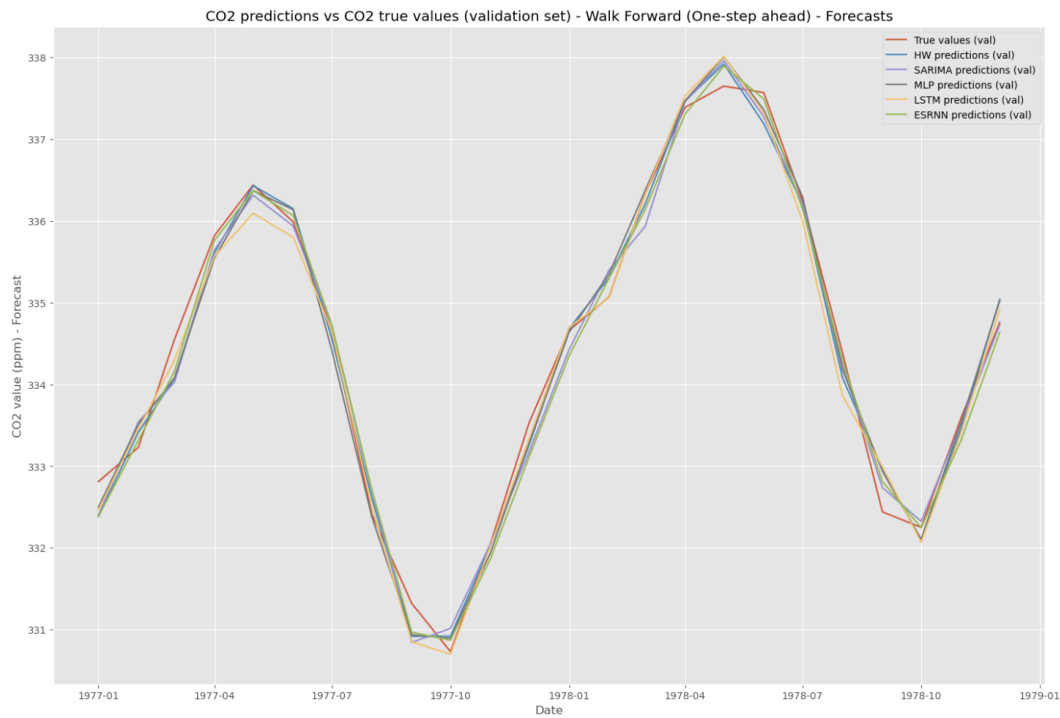


Figure 6.1: CO2 dataset - Validation set - Forecasts

When evaluating the models on the CO2 validation dataset, the Exponential Smoothing Recurrent Neural Network (ES-RNN), Multilayer Perceptron (MLP), and Seasonal Autoregressive Integrated Moving Average (SARIMA) models emerged as the best performers, excelling at capturing observations associated with peaks. Despite the challenge, these three models showed superior performance in this regard.

In particular, the ES-RNN model stood out among the three, showing remarkable accuracy at the peaks. Unlike the other models, it showed minimal underestimations or overestimations, making its predictions a more accurate representation of the true observations. This aspect contributed to the overall effectiveness of the ES-RNN model on the CO2 dataset during the validation phase.

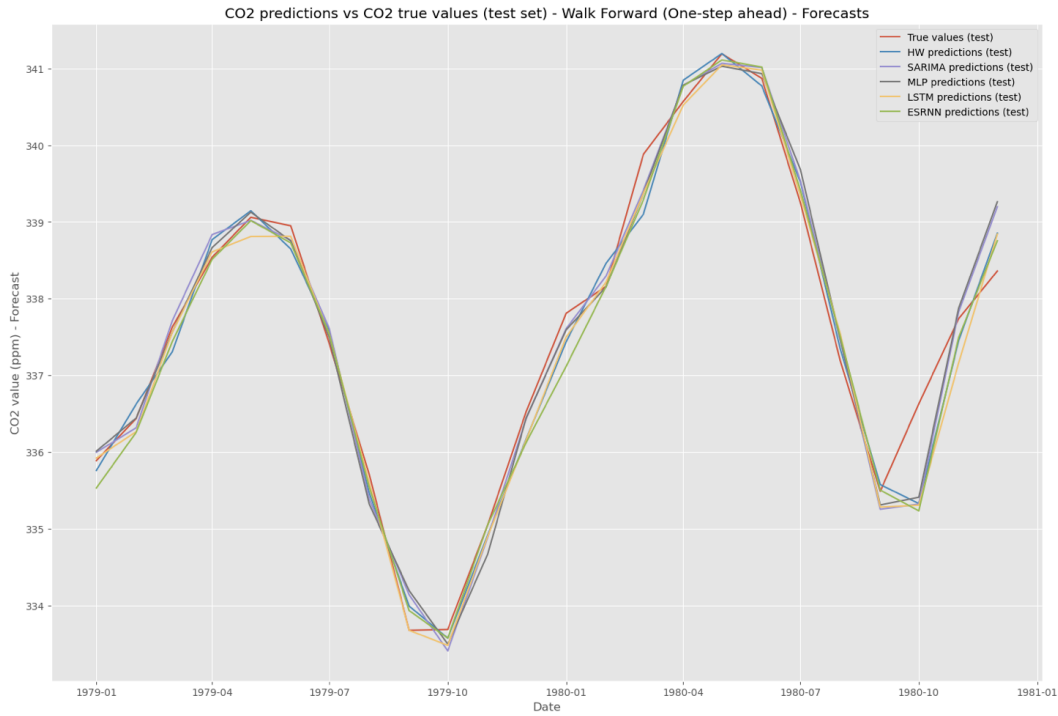


Figure 6.2: CO2 dataset - Test set - Forecasts

Unfortunately, despite its impressive performance during the validation, the Exponential Smoothing Recurrent Neural Network (ES-RNN) model did not maintain its superiority during the test phase, as shown in Figure 6.2. In fact, it struggled to adapt to the test set and had difficulty in capturing the flat regions as effectively as it did during validation. This discrepancy between its performance on the validation and test sets suggests that the ES-RNN may have encountered generalization problems during the validation phase.

Conversely, the Long Short-Term Memory (LSTM) model, which performed moderately in the validation phase, emerged as the best performer in the test phase. By striking a balance between fitting the flat regions and the peaks, the LSTM model demonstrated superior generalization capabilities and provided the most accurate predictions for the CO2 test dataset of all the models.

It is worth noting that the Seasonal Autoregressive Integrated Moving Average (SARIMA) model produced consistent results between the two phases. Although it ranked third in the validation phase, it became the second-best performer in the test phase. Like the LSTM model, the SARIMA model achieved a respectable balance between fit and generalization.

6.2.2.1.2 Electricity Dataset

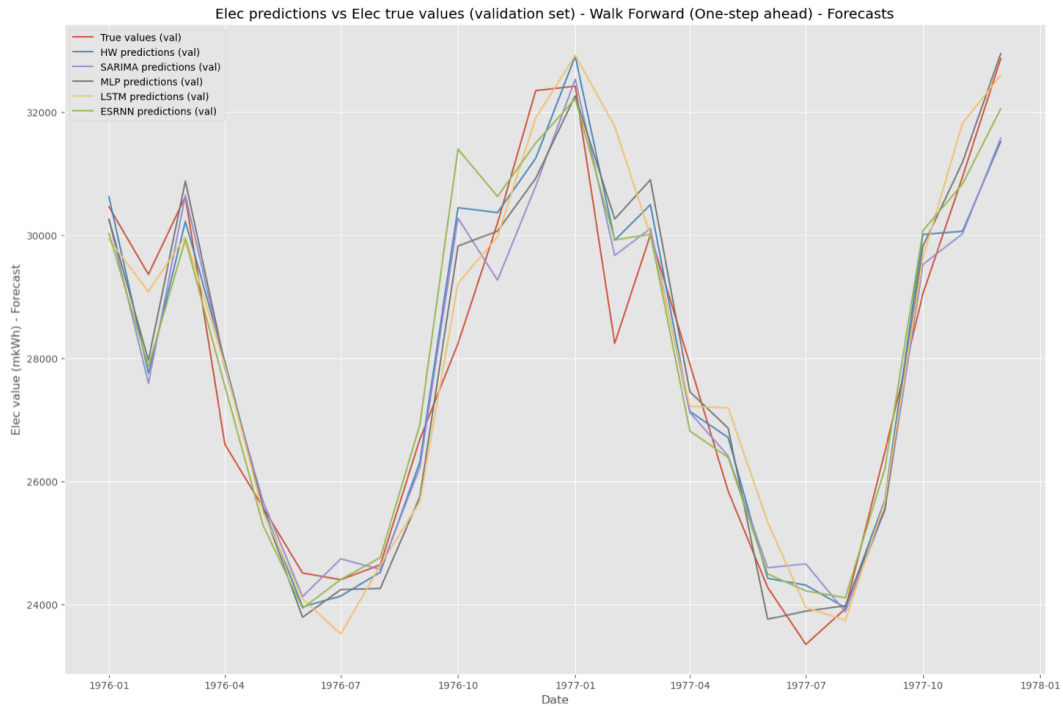


Figure 6.3: Electricity dataset - Validation set - Forecasts

During the validation phase, the neural network models encountered difficulties in accurately fitting two datasets: the Electricity dataset and the Iron Production dataset.

Of the models investigated, the Multilayer Perceptron (MLP) performed best. However, as mentioned above, there were concerns about overfitting. As the Multilayer Perceptron's predictions were too close to the true values, we suspect that the MLP model may have learned too much from the training data.

The Long Short-Term Memory (LSTM) model struggled to fit the data accurately and showed poor performance. Figure 6.3 shows that the LSTM produced the least accurate predictions of all the models examined. Unlike the other models, the LSTM often tended to either over- or under-estimate the true observations, resulting in less reliable predictions and ultimately poor prediction accuracy.

Finally, the Exponential Smoothing Recurrent Neural Network (ES-RNN) model provided reliable predictions that were in good agreement with those of the statistical models during validation.

Interestingly, the ES-RNN and Holt-Winters (HW) models showed similar trends. However, the ES-RNN occasionally outperformed the statistical model in certain regions. A notable advantage of the ES-RNN over the Holt-Winters model was its ability to minimize overestimation or underestimation, particularly at peaks, resulting in more accurate predictions.

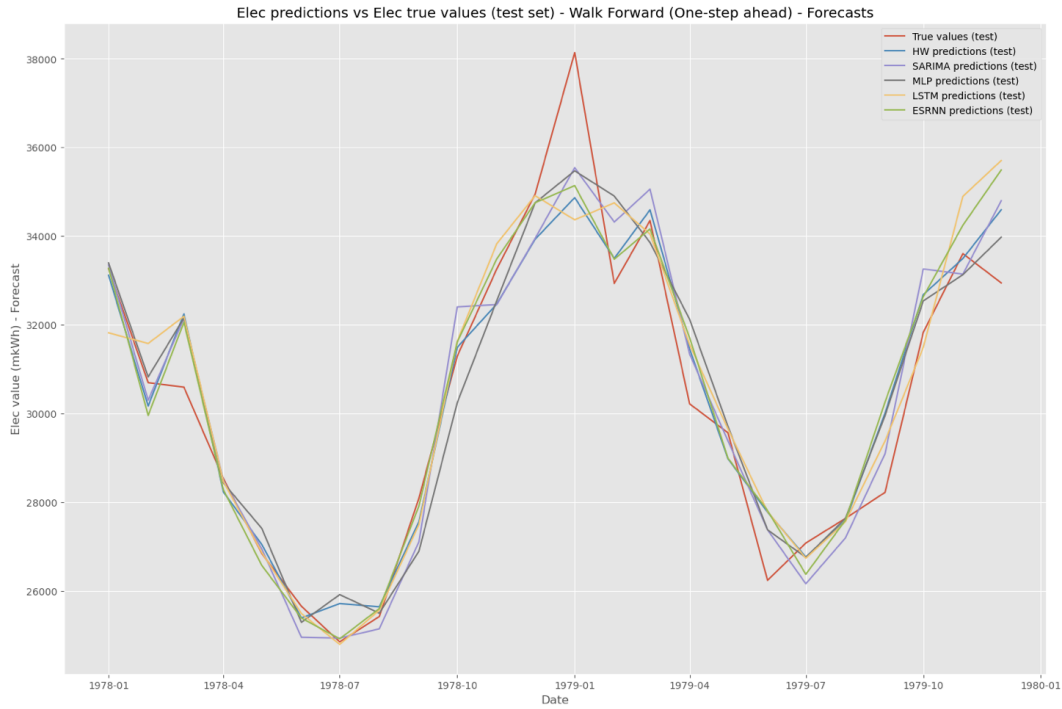


Figure 6.4: Electricity dataset - Test set - Forecasts

Surprisingly, the Seasonal Autoregressive Integrated Moving Average (SARIMA) model again achieved a good balance between fit and generalization. In fact, based on Figure 6.4, we can argue that this is the model that best fits the Electricity test data.

In contrast to its good performance during validation, the Exponential Smoothing Recurrent Neural Network (ES-RNN) model encountered difficulties in fitting certain regions during the test. As shown in Figure 6.4, both the SARIMA and ES-RNN models showed a similar overall trend, but the SARIMA model showed greater accuracy, particularly in the region of peaks or high points.

Further examination of Figure 6.4 shows that the ES-RNN model tended to underestimate the true values, in contrast to the SARIMA model which consistently provided more accurate predictions. This observation suggests that the SARIMA model may be better at capturing the nuances of the data, especially at critical points such as spikes in the given context.

Finally, it should be noted that although the Multilayer Perceptron (MLP) model overfitted the data, its results on the test set did not drop dramatically. It can be seen that the model generally performed as well as the statistical models, especially SARIMA model.

6.2.2.1.3 Iron Production Dataset

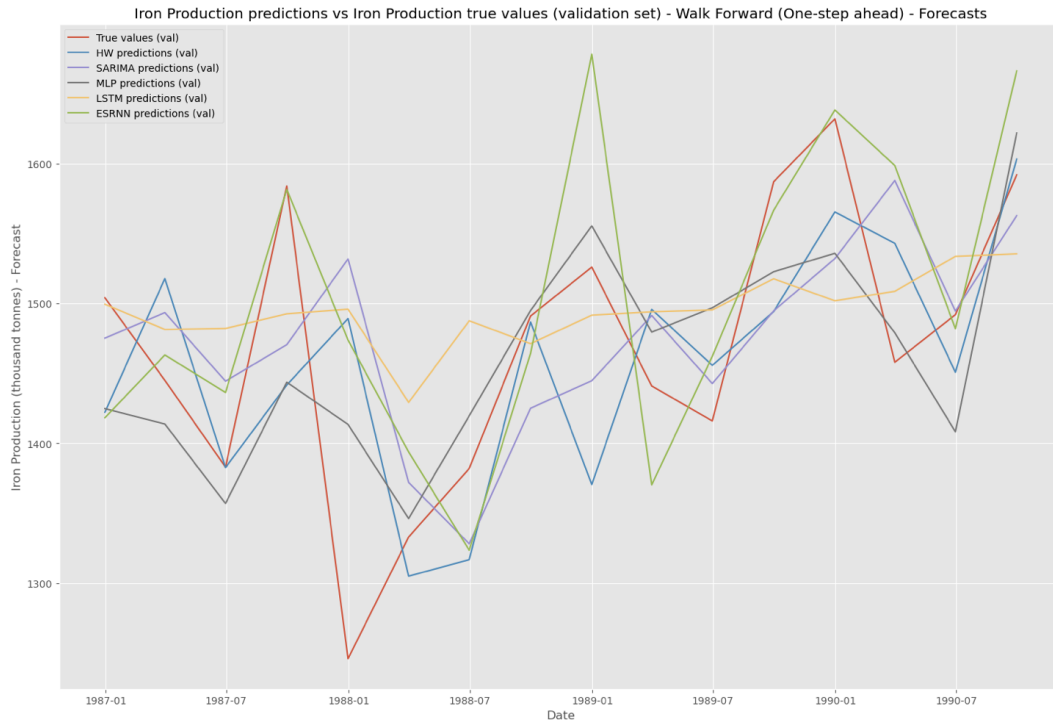


Figure 6.5: Iron Production dataset - Validation set - Forecasts

During validation, the Multilayer Perceptron (MLP) model emerged as the best performer, followed by the Exponential Smoothing Recurrent Neural Network (ES-RNN) and Long Short-Term Memory (LSTM) models in second and third place respectively.

The ES-RNN, Seasonal Autoregressive Integrated Moving Average (SARIMA), Holt-Winters (HW) and MLP models made considerable efforts to fit the series accurately. The MLP in particular, came close to the true values. However, we suspect that it overfitted again the data.

Unfortunately, the LSTM model struggled to fit the data accurately, resulting in poor performance. Figure 6.5 shows that the LSTM produced the least accurate predictions of all the models examined. Unlike its counterparts, the LSTM often showed a tendency to either overestimate or underestimate the actual observations.

The ES-RNN, which performed well during the validation phase and secured second place, showed reasonably good predictions. In particular, unlike the other models, the ES-RNN again excelled at fitting some of the peaks, demonstrating its strength in capturing the subtle nuances of the data in critical regions.

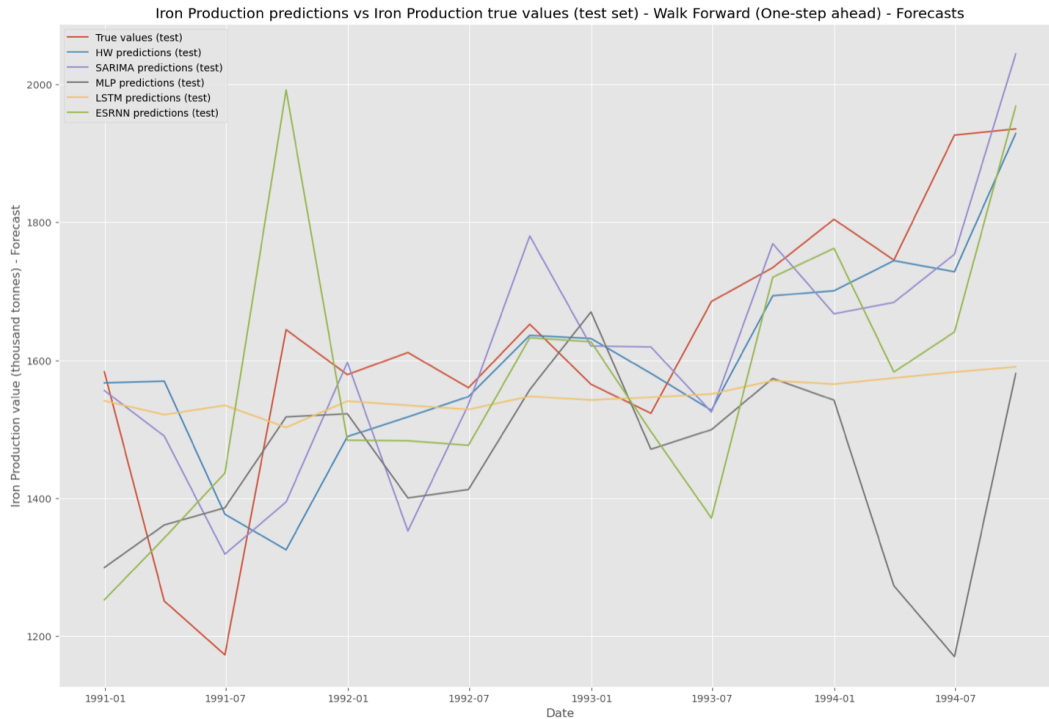


Figure 6.6: Iron Production dataset - Test set - Forecasts

Figure 6.6 confirms the hypothesis that the Multilayer Perceptron (MLP) model significantly overfitted the data, as evidenced by its significant performance degradation during the test phase.

Similarly, the Long Short-Term Memory (LSTM) model, which had already shown poor performance with an increasing trend as a predictor during validation, repeated inaccurate predictions during the test phase.

The Exponential Smoothing Recurrent Neural Network (ES-RNN), although initially one of the best performers during validation, also experienced a significant drop in performance during the test phase. Looking at Figure 6.6, we can see that the ES-RNN produced some of the largest errors. While it effectively captured moderate spikes, it struggled with fast shifts and variations.

Finally, it's important to note that the Seasonal Autoregressive Integrated Moving Average (SARIMA) model emerged as the best performer. Unlike the other models, SARIMA achieved a convincing balance between precise fitting and robust generalization. Although its performance was not equally strong in all segments, overall it was able to provide forecasts with a higher degree of accuracy.

6.2.2.1.4 Daily Min. Temperature Dataset

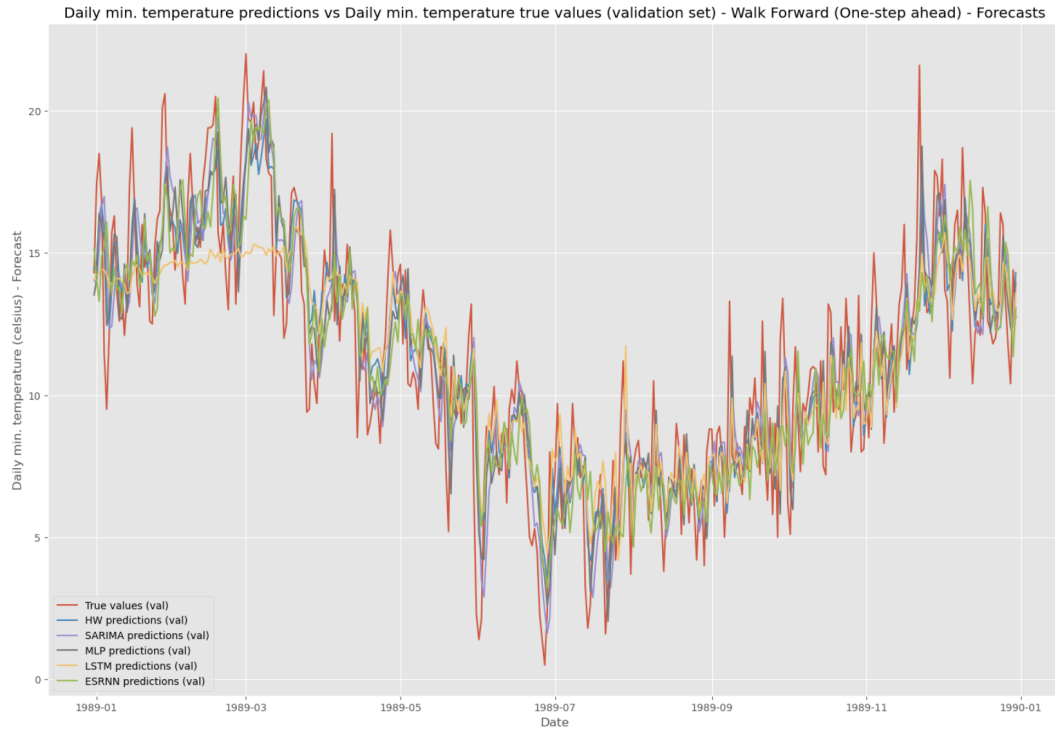


Figure 6.7: Daily min. temperature dataset - Validation set - Forecasts

During the validation phase, the models used for the Daily min. temperature dataset showed contrasting performances. All models except the Exponential Smoothing Recurrent Neural Network (ES-RNN) model produced similar results.

As discussed earlier (see point 6.1.3), the ES-RNN model exhibited a glass ceiling effect in its predictions during the validation. Further insight into this observation is provided by Figure 6.7, which illustrates the consistent tendency of the ES-RNN to either under- or over-estimate the observations, particularly in terms of variation from one observation to another.

In the early part of the validation set, the ES-RNN model showed accuracy in its predictions. However, as the validation progressed, it encountered significant challenges that resulted in significant underestimations or overestimations. This behavior contributed to the observed glass ceiling effect in its predictions and ultimately limited its overall performance during the validation phase.

As shown in Figure 6.7, it is important to note that the Long Short-Term Memory (LSTM) model initially struggled with the first part of the validation set. However, unlike the ES-RNN, as the amount of training data increased, its performance improved significantly, resulting in more accurate predictions in the second part of the validation set.

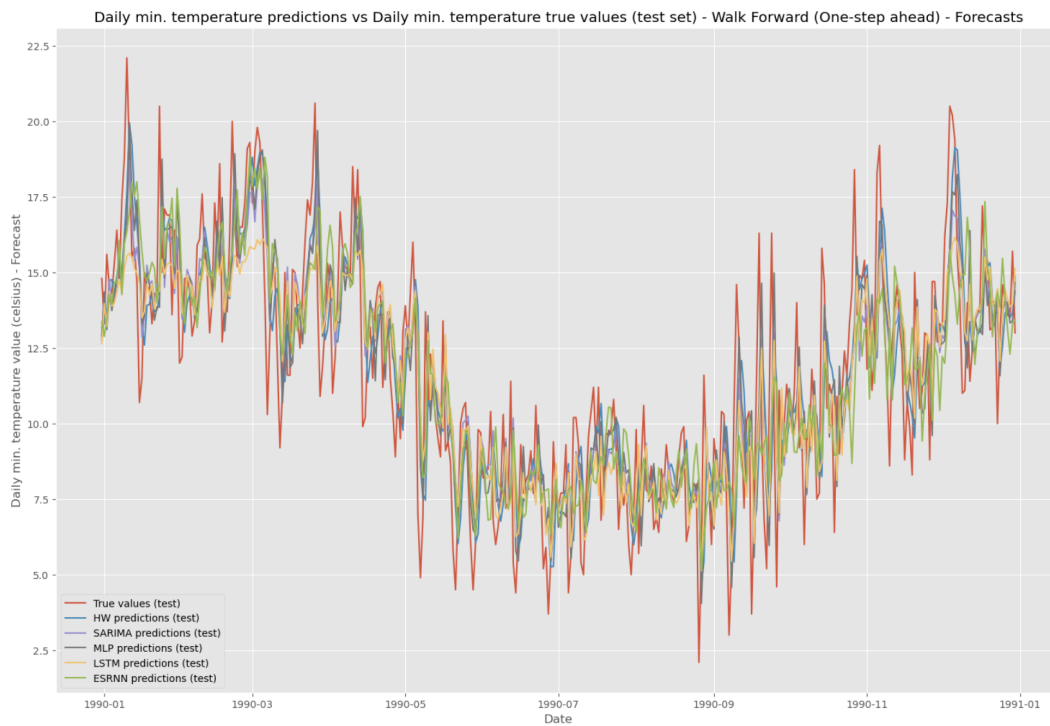


Figure 6.8: Daily min. temperature dataset - Test set - Forecasts

The results of the test phase remained consistent with those of the validation phase, showing similar trends and performance. The Exponential Smoothing Recurrent Neural Network (ES-RNN) continued to underestimate or overestimate the true values, similar to what was observed in the validation phase. In addition, the Long Short-Term Memory (LSTM) model performed similarly to the validation phase.

Once again, the Seasonal Autoregressive Integrated Moving Average (SARIMA) model showed superior performance during the test phase. Similar to the observations made during validation, the SARIMA model provided the most accurate forecasts without under- or over-estimating the observations, unlike the other models.

6.2.2.2 Test Set Results

Table 6.8: Test results (RMSE)

Models	CO2	ELECTRICITY	IRON PRODUCTION	DAILY MIN. TEMP
Holt-Winters	0.3922	1081.4712	147.1269	2.4967
SARIMA	0.3878	1045.1695	143.3327	2.2706
MLP	0.3917	1084.7214	284.3575	2.3725
LSTM	0.3665	1273.9194	195.7984	2.3309
ES-RNN	0.3931	1112.7403	186.0409	2.5005

Interestingly, although the three neural network models demonstrated their effectiveness during the validation phase, as shown in section 6.2.1, only two of these models, namely the Long Short-Term Memory (LSTM) and the Multilayer Perceptron (MLP), managed to maintain their strong performance in the subsequent test phase. However, it's important to note that this was observed on a limited number of datasets.

The LSTM model is the only neural network model to outperform all other competing models on a specific dataset during the test phase. This was achieved on the CO2 dataset, where it was around 6% more accurate than the second-best model, the Seasonal Autoregressive Integrated Moving Average (SARIMA) model.

Apart from the CO2 dataset, the LSTM model did not perform well during the test phase. Moreover, the test phase confirmed our earlier hypothesis that the LSTM model suffered from its complexity in fitting the datasets, as its results on the test sets are among the worst.

Although our assumption about the overfitting of the MLP model was confirmed by the testing phase, it is important to note that this assumption was not universally true for all datasets, as the results in Table 6.8 show.

We can see that in general, when the MLP score on a given validation set was much better than that of the other models, the result on the test set confirmed that the MLP was overfitting, e.g. the CO2 and Iron Production datasets³.

On the contrary, if its result on the validation set was similar or close to that of the other models, its result on the test set was also close to that of the other models. This can be seen with its result on the Daily min. temperature dataset.

Regrettably, the Exponential Smoothing Recurrent Neural Network (ES-RNN) model fell short of the initial expectations set at the beginning of this thesis. In fact, at the end of the study, the ES-RNN had one of the lowest levels of performance among the models evaluated for the selected datasets.

³Note that the Multilayer Perceptron (MLP) model overfitted the Electricity dataset during the validation phase, but this did not result in a significant downgrade during the test phase, as its test score ranked third.

Despite its initially promising attributes, which positioned the ES-RNN model as a potential contender in the given study context, it ultimately showed a propensity to overfit the datasets.

Indeed, the ES-RNN model consistently demonstrated its capabilities by producing some of the best results during the validation phase. It emerged as the top performer for 3 of the 4 datasets and even took the lead for the CO2 dataset.

However, the results of the model were significantly worse during the testing phase, meaning that the model actually overfitted the data during the validation phase. As a result, the ES-RNN went from being one of the best-performing models to one of the worst-performing models between the two phases.

6.3 Conclusion

A recurring observation throughout our study was the tendency of neural networks to either overfit or underfit the datasets. Despite their exceptional performance during the validation phase, their accuracy dropped significantly during the testing phase, confirming our finding that they have a tendency to overfit or underfit the data. In particular, we also found that the quality of the results deteriorated as the complexity of the model increased.

Unfortunately, the model we expected the most from, the Exponential Smoothing Recurrent Neural Network (ES-RNN), did not live up to our expectations. It went from being one of the best-performing models during the validation phase to one of the worst-performing models after the testing phase.

In general, because of its complexity, but also because of the way it adapts to the series thanks to its two-stage phases, the ES-RNN overfitted some of the datasets the most and then gave some of the worst results in the test phase.

Chapter 7

Study Conclusion

7.1 Study Conclusion

Surprisingly, our benchmarks, the statistical models, ended up being the best-performing models in the study. Although the Holt-Winters (HW) model delivered promising results in the study, we refer here specifically to the Seasonal Autoregressive Integrated Moving Average (SARIMA) model. The SARIMA model outperformed all other models for three of the four selected time series and came second for the fourth. Overall, the model produced the best and most consistent results in the study for both the validation and test sets.

Simple, stable, and easy to train, tune and evaluate, the SARIMA model was one of the best performers in the validation phase and the most consistent model in the testing phase. Linear in its modelling, its performance in this study once again confirmed the hypothesis put forward during the forecasting competitions that simpler methods are often sufficient to tackle univariate time series forecasting problems.

Additionally, the SARIMA and Holt-Winters models exhibited robust performance with minimal tuning, often surpassing the neural networks that demanded extensive tuning to attain similar levels of accuracy.

In contrast to the neural network models, and as noted above, the statistical models produced some of the best results in the validation phase and were able to replicate this level of accuracy in the test phase. We, therefore, concluded that the statistical models were more appropriate to use in the given study context.

While the neural network models were able to fit the validation sets well, they were unable to replicate their promising performance on the test sets, unlike the statistical models. In particular, the neural network models tended to overfit the datasets.

7.2 Research Question

Unfortunately, at the end of the study, only one neural network model was able to compete with the statistical models during the test phase, the Long Short-Term Memory (LSTM) model. However, as mentioned above, the model was only able to compete with the statistical models on the CO2 dataset. For the other datasets, the neural network model was unable to match the accuracy of the benchmarks.

Although we had high expectations for the Exponential Smoothing Recurrent Neural Network (ES-RNN) model, by the end of the study, it had not lived up to our expectations. While the model was one of the best during the validation phase, its performance dropped dramatically when tested on unseen data.

In fact, during the study, we found that the ES-RNN model was able to fit a given function well, but not necessarily make that fit general enough to be applied to unseen data. Despite being stable and easy to train, tune and evaluate, the ES-RNN ended up overfitting the series more than the other models, as its results on the test sets were among the worst for most of the selected time series.

Also, since we noticed during the study that the more complex the model, the worse the observed results, we suggest that the ES-RNN model, due to its two-stage implementation, fitted the time series so well that the model was unable to generalize sufficiently during the test phase. The ES-RNN model, whose results remained stable throughout the sensitivity analyses, was already overfitting the series.

It is important to note that the walk-forward validation procedure we used generally caused the models to overfit the datasets, as we used all available training data to predict the next time step (R. Hyndman & Athanasopoulos, 2018). However, unlike the neural networks, the statistical models were able to find a compromise so that they did not overfit the series too much.

In conclusion, although the ES-RNN did not perform well in the given study context, this does not necessarily mean that the model cannot be used in practice or in other contexts. In fact, as most of the series were simple, the statistical models were simply sufficient to fit them well. However, in a different context, the conclusions might have been different.

Chapter 8

Conclusion & Discussion

8.1 Summary

Using the Seasonal Autoregressive Integrated Moving Average (SARIMA) and Holt-Winters (HW) models as benchmarks, this thesis conducted a comprehensive comparative analysis of various neural network models to assess their forecasting accuracy. To lay the foundation for our research objectives, the thesis began with a thorough review of the literature on forecasting and forecasting competitions.

Next, we looked at the theoretical underpinnings of the neural network models used in the study, which provided important insights into how they work. Having established a solid theoretical foundation, we then carefully defined the study parameters and outlined the methodology used to ensure a rigorous and robust evaluation.

Before testing the models, we performed sensitivity analyses on specific hyperparameters to understand how the neural network models responded to variations in these parameters. This process helped us gain valuable insights into how the neural networks responded to changes and to find optimal models.

Contrary to our initial expectations, the results of the study raised doubts about the effectiveness of neural network models. We observed instances of both overfitting and underfitting problems. The statistical models, on the other hand, performed satisfactorily and proved adequate for the time series selected.

Despite our initial high hopes for the Exponential Smoothing Recurrent Neural Network (ES-RNN), by the end of the study it had failed to live up to our expectations. Although it showed a good ability to fit the data points provided, its ability to generalize and predict unseen data was limited. The same applied to the other neural network models.

Overall, the study highlighted the strengths and limitations of neural network models in the given research context. It also highlighted the importance of considering different factors when selecting the appropriate model for a given time series and research scenario.

8.2 Future Works

In the study, we concluded that the statistical models were good enough to fit the given time series well, but what if the series were different? One way to extend the study is to add more series and see if it is consistently the case that statistical models outperform neural networks. In addition, as we have specifically used monthly, quarterly and daily seasonal time series in the study, the use of a wider variety of time series periodicity may also be of interest to consolidate or dissipate the conclusions we have drawn.

Furthermore, since we have limited the study to (1) one-step-ahead forecasts and (2) univariate time series, two extensions are possible that may strengthen or weaken our conclusions. First, one way to extend the study is to forecast multiple steps instead of one step. Second, since neural networks are widely used in practice for multivariate time series, investigating their accuracy on multivariate time series may also be a direction to investigate. In both directions, it may be of interest to see if our conclusions still hold.

Finally, throughout this thesis, the main focus has been on investigating the Exponential Smoothing Recurrent Neural Network (ES-RNN), a specific hybrid model. However, it is important to note that there are many other hybrid models in the literature that deserve investigation. Prominent examples include the Autoregressive Integrated Moving Average Artificial Neural Network (ARIMA-ANN) (Zhang, 2003), the Autoregressive Integrated Moving Average Support Vector Machine (ARIMA-SVM) (Shi et al., 2012), and the Long Short-Term Memory Prophet (LSTM-Prophet) (Arslan, 2022) models.

Expanding the study to include several hybrid models could provide valuable information about the specific characteristics and capabilities of each. In addition, such an investigation would allow us to identify which hybrid models stand out in specific contextual circumstances. Examining the performance of different hybrid models can deepen our understanding of their strengths and weaknesses, ultimately paving the way for more informed and context-aware forecasting.

Bibliography

- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. <https://doi.org/10.1037/h0042519>
- Werbos, P. J. (1990). Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10), 1550–1560. <https://doi.org/10.1109/5.58337>
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9, 1735–80. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Schuster, M., & Paliwal, K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45, 2673–2681. <https://doi.org/10.1109/78.650093>
- Hu, M. Y., Zhang, G. P., Jiang, C. X., & Patuwo, B. E. (1999). A cross-validation analysis of neural network out-of-sample performance in exchange rate forecasting. *Decision Sciences*, 30, 197–216.
- Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., Newton, J., Parzen, E., & Winkler, R. (2000). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2), 111–153.
- Makridakis, S., & Hibon, M. (2000). The m3-competition: Results, conclusions and implications. *International Journal of Forecasting*, 16(4), 451–476. [https://doi.org/https://doi.org/10.1016/S0169-2070\(00\)00057-1](https://doi.org/https://doi.org/10.1016/S0169-2070(00)00057-1)
- Tashman, L. (2000). Out-of-sample tests of forecasting accuracy: An analysis and review. *International Journal of Forecasting*, 16, 437–450. [https://doi.org/10.1016/S0169-2070\(00\)00065-0](https://doi.org/10.1016/S0169-2070(00)00065-0)
- Zhang, G. (2003). Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50, 159–175. [https://doi.org/https://doi.org/10.1016/S0925-2312\(01\)00702-0](https://doi.org/https://doi.org/10.1016/S0925-2312(01)00702-0)
- De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. *International Journal of Forecasting*, 22(3), 443–473. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2006.01.001>
- Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4), 679–688. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2006.03.001>
- Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. *Journal of Machine Learning Research - Proceedings Track*, 9, 249–256.

- Seabold, S., & Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with python. *9th Python in Science Conference*.
- Crone, S. F., Hibon, M., & Nikolopoulos, K. (2011). Advances in forecasting with neural networks? empirical evidence from the nn3 competition on time series prediction [Special Section 1: Forecasting with Artificial Neural Networks and Computational Intelligence Special Section 2: Tourism Forecasting]. *International Journal of Forecasting*, 27(3), 635–660. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2011.04.001>
- Bergmeir, C., & Benítez, J. (2012). On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191, 192–213. <https://doi.org/10.1016/j.ins.2011.12.028>
- Pascanu, R., Mikolov, T., & Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063. <http://arxiv.org/abs/1211.5063>
- Shi, J., Guo, J., & Zheng, S. (2012). Evaluation of hybrid forecasting approaches for wind speed and power generation time series. *Renewable and Sustainable Energy Reviews*, 16(5), 3471–3480. <https://doi.org/https://doi.org/10.1016/j.rser.2012.02.044>
- Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (pp. 1310–1318). PMLR. <https://proceedings.mlr.press/v28/pascanu13.html>
- Cho, K., Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. <https://doi.org/10.3115/v1/D14-1179>
- Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: Forecasting and control*. John Wiley & Sons.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521, 436–44. <https://doi.org/10.1038/nature14539>
- Olah, C. (2015). Understanding LSTM networks [Accessed: 2023-04-22]. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Yao, K., Cohn, T., Vylomova, K., Duh, K., & Dyer, C. (2015). Depth-gated recurrent neural networks.
- Brockwell, P. J., & Davis, R. A. (2016). *Introduction to time series and forecasting*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-29854-2>
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. <http://www.deeplearningbook.org>
- Brownlee, J. (2017). How to frame time series forecasting as a supervised learning problem. *Machine Learning Mastery*. <https://machinelearningmastery.com/time-series-forecasting-supervised-learning/>
- Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M., & Huang, T. S. (2017). Dilated recurrent neural networks. *arXiv*. <https://arxiv.org/abs/1710.02224>
- Chollet, F. (2017). *Deep learning with python*. Manning Publications.
- Géron, A. (2017). *Hands-on machine learning with scikit-learn and tensorflow : Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

- Kingma, D. P., & Ba, J. (2017). Adam: A method for stochastic optimization.
- Bergmeir, C., Hyndman, R., & Koo, B. (2018). A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics Data Analysis*, 120(100), 70–83. <https://EconPapers.repec.org/RePEc:eee:csdana:v:120:y:2018:i:c:p:70-83>
- Hyndman, R., & Athanasopoulos, G. (2018). *Forecasting: Principles and practice* (2nd). OTexts.
- Majid, R., & Mir, S. (2018). Advances in statistical forecasting methods: An overview. *Economic Affairs*, 63, 815–831. <https://doi.org/10.30954/0424-2513.4.2018.5>
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4), 802–808.
- Michelucci, U. (2018). *Applied deep learning: A case-based approach to understanding deep neural networks*. <https://doi.org/10.1007/978-1-4842-3790-8>
- Denuit, M., Hainaut, D., & Trufin, J. (2019). *Effective statistical learning methods for actuaries iii: Neural networks and extensions*. Springer Actuarial. <https://doi.org/10.1007/978-3-030-25827-6>
- Redd, A., Khin, K., & Marini, A. (2019). Fast es-rnn: A gpu implementation of the es-rnn algorithm. *ArXiv, abs/1907.03329*.
- Gilliland, M. (2020). The value added by machine learning approaches in forecasting. *International Journal of Forecasting*, 36(1), 161–166. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.04.016>
- Makridakis, S., & Petropoulos, F. (2020). The m4 competition: Conclusions [M4 Competition]. *International Journal of Forecasting*, 36(1), 224–227. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.05.006>
- Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The m4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1), 54–74. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.04.014>
- Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting [M4 Competition]. *International Journal of Forecasting*, 36(1), 75–85. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2019.03.017>
- Spiliotis, E., Kouloumos, A., Assimakopoulos, V., & Makridakis, S. (2020). Are forecasting competitions data representative of the reality? *International Journal of Forecasting*, 36(1), 37–53. <https://doi.org/https://doi.org/10.1016/j.ijforecast.2018.12.007>
- Koutsandreas, D., Spiliotis, E., Petropoulos, F., & Assimakopoulos, V. (2021). On the selection of forecasting accuracy measures. *Journal of the Operational Research Society*, 73, 1–18. <https://doi.org/10.1080/01605682.2021.1892464>
- Arslan, S. (2022). A hybrid forecasting model using lstm and prophet for energy consumption with decomposition of time series data. *PeerJ Computer Science*, 8, e1001. <https://doi.org/10.7717/peerj-cs.1001>

Appendix A

SARIMA coefficients

This section presents the coefficients of the SARIMA models. It should be noted that these coefficients have been obtained by training the models on both the training and validation sets.

Furthermore, given the predictive nature of this work, it is important to stress that the SARIMA models identified were retained without attempting to remove coefficients associated with significant P-values.

A.1 SARIMA - Coefficients - CO2 Dataset

SARIMAX Results							
Dep. Variable:		y			No. Observations:		168
Model:		SARIMAX(1, 3, 3)x(1, 1, [], 12)			Log Likelihood		-53.468
Date:		Mon, 14 Aug 2023			AIC		118.937
Time:		06:05:49			BIC		136.587
Sample:		01-01-1965			HQIC		126.109
		- 12-01-1978					
Covariance Type:		opg					
	coef	std err	z	P> z	[0.025	0.975]	
ar.L1	-0.6856	0.098	-7.027	0.000	-0.877	-0.494	
ma.L1	-1.5072	2.042	-0.738	0.460	-5.509	2.494	
ma.L2	0.0145	1.067	0.014	0.989	-2.077	2.106	
ma.L3	0.4937	1.021	0.483	0.629	-1.508	2.495	
ar.S.L12	-0.5377	0.075	-7.152	0.000	-0.685	-0.390	
sigma2	0.1128	0.231	0.488	0.625	-0.340	0.566	
Ljung-Box (L1) (Q):		0.15	Jarque-Bera (JB):		0.95		
Prob(Q):		0.70	Prob(JB):		0.62		
Heteroskedasticity (H):		0.58	Skew:		-0.07		
Prob(H) (two-sided):		0.07	Kurtosis:		2.62		
Warnings:							
[1] Covariance matrix calculated using the outer product of gradients							

Figure A.1: SARIMA - Coefficients - Test Model - CO2 Dataset

A.2 SARIMA - Coefficients - Electricity Dataset

SARIMAX Results

Dep. Variable:	y	No. Observations:	276
Model:	SARIMAX(3, 0, 3)x(2, 1, 3, 12)	Log Likelihood	-1757.314
Date:	Mon, 14 Aug 2023	AIC	3538.629
Time:	06:11:57	BIC	3579.569
Sample:	01-01-1955	HQIC	3555.154
	- 12-01-1977		
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.5975	0.729	0.819	0.413	-0.832	2.027
ar.L2	0.8413	0.090	9.309	0.000	0.664	1.018
ar.L3	-0.4562	0.632	-0.722	0.471	-1.695	0.783
ma.L1	0.0069	0.873	0.008	0.994	-1.704	1.718
ma.L2	-0.7669	0.504	-1.523	0.128	-1.754	0.220
ma.L3	0.2250	0.375	0.600	0.549	-0.510	0.960
ar.S.L12	0.3067	0.515	0.595	0.552	-0.704	1.317
ar.S.L24	0.5567	0.454	1.226	0.220	-0.334	1.447
ma.S.L12	-1.1582	0.536	-2.159	0.031	-2.210	-0.107
ma.S.L24	-0.2230	0.899	-0.248	0.804	-1.985	1.539
ma.S.L36	0.5776	0.453	1.276	0.202	-0.310	1.465
sigma2	3.149e+05	1.15e+05	2.745	0.006	9e+04	5.4e+05

Ljung-Box (L1) (Q):	0.19	Jarque-Bera (JB):	42.25
Prob(Q):	0.67	Prob(JB):	0.00
Heteroskedasticity (H):	4.90	Skew:	-0.44
Prob(H) (two-sided):	0.00	Kurtosis:	4.94

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Figure A.2: SARIMA - Coefficients - Test Model - Electricity Dataset

A.3 SARIMA - Coefficients - Iron Production Dataset

SARIMAX Results

Dep. Variable:	Iron production (thousand tonnes)	No. Observations:	139			
Model:	SARIMAX(2, 0, 2)x(3, 1, [1, 2, 3], 4)	Log Likelihood	-751.304			
Date:	Mon, 14 Aug 2023	AIC	1524.608			
Time:	15:56:01	BIC	1555.270			
Sample:	03-31-1956	HQIC	1537.060			
	- 09-30-1990					
Covariance Type:	opg					
	coef	std err	z	P> z 	[0.025	0.975]
ar.L1	-0.0202	0.101	-0.200	0.841	-0.218	0.177
ar.L2	0.8306	0.096	8.657	0.000	0.643	1.019
ma.L1	0.7793	0.143	5.450	0.000	0.499	1.060
ma.L2	-0.2330	0.095	-2.458	0.014	-0.419	-0.047
ar.S.L4	0.1283	2.588	0.050	0.960	-4.945	5.202
ar.S.L8	-0.1381	1.735	-0.080	0.937	-3.539	3.262
ar.S.L12	-0.2135	1.098	-0.194	0.846	-2.366	1.939
ma.S.L4	-0.7390	2.601	-0.284	0.776	-5.838	4.360
ma.S.L8	-0.0745	3.303	-0.023	0.982	-6.548	6.399
ma.S.L12	0.2266	1.420	0.160	0.873	-2.557	3.010
sigma2	1.49e+04	2840.614	5.244	0.000	9328.365	2.05e+04
Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	42.17			
Prob(Q):	0.89	Prob(JB):	0.00			
Heteroskedasticity (H):	2.40	Skew:	-0.82			
Prob(H) (two-sided):	0.01	Kurtosis:	5.40			

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Figure A.3: SARIMA - Coefficients - Test Model - Iron Production Dataset

A.4 SARIMA - Coefficients - Daily min. temperature Dataset

SARIMAX Results

Dep. Variable:	Daily min temp. (Celsius)	No. Observations:	1460
Model:	SARIMAX(1, 1, 2)x(1, 0, [], 7)	Log Likelihood	-3297.432
Date:	Tue, 15 Aug 2023	AIC	6604.865
Time:	10:57:13	BIC	6631.265
Sample:	0	HQIC	6614.716
	- 1460		
Covariance Type:	opg		

	coef	std err	z	P> z	[0.025	0.975]
ar.L1	0.3003	0.065	4.645	0.000	0.174	0.427
ma.L1	-0.7206	0.066	-10.887	0.000	-0.850	-0.591
ma.L2	-0.1792	0.053	-3.379	0.001	-0.283	-0.075
ar.S.L7	0.0019	0.024	0.079	0.937	-0.046	0.050
sigma2	5.5092	0.192	28.768	0.000	5.134	5.885

Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	9.90
Prob(Q):	0.89	Prob(JB):	0.01
Heteroskedasticity (H):	1.00	Skew:	0.13
Prob(H) (two-sided):	0.97	Kurtosis:	3.31

Warnings:

[1] Covariance matrix calculated using the outer product of gradients (complex-step).

Figure A.4: SARIMA - Coefficients - Test Model - Daily min. temperature Dataset

Appendix B

Sensitivity Analysis

Please find below the results of our sensitivity analyses. The tables display both the initial configurations, which were obtained through grid search, and the final configurations, which were updated following the sensitivity analyses. It is essential to note that each table reflects changes in only one hyperparameter, while the remaining configuration remains the same as the initial setup.

B.1 Multilayer Perceptron - CO2 Dataset

Hyperparameter	Initial Value	Final Value
Inputs	15	15
Nodes	128	32
Hidden Layers	3	3
Epochs	30	30
Batch Size	16	16
Trend Diff	1	1
Seasonal Diff	1	1
Optimiser	SGD	SGD
Learning Rate	0.001	0.001
Standardized	True	True
Log Transformation	False	False
RMSE - Validation Score	0.2644	0.2457

Table B.1: Initial & Final Configurations - Multilayer Perceptron - Hyperparameters

Epochs	RMSE - Validation set
15	0.2559
20	0.2568
25	0.2579
30	0.2644
35	0.2595
40	0.2600
45	0.2603

Table B.2: Multilayer Perceptron
- RMSE - Validation - Epochs

Inputs	RMSE - Validation set
12	0.2804
13	0.2647
14	0.2568
15	0.2644
16	0.2581
17	0.2469
18	0.2711

Table B.3: Multilayer Perceptron
- RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
2	0.2635
4	0.2627
8	0.2599
16	0.2644
32	0.2567
64	0.2665
128	0.2936

Table B.4: Multilayer Perceptron
- RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
16	0.3426
32	0.2457
64	0.2579
128	0.2644
256	0.2558
512	0.2560
1024	0.2578

Table B.5: Multilayer Perceptron
- RMSE - Validation - Nodes

B.2 Multilayer Perceptron - Electricity Dataset

Hyperparameter	Initial Value	Final Value
Inputs	30	27
Nodes	64	64
Hidden Layers	1	1
Epochs	370	370
Batch Size	8	8
Trend Diff	1	1
Seasonal Diff	0	0
Optimiser	Adam	Adam
Learning Rate	0.0001	0.0001
Standardized	True	True
Log Transformation	False	False
RMSE - Validation Score	932.4015	868.3411

Table B.6: Initial & Final Configurations - Multilayer Perceptron - Hyperparameters

Epochs	RMSE - Validation set
340	933.8287
350	932.9768
360	932.5091
370	932.4015
380	932.6278
390	933.1612
400	933.9746

Table B.7: Multilayer Perceptron
- RMSE - Validation - Epochs

Inputs	RMSE - Validation set
27	868.3411
28	954.3185
29	908.0222
30	932.4015
31	1017.6119
32	944.8071
33	991.0806

Table B.8: Multilayer Perceptron
- RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
1	1035.1145
2	1010.2946
4	974.9219
8	932.4015
16	949.7525
32	1015.9378
64	1047.8765

Table B.9: Multilayer Perceptron
- RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
4	1055.4618
8	1258.4736
32	989.3654
64	932.4015
128	970.1521
256	917.8842
512	980.2496

Table B.10: Multilayer Perceptron
- RMSE - Validation - Nodes

B.3 Multilayer Perceptron - Iron Production Dataset

Hyperparameter	Initial Value	Final Value
Inputs	16	12
Nodes	512	512
Hidden Layers	1	1
Epochs	120	120
Batch Size	16	16
Trend Diff	0	0
Seasonal Diff	0	0
Optimiser	SGD	SGD
Learning Rate	0.0001	0.0001
Standardized	False	False
Log Transformation	True	True
RMSE - Validation Score	103.4557	74.1208

Table B.11: Initial & Final Configurations - Multilayer Perceptron - Hyperparameters

Epochs	RMSE - Validation set
105	103.4912
110	103.4822
115	103.4732
120	103.4557
125	103.4375
130	103.4286
135	103.4196

Table B.12: Multilayer Perceptron
- RMSE - Validation - Epochs

Inputs	RMSE - Validation set
4	139.5623
8	150.3855
12	74.1208
16	103.4557
20	121.4205
24	197.2345
28	239.6684

Table B.13: Multilayer Perceptron
- RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
2	150.8958
4	136.8685
8	115.2154
16	103.4557
32	112.4326
64	102.1612
128	102.4447

Table B.14: Multilayer Perceptron
- RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
8	1383.7601
16	779.7481
32	346.3836
64	133.2018
128	102.5854
256	99.2439
512	103.4557

Table B.15: Multilayer Perceptron
- RMSE - Validation - Nodes

B.4 Multilayer Perceptron - Daily min. temperature Dataset

Hyperparameter	Initial Value	Final Value
Inputs	30	7
Nodes	64	64
Hidden Layers	1	1
Epochs	20	20
Batch Size	256	256
Trend Diff	1	1
Seasonal Diff	0	0
Optimiser	SGD	SGD
Learning Rate	0.01	0.01
Standardized	False	False
Log Transformation	False	False
RMSE - Validation Score	2.2465	2.2240

Table B.16: Initial & Final Configurations - Multilayer Perceptron - Hyperparameters

Epochs	RMSE - Validation set
15	2.2695
20	2.2465
25	2.2495
30	2.2769
35	2.2533
40	2.2814
45	2.3238

Table B.17: Multilayer Perceptron
- RMSE - Validation - Epochs

Inputs	RMSE - Validation set
7	2.2240
14	2.2445
21	2.2480
30	2.2465
35	2.2771
42	2.3047
49	2.3031

Table B.18: Multilayer Perceptron
- RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
16	3.1089
32	2.6795
128	2.3549
256	2.2465
512	2.2949
1024	2.3114
2048	2.3107

Table B.19: Multilayer Perceptron
- RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
16	2.2703
32	2.3494
64	2.2465
128	2.2351
256	2.2293
512	2.2312

Table B.20: Multilayer Perceptron
- RMSE - Validation - Nodes

B.5 Long Short-Term Memory - CO2 Dataset

Hyperparameter	Initial Value	Final Value
Inputs	24	22
Nodes	256	256
Hidden Layers	1	1
Epochs	90	90
Batch Size	16	16
Trend Diff	0	0
Seasonal Diff	1	1
Optimiser	SGD	SGD
Learning Rate	0.01	0.01
Standardized	True	True
Log Transformation	False	False
RMSE - Validation Score	0.2742	0.2645

Table B.21: Initial & Final Configurations - Long Short-Term Memory - Hyperparameters

Epochs	RMSE - Validation set
75	0.2732
80	0.2745
85	0.2729
90	0.2742
95	0.2732
100	0.2735
105	0.2733

Table B.22: Long Short-Term
Memory - RMSE - Validation -
Epochs

Inputs	RMSE - Validation set
21	0.2652
22	0.2645
23	0.2760
24	0.2742
25	0.2684
26	0.2719
27	0.2720

Table B.23: Long Short-Term
Memory - RMSE - Validation -
Inputs

Batch Size	RMSE - Validation set
2	0.2939
4	0.2776
8	0.2725
16	0.2742
32	0.2889
64	0.2924
128	0.3143

Table B.24: Long Short-Term Memory - RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
16	0.2788
32	0.2807
64	0.2735
128	0.2741
256	0.2742
512	0.2739
1024	0.2754

Table B.25: Long Short-Term Memory - RMSE - Validation - Nodes

B.6 Long Short-Term Memory - Electricity Dataset

Hyperparameter	Initial Value	Final Value
Inputs	12	12
Nodes	64	64
Hidden Layers	1	1
Epochs	270	270
Batch Size	32	32
Trend Diff	1	1
Seasonal Diff	0	0
Optimiser	Adam	Adam
Learning Rate	0.0001	0.0001
Standardized	True	True
Log Transformation	True	True
RMSE - Validation Score	999.0565	999.0565

Table B.26: Initial & Final Configurations - Long Short-Term Memory - Hyperparameters

Epochs	RMSE - Validation set
270	999.0565
280	1028.8421
290	1015.1513
300	1085.8597
310	1108.8937
320	1050.8933
330	1043.0034

Table B.27: Long Short-Term Memory - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
6	1287.5794
12	999.0565
18	1428.1622
24	1149.4644
30	1252.3380
36	1240.2518
42	1253.9936

Table B.28: Long Short-Term Memory - RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
4	1078.5115
8	1057.2294
16	1063.8618
32	999.0565
64	1098.5040
128	1224.8146
256	1277.9974

Table B.29: Long Short-Term Memory - RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
8	1294.1177
16	1207.2665
32	1073.6878
64	999.0565
128	1115.2638
256	1154.6604
512	1169.5186

Table B.30: Long Short-Term Memory - RMSE - Validation - Nodes

B.7 Long Short-Term Memory - Iron Production Dataset

Hyperparameter	Initial Value	Final Value
Inputs	24	24
Nodes	16	16
Hidden Layers	3	3
Epochs	35	35
Batch Size	1	1
Trend Diff	0	0
Seasonal Diff	0	0
Optimiser	Adam	Adam
Learning Rate	0.00001	0.00001
Standardized	True	True
Log Transformation	True	True
RMSE - Validation Score	94.3165	94.3165

Table B.31: Initial & Final Configurations - Long Short-Term Memory - Hyperparameters

Epochs	RMSE - Validation set
20	151.8296
25	110.0261
30	104.7661
35	94.3165
40	103.0477
45	107.6646
50	117.2994

Table B.32: Long Short-Term Memory - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
12	125.4239
16	107.0271
20	97.4845
24	94.3165
28	114.7067
32	110.3697
36	108.9608

Table B.33: Long Short-Term Memory - RMSE - Validation - Inputs

Batch Size	RMSE - Validation set
1	94.3165
2	168.7524
4	226.8382
8	252.7733
16	269.0261
32	273.4918
64	279.0173

Table B.34: Long Short-Term Memory - RMSE - Validation - Batch Size

Nodes	RMSE - Validation set
8	99.2034
16	94.3165
32	104.6398
64	104.4896
128	114.1016
256	117.5681
512	118.2926

Table B.35: Long Short-Term Memory - RMSE - Validation - Nodes

B.8 Exponential Smoothing Recurrent Neural Network - CO2 Dataset

Hyperparameter	Initial Value	Final Value
Inputs	48	48
Nodes	48	48
Cell Type	RNN	RNN
Dilations	[1, 4, 8, 16]	[1, 4, 8, 16]
Epochs	95	90
Learning Rate	0.001	0.001
Percentile	50	50
Normalization Layer	True	True
Log Transformation	False	False
RMSE - Validation Score	0.2480	0.2342

Table B.36: Initial & Final Configurations - Exponential Smoothing Recurrent - Hyperparameters

Epochs	RMSE - Validation set
60	0.3524
65	0.3108
70	0.3157
75	0.3178
85	0.2534
90	0.2342
95	0.2480

Table B.37: Exponential Smoothing Recurrent - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
30	1.1568
36	0.3215
42	0.4314
48	0.2480
54	0.3223
60	0.2921
66	0.2856

Table B.38: Exponential Smoothing Recurrent - RMSE - Validation - Inputs

Nodes	RMSE - Validation set
16	0.8299
32	0.7901
48	0.2480
64	0.3309
128	0.3309
256	0.3721
512	0.8381

Table B.39: Exponential Smoothing Recurrent - RMSE - Validation - Nodes

Dilation level	RMSE - Validation set
1	17.6040
1, 4	7.8933
1, 4, 8	0.2897
1, 4, 8, 16	0.2480
1, 4, 8, 16, 32	0.5654
1, 4, 8, 16, 32, 64	0.2353

Table B.40: Exponential Smoothing Recurrent - RMSE - Validation - Dilation

B.9 Exponential Smoothing Recurrent Neural Network - Electricity Dataset

Hyperparameter	Initial Value	Final Value
Inputs	66	78
Nodes	66	66
Cell Type	GRU	GRU
Dilations	[1, 4, 8, 16]	[1, 4, 8, 16]
Epochs	65	65
Learning Rate	0.01	0.01
Percentile	50	50
Normalization Layer	True	True
Log Transformation	True	True
RMSE - Validation Score	973.4308	960.8040

Table B.41: Initial & Final Configurations - Exponential Smoothing Recurrent - Hyperparameters

Epochs	RMSE - Validation set
60	978.6932
65	973.4308
70	1144.4512
75	980.0269
85	1022.1935
90	1034.7187
95	1033.0277

Table B.42: Exponential Smoothing Recurrent - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
42	1015.9840
48	1075.2985
54	1022.1090
60	1099.1521
66	973.4308
72	1035.3105
78	960.8040

Table B.43: Exponential Smoothing Recurrent - RMSE - Validation - Inputs

Nodes	RMSE - Validation set
8	1051.8139
16	1009.6622
32	998.4593
64	1030.0062
66	973.4308
128	1096.5407
256	8822.9420

Table B.44: Exponential Smoothing Recurrent - RMSE - Validation - Nodes

Dilation level	RMSE - Validation set
1	1043.0223
1, 4	1044.0615
1, 4, 8	1038.0535
1, 4, 8, 16	973.4308
1, 4, 8, 16, 32	1027.5523
1, 4, 8, 16, 32, 64	1034.6547

Table B.45: Exponential Smoothing Recurrent - RMSE - Validation - Dilation

B.10 Exponential Smoothing Recurrent Neural Network - Iron Production Dataset

Hyperparameter	Initial Value	Final Value
Inputs	32	32
Nodes	32	32
Cell Type	LSTM	LSTM
Dilations	[1, 3, 6, 12]	[1, 3, 6, 12]
Epochs	35	35
Learning Rate	0.1	0.1
Percentile	40	40
Normalization Layer	True	True
Log Transformation	False	False
RMSE - Validation Score	88.8869	88.8869

Table B.46: Initial & Final Configurations - Exponential Smoothing Recurrent - Hyperparameters

Epochs	RMSE - Validation set
15	267.4131
20	330.4912
25	276.5010
30	230.3006
35	88.8869
40	150.0884
45	123.8782
50	101.9656

Table B.47: Exponential Smoothing Recurrent - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
4	160.6438
8	165.4803
12	106.0345
16	177.2274
20	200.6026
24	198.6514
28	95.0904
32	88.8869

Table B.48: Exponential Smoothing Recurrent - RMSE - Validation - Inputs

Nodes	RMSE - Validation set
4	92.1149
8	98.8408
16	93.9125
32	88.8869
64	242.3012
128	904.5695
256	4260.2483
512	372.5262

Table B.49: Exponential Smoothing Recurrent - RMSE - Validation - Nodes

Dilation level	RMSE - Validation set
1	104.0518
1, 3	137.3929
1, 3, 6	186.1612
1, 3, 6, 12	88.8869
1, 3, 6, 12, 24	94.2409
1, 3, 6, 12, 24, 48	94.7710

Table B.50: Exponential Smoothing Recurrent - RMSE - Validation - Dilation

B.11 Exponential Smoothing Recurrent Neural Network - Daily min. temperature Dataset

Hyperparameter	Initial Value	Final Value
Inputs	7	35
Nodes	16	16
Cell Type	LSTM	LSTM
Dilations	[1]	[1]
Epochs	35	35
Learning Rate	0.1	0.1
Percentile	50	50
Normalization Layer	True	True
Log Transformation	False	False
RMSE - Validation Score	2.5647	2.4497

Table B.51: Initial & Final Configurations - Exponential Smoothing Recurrent - Hyperparameters

Epochs	RMSE - Validation set
10	2.7869
15	2.6148
20	2.5762
25	2.5843
30	2.5743
35	2.5647
40	2.5630

Table B.52: Exponential Smoothing Recurrent - RMSE - Validation - Epochs

Inputs	RMSE - Validation set
7	2.5647
14	2.7066
21	2.5746
28	2.5774
35	2.4497
42	2.5677
49	2.5736

Table B.53: Exponential Smoothing Recurrent - RMSE - Validation - Inputs

Nodes	RMSE - Validation set
4	2.6029
8	2.7051
16	2.5647
32	2.7000
64	4.1202
128	10.0875

Table B.54: Exponential Smoothing Recurrent - RMSE - Validation - Nodes

Dilation level	RMSE - Validation set
1	2.5647
1, 3	2.6104
1, 3, 7	2.5773
1, 3, 7, 14	2.6913

Table B.55: Exponential Smoothing Recurrent - RMSE - Validation - Dilation

