Pedro Nuno
Gomes Gusmão

Previsão de Fluxo de Clientes com modelos baseados em Redes Neuronais

Retail Client Flow Forecasting with Neural Network based models

Pedro Nuno
Gomes Gusmão

Previsão de Fluxo de Clientes com modelos baseados em Redes Neuronais

Retail Client Flow Forecasting with Neural Network based models

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica da Profª. Doutora Ana Maria Perfeito Tomé, professora associada da Universidade de Aveiro, e do Prof. Doutor José Manuel Matos Moreira, professor auxiliar da Universidade de Aveiro.

Este trabalho foi parcialmente financiado pelo Fundo Europeu de Desenvolvimento Regional (FEDER) através do Programa Operacional Competitividade e Internacionalização no contexto do projeto "RH 4.0 FeD: Forecast e Dimensionamento Automático para equipas de Retalho", referência POCI-01-0247-FEDER-039719.

o júri / the jury

presidente / president

Professor Doutor Sérgio Guilherme Aleixo de Matos
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Doutor Rogério Luís de Carvalho Costa
Investigador do Instituto Politécnico de Leiria

Professora Doutora Ana Maria Perfeito Tomé
Professora Associada da Universidade de Aveiro

acknowledgments

With the accomplishment of this milestone, I thank my family for the graced opportunity.
My advisors for the placed thrust and priceless guidance.
My colleagues and friends, they rose my spirit in many occasions.
Lastly, I dedicate this effort to my partner, Tatiana, without her none of this would have been possible.

Palavras-chave Previsão em Séries Temporais, Redes Neuronais, Modelos Estatísticos, Lojas de Retalho.

Resumo

Com a disponibilização recente de variadas tecnologias informáticas em lojas de retalho, nomeadamente, a possibilidade de recolha de dados sobre visitas de clientes sob forma de séries temporais, campos emergentes como Big Data e Deep Learning podem ser aproveitados. Esta dissertação teve como um dos seus objetivos a exploração, teste e combinação de noções clássicas de séries temporais, como estacionariedade e decomposição, com múltiplas arquiteturas de redes neuronais, ao mesmo tempo investigando eventuais benefícios de tal combinação.

As experiências foram realizadas com dados de elevada frequência capturados em lojas de retalho reais e, por essa razão, um impasse que compromete o princípio da regularidade em séries temporais foi detetado. Para combater isso, um método para a uniformização dos horários das lojas foi desenvolvido e acabou por demonstrar melhorias muito significativas de performance dos modelos de previsão.

Tendo o requisito de prever longos horizontes (30 dias), todas as arquiteturas foram testadas com diferentes configurações correspondentes à sua estratégia de previsão (direta ou recursiva). Uma Multilayer Perceptron e duas redes Long Short-Term Memory (many-to-one) foram os melhores modelos de previsão apresentando MAAPEs melhores até $\sim$8% em relação às outras arquiteturas aplicadas.

Keywords Time Series Forecasting, Neural Networks, Statistical Models, Retail Stores.

Abstract

With the recent availability of computer technologies in retail stores, namely, the possibility of collecting data about customer visits in the form of time series, doors to emerging fields such as Big Data and Deep Learning are opened. This dissertation had as one of its goals the exploration, testing, and merging of classical notions of time series, such as stationarity and decomposition, with multiple neural network architectures, at the same time investigating possible benefits of such a combination.

Experiments were conducted with high-frequency (half-hourly) data captured from real retail stores and, for this reason, a deadlock compromising the principle of regularity in time series was detected. To combat this, a method to standardize the stores' schedules was developed and later demonstrated highly significant performance improvements of the prediction models.

Having the requirement of predicting large horizons (30 days), all architectures were tested with different configurations corresponding to their prediction strategy (direct or recursive). A Multilayer Perceptron and two Long Short-Term Memory (many-to-one) networks were the best prediction models showing better MAAPEs up to $\sim$8% in comparison to the other applied architectures.

Table of contents

Table of contents i

List of figures iii

List of tables vii

List of abbreviations ix

1 Introduction 1
- 1.1 Dissertation Structure . . . 3

2 Forecasting Methods 5
- 2.1 Judgemental Forecasting . . . 5
- 2.2 Quantitative Forecasting . . . 9
  - 2.2.1 Introducing Time Series . . . 10
- 2.3 Time Series Analysis . . . 11
  - 2.3.1 Decomposition . . . 11
  - 2.3.2 Stationarity . . . 12
  - 2.3.3 The Augmented Dickey-Fuller Test . . . 14
  - 2.3.4 Transformations . . . 15
- 2.4 Time Series Forecasting . . . 17
  - 2.4.1 Classical Models . . . 17
- 2.5 Forecast Error Metrics . . . 20
- 2.6 Summary . . . 22

3 Deep Learning and Artificial Neural Networks 23
- 3.1 Artificial Neural Networks . . . 23
- 3.2 Learning Designs . . . 26
- 3.3 Supervised Learning . . . 26
  - 3.3.1 Backpropagation and Gradient Descent Optimizers . . . 27
  - 3.3.2 Regularization Techniques . . . 29
- 3.4 Convolutional Neural Networks . . . 30
- 3.5 Recurrent Neural Networks . . . 34

3.5.1 Bidirectional RNNs . . . 37
3.5.2 Long Short-Term Memory Networks . . . 38
3.6 Summary . . . 40

4 Time Series Forecasting with Neural Networks 41
4.1 Neural Network Multi-Step Forecasting Strategies . . . 41
4.2 Convolutional Neural Networks . . . 42
4.2.1 WaveNet . . . 44
4.2.2 Adapted WaveNets . . . 47
4.3 Comparative Studies . . . 49
4.4 Summary . . . 52

5 Case Study and Experiments 53
5.1 Problem Definition and Datasets . . . 53
5.2 Evaluation Methodology . . . 53
5.2.1 Time Series Cross-Validation . . . 55
5.2.2 Train and Test Sets . . . 55
5.3 Data Preprocessing . . . 56
5.3.1 The Sliding Window Method . . . 56
5.3.2 Store Schedules: Standardization . . . 57
5.3.3 Detrending and Deseasonalizing with Harmonic Regression . . . 59
5.4 Technologies and Libraries . . . 61
5.5 Model Architectures and Configurations . . . 62
5.5.1 Baseline Model . . . 63
5.5.2 MLP . . . 63
5.5.3 LSTM-1 and BiLSTM-1 . . . 64
5.5.4 LSTM-2 and BiLSTM-2 . . . 66
5.5.5 SeriesNet . . . 67
5.6 Summary . . . 67

6 Results and Discussion 69

7 Conclusion and Work Ahead 77

References 79

Appendices 95

A Evaluation and Parameterization 97

List of figures

1.1 Online retail growth in different countries from 2012 to 2018. [Fisher and Raman, 2018] mention that it is growing linearly at a rate for about 1% per year. **Source**: Center for Retail Research, https://www.retailresearch.org/online-retail.html . . . 2

2.1 Ways to integrate judgement with objective methods in forecasting. **Adapted from**: [Webby and O'Connor, 1996] . . . 7
2.2 An example of a regular discrete TS: Recorded monthly sales of Australian wine between January 1980 and October 1991. **Source**: [Brockwell and Davis, 2016] . . . 10
2.3 An example of stationary and non-stationary TSs. **Adapted from**: [Brockwell and Davis, 2016] . . . 13
2.4 Simple exponential smoothing with three different values for α. In the smoothed signals, it is easy to observe that when $\alpha = 0.1$ there is more emphasis on older observations. **Source**: [*Forecasting - Exponential Smoothing* 2017] . . . 19

3.1 A representation of a single layer perceptron, first introduced in [Rosenblatt, 1957]. With an input layer with n units and an output layer with a single perceptron, containing **bias**, it generates an output through an activation function. Each connection between the input layer and the perceptron has an associated weight. . . . 24
3.2 The architecture of a MLP. Now, the ANN includes a hidden layer with m perceptrons, also usually called neurons or hidden units. . . . 25
3.3 Overview of the backpropagation process. a) The forward pass consists in the activations h_1, h_2 and, assuming that the output unit is activated, $\hat{y}$ that are processed with the inputs x_1 and x_2. The cost between the final output $\hat{y}$ and the target value y is then calculated. b) In the backward pass the cost is backpropagated to calculate the gradients with respect to the weights $w_1, ...w_6$. **Adapted from**:[Güneş Baydin *et al.*, 2018] . . . 28
3.4 Comparison between a standard FNN and a FNN with dropout. **Source**: [Khalifa and Frigui, 2016] . . . 31

3.5 The generic architecture of a CNN. **Source**: [Kiranyaz *et al.*, 2019] 31
3.6 A *2D*-convolution with **padding** $P = 1$. The feature map does not have the same size as the input due to the **stride** being $S = 2$. The first matrix (gray) is the kernel, the blue matrix represents the input and the green matrix is the resulting feature map. The gray color over the blue matrix represents the **receptive field**. **Source**: [Dumoulin and Visin, 2016] 33
3.7 An example of max-pooling. 34
3.8 Visualization of two filters, picked randomly, in five convolutional layers of a CNN. This depiction was accomplished with a deconvolutional network ([Zeiler *et al.*, 2010]) based method ([Ranzato *et al.*, 2006]). **Source**: [Qin *et al.*, 2018] 35
3.9 The reccurent neuron on the left can be represented by the units on the right if its processes are unfolded through time. x and y are the input and output, respectively, while W_x, W_h and W_y are the temporally shared weights for each connection. **Adapted from**: [Bao *et al.*, 2017] 35
3.10 The green rectangles (captioned as networks) hold the RNNs' internal state vectors. The one-to-one sequence problem is the conventional way of solving problems with MLPs and CNNs, a fixed-sized vector input results in a fixed-sized vector output. Since the remaining possess sequences of vectors in at least one of the sections they can only be modelled by RNN architectures. **Source**: [Mosconi *et al.*, 2019] 36
3.11 The RNNs represent the previously described recurrent units unrolled through t. The vector arrows on each $\boldsymbol{h}_t$ denote the different directions (positive and negative) that are taken to traverse the inputs for the calculations. In the end, the outputs from each unit ($\overleftarrow{\boldsymbol{h}}_t$ and $\overrightarrow{\boldsymbol{h}}_t$) are merged and activated (optional) at each step. **Adapted from**: [Cui *et al.*, 2018] 37
3.12 The recurrent processes in a LSTM unit. **Adapted from**: [Varsamopoulos *et al.*, 2019] 39

4.1 Forecasting with the recursive strategy. **Adapted from**: [An and Anh, 2015] 42
4.2 Example of an ANN architecture for a direct forecast. **Source**: [Hamzaçebi *et al.*, 2009] 43
4.3 The sliding occurs only in the time dimension. Thus, performing a *1D* convolution, in this case, applied to a multivariate TS, i.e., k context-related TSs with the same length. The filter has its height fixed to the number of TSs (size $4 \times k$). 43
4.4 Stacked dilated causal convolutional layers. **Source**: [*Convolutions in Autoregressive Neural Networks* 2019] 44
4.5 The gated convolutional layer of the WaveNet model. **Adapted from**: [Oord *et al.*, 2016] 45

4.6 A residual connection skipping two convolutional layers. In this case, the residual function $\mathcal{F}$, which represents two convolutional layers, and the input x have matched dimensions. **Source**: [He *et al.*, 2016] 46
4.7 Comparison of residual CNNs with plain CNNs, all being 34-layers-deep networks. In ResNet-34 A the zero-padding strategy was used, along with the rest of the residual connections being parameter-free. B used parameterised skip connections only in situations where dimensions needed to be increased and C used parameterised skip connections at all times. **Adapted from**: [He *et al.*, 2016] 46
4.8 The WaveNet's architecture. **Source**: [Oord *et al.*, 2016] 47
4.9 The Augmented WaveNet architecture. **Source**: [Borovykh *et al.*, 2018] . . 48
4.10 The SeriesNet architecture. **Source**: [Papadopoulos, 2018] 49
4.11 Evaluation of forecasts on various horizons (best in bold). The short-term horizon regards to 1 to 6 months ahead, medium-term 7 to 12 months ahead and long-term 13 to 18 months ahead. CC is the computational complexity, it refers to the time a model requires to train and, lastly, predict. **Source**: [Makridakis *et al.*, 2018a] 50
4.12 Results for one-step-ahead forecasts, now with the LSTM and others ANNs. **Source**: [Makridakis *et al.*, 2018a] 50
4.13 a) Models' popularity distribution. The outer circle considers studies from three years previous and up to 2019, while the inner circle considers all the studies reviewed. b) From all the RNN models LSTMs are the preferred approach. **Source**: [Makridakis *et al.*, 2018a] 51

5.1 The first 1400 observations (∼2 months) of each TS. 54
5.2 The COVID-19 lockdown effect is visible in both stores around the middle of March. 54
5.3 TS CV in an expanding walk-forward basis. 55
5.4 The sliding window sampling method. **Adapted from**: [Paoli *et al.*, 2010] 57
5.5 The constructed framework to apply the HR decomposition method. 60
5.6 FFT plot of store C's third walk-forward iteration. 60
5.7 Fitted HR plot of store A's first walk-forward iteration training set. 62
5.8 The graph on the right is the result of the Autodiff algorithm executed on the left operation (forward pass of a feedforward network). The cost C calculations are not shown for simplicity reasons. **Source**: [Abadi *et al.*, 2016] 62
5.9 The MLP architecture for multi-step forecasting. 64
5.10 The LSTM-1 (many-to-one) architecture. 65
5.11 The BiLSTM-1 (many-to-one) architecture. 65
5.12 The LSTM-2 (many-to-many) architecture. 67
5.13 The BiLSTM-2 (many-to-many) architecture. 68

6.1 748-step forecasts done by the best MLP and SeriesNet (as in Table 6.5). . 73
6.2 743-step forecasts done by the LSTM many-to-many (best) and many-to-one (worst) models. 73
6.3 679-step forecasts done by LSTM-1 with $p = 10$ (worst) and with $p = 175$ (best). 74
6.4 679-step forecasts done by LSTM-2 with the three different data settings. . 75

List of tables

5.1 The sizes of each store test sets. 56
5.2 Number of imputations against the size of the training sets. 58
5.3 The number of daily occurrences of the three most frequent schedules (MFS) on the initial training set (the order does not change with further walk-forward iterations). 58
5.4 ADF test results on store A training sets. 59
5.5 ADF test results on store B training sets. 59
5.6 ADF test results on store C training sets. 61
5.7 Number of learnable parameters of the MLP used on the raw setting. 64

6.1 Store A's raw setting results. 69
6.2 Store B's raw setting results. 70
6.3 Store C's raw setting results. 70
6.4 Store A's standardized setting results. 70
6.5 Store B's standardized setting results. 71
6.6 Store C's standardized setting results. 71
6.7 Store A's standardized plus HR setting results. 71
6.8 Store B's standardized plus HR setting results. 72
6.9 Store C's standardized plus HR setting results. 72
6.10 The best performance of BiLSTM-1 on store B's raw setting. 75

A.1 Number of forecasts that are valid for evaluation. 97
A.2 Number of learnable parameters of the LSTM-1 on the unstandardized setting ($p = 25$). 97
A.3 Number of learnable parameters of the BiLSTM-1 (unstandardized) ($p = 25$). 97
A.4 Number of learnable parameters of the MLP with $p = H$ if it had one output unit (unstandardized). 98
A.5 Number of learnable parameters of the LSTM-2 (unstandardized). 98
A.6 Number of learnable parameters of the BiLSTM-2 (unstandardized). 98
A.7 Configuration and number of learnable parameters of the SeriesNet. 98

List of abbreviations

ADAM	Adaptive Moment Estimation
ADF	Augmented Dickey-Fuller
AIC	Akaike information criterion
ANN	Artificial Neural Network
API	Application Programming Interface
AR	Autoregressive
ARIMA	Autoregressive integrated moving average
ARMA	Autoregressive moving average
Autodiff	Automatic Differentiation
BiLSTM	Bidirectional Long Short-Term Memory
BPTT	Backpropagation Through Time
BRNN	Bidirectional Recurrent Neural Network
CNN	Convolutional Neural Network
CV	Cross-Validation
DL	Deep Learning
DNN	Deep Neural Network
ETS	Exponential Smoothing
FFT	Fast Fourier transform
FNN	Fully-connected Neural Network
FSS	Forecasting Support System
HR	Harmonic Regression
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LSTM	Long Short-Term Memory
MA	Moving Average
MAAPE	Mean Arctangent Absolute Percentage Error
MAE	Mean Absolute Error

MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NLP	Natural Language Processing
PACF	Partial Autocorrelation Function
ReLU	Rectified Linear Unit
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SELU	Scaled Exponential Linear Unit
SGD	Stochastic Gradient Descent
sMAPE	Squared Mean Absolute Percentage Error
TS	Time Series
VAR	Vector Autoregression

Chapter 1

Introduction

In the last few decades, the retail commerce has experienced significant developments as various technologies are continuously being introduced. Nevertheless, in some cases, strategic or operational decisions such as the introduction or discontinuation of products in the market or the investment in new technologies are still mainly supported by the experience of managers that do not take advantage of technological approaches, namely, from areas such as Big Data and Machine Learning (ML). Indeed, some companies are beginning to convert from judgemental forecasting to data-based solutions and to achieve that goal, multiple measures that possibilitate recording data from clients are put into effect. A measure of such class can be, for example, the creation of discount cards or coupons that require clients to preregister before accessing promotional campaigns. Such initiatives possibilitate the attainment of useful information that can be used in forecasting methods.

Outside the forecasting scope, there are proven benefits obtained from the computerization and the data retail companies already collect. In [Weber and Kantamneni, 2002], it is delineated what improvements managers searched for and have gotten with the use of point-of-sales data and the exchange of data done between suppliers, stores or other collaborative organizations. Some of those improvements were shorter checkout times, reduced stockouts in stores and reduced investments in inventory. However, as years passed, new advantages emerged with the global recognition of Big Data.

Big Data is a discipline that studies big volumes of data with structure or not, these volumes of data are usually so large that it is nearly impossible to process them with traditional methods. This area of knowledge is aligned with retail forecasting but before jumping to these concepts it is important to have some more background behind the retail market. In [Fisher and Raman, 2018], it is declared that stores are still very much active and have a greater share of sales than e-commerce, this is expected to remain the same for at least a couple decades (see Figure 1.1), experienced retailers get better results optimizing sales on their existing stores instead of trying to expand, also labor planning is still a very significant problem to solve as it has a great effect on profits. In [Gaur *et al.*, 2014], it is described that inventory management is becoming a greater challenge

as the diversity of products increases and that it is key to minimize errors since market capitalization can be heavily impacted, especially with brick-and-mortar retailing since it is a high-volume low-margin market.

Figure 1.1: Online retail growth in different countries from 2012 to 2018. [Fisher and Raman, 2018] mention that it is growing linearly at a rate for about 1% per year. **Source**: Center for Retail Research, https://www.retailresearch.org/online-retail.html

Knowing these aspects and challenges we can move on to capture a general view on how forecasting methods, allied with Big Data, can have positive impact in retail companies. As mentioned before, in some areas of retail even the smallest management mistake can have a great impact, consequently, a resultful future planning can prove to be invaluable and having accurate client flow and sales forecasts can be significant towards that end. Forecasts are usually acquired through a set of procedures that form a Forecasting Support System (FSS). A FSS is built to ease the task of the company's forecasters and, besides other features, it always includes a set of quantitative methods ([Petropoulos, 2015]).

This work was made with a special purpose, it is in parallel with a research made for a company who provides FSSs for many retail stores. The objective is, predominantly, to explore the quantitative methods section of the FSSs by applying a wide range of models.

Quantitative forecasting methods are usually applied on numerical data referred to as Time Series (TS). Since TS data began to be available for forecasting, classical statistical models have been the preferred approach (e.g. Autoregressive integrated moving average (ARIMA) and Exponential Smoothing models), however there has been a recent growing interest in ML models due to the mentioned increase volume of data, more specifically, Deep Learning (DL) models. These, such as, Artificial Neural Networks (ANNs), have the capability to easily incorporate different variables and capture complex nonlinear relationships present in the data which can improve forecasts. In spite of this rise of interest, some authors point out that DL models do not perform better than classical methods, while others recommend merging statistical and DL worlds. Nevertheless and as it will be shown, there is no question that DL is quickly evolving and moving to become the new preferred approach. Hence, in order to better understand DL forecasting models,

this work will detail multiple DL architectures, namely, ANN architectures. It will also be explaining how they have performed in past forecasting challenges, e.g. the famous Makridakis Competitions ([Makridakis and Hibon, 2000]), and will be demonstrating their abilities in a real world scenario alongside a few classical techniques.

1.1 Dissertation Structure

Including this one, this dissertation is composed of seven chapters. The remaining chapters are organized as follows:

- Chapter 2 gives a background on a variety of forecasting methods, quantitative and judgemental, data-based or not;
- Chapter 3 details the fundamentals of the inner processes of ANNs, including feedforward and recurrent networks. This, alongside various ML and DL specific notions;
- In Chapter 4, various works that describe and compare state of the art ANN strategies and models for TS forecasting are explored;
- In Chapter 5, experiments with several ANN architectures, strategies and configurations on a real world case study are detailed;
- In Chapter 6, the performances of the models are evaluated and discussed;
- Chapter 7 gives the final conclusions about the conducted experiments and suggests further work.

Chapter 2

Forecasting Methods

In this chapter, various methods belonging to the two main classes of forecasting will be unfolded. A great part of this chapter details some methods that will not be brought up in further chapters. The reason for this is to provide background at forecasting in general. Differences between methods are exposed with special focus on TS forecasting as it is most applicable to DL models.
Forecasting models can be divided in two classes ([Chambers *et al.*, 1971]):

- Qualitative, subjective or judgemental methods;
- Quantitative or objective methods.

2.1 Judgemental Forecasting

Judgemental forecasting typically makes use of domain knowledge, therefore, the process is originated through the judgement of experts or discussions happening in the company's board. Qualitative methods are usually, but not always, used when data is not available or when the data does not fit the required standards to employ quantitative methods. Even so, there has been debates as to whether these methods should be given relevancy in the forecasting practice. Whilst discussing the problems of objective forecasting, in [Makridakis, 1988] it is stated that, in most cases, not only the accuracy obtained from intuitive approaches is worse than from statistical models, judgemental forecasts are costlier to apply as well. Additionally, the author also states that judgemental forecasts are affected by chimerical conjectures and political affinities[1]. It is also worth mentioning that, qualitative methods are now usually a required feature in FSSs ([Petropoulos, 2015]). Contrast, however, is displayed in [Lawrence *et al.*, 2006], where it is claimed that judgemental forecasting was becoming more accepted especially when combined with statistical models as both methods have different advantages.

Multiple works, for example, in [Webby and O'Connor, 1996] and in [Lawrence *et*

[1] Concerning those last aspects, in [Hogarth and Makridakis, 1981] it is presented an in-depth research on psychological factors that can negatively influence forecasts.

al., 2006], mention the benefits obtained from merging or associating quantitative with qualitative methods. Therefore, judgemental forecasting originated entirely from just looking at data (e.g. "eyeballing" trends in plots), i.e., without adding expert domain knowledge, is not given attention in this chapter. Furthermore, as this forecasting class is not the main focus of this work, the next subsections will only provide some background regarding two famous qualitative methods and briefly outline the broached combination processes.

The **Delphi Technique** is a known method which involves intensively applying enquiries to obtain consensus from a team of domain experts that are kept ignorant of identities. The aim is that factors such as reputation, personalities and social pressures are not taken into account. Delphi has four foundational characteristics: anonimity, iteration, controlled feedback, and the final statistical composition ([Rowe and Wright, 1999]). The typical stages are the following ([Hyndman and Athanasopoulos, 2018]):

1. A study is conducted with the objective of assembling a group of specialists with diverse areas of expertise;
2. The forecasting problems are issued to the specialists;
3. The group returns the forecasts with their justifications to be analyzed;
4. Specialists are given feedback to further refine their own forecasts. This stage can be repeated until harmony is reached;
5. A final forecast is generated by combining all of the specialists' forecasts.

As a key element, the coordinator may construct the final forecast by giving the same significance to all specialists' forecasts or in specific cases, by applying different significances.

Forecasting by analogy is a method that encompasses comparisons between non-identical incidents with identical behaviours. A prevalent example of this in retail is an attempt at predicting sales of a newly introduced product. To achieve this, the market can be analyzed for products that are included in the same category or that possess similar features. With similarities to the Delphi technique, i.e., a coordinator is also assigned, forecasting by analogy assumes five steps ([Green and Armstrong, 2007]):

1. The coordinator gathers specialists that usually are knowledgeable about analogous incidents;
2. The forecasting problem is given to the specialists;
3. The experts identify, detail and forecast on analogous situations;
4. The coordinator requests the experts to evaluate the similarities and differences between their analogies and the forecasting problem;

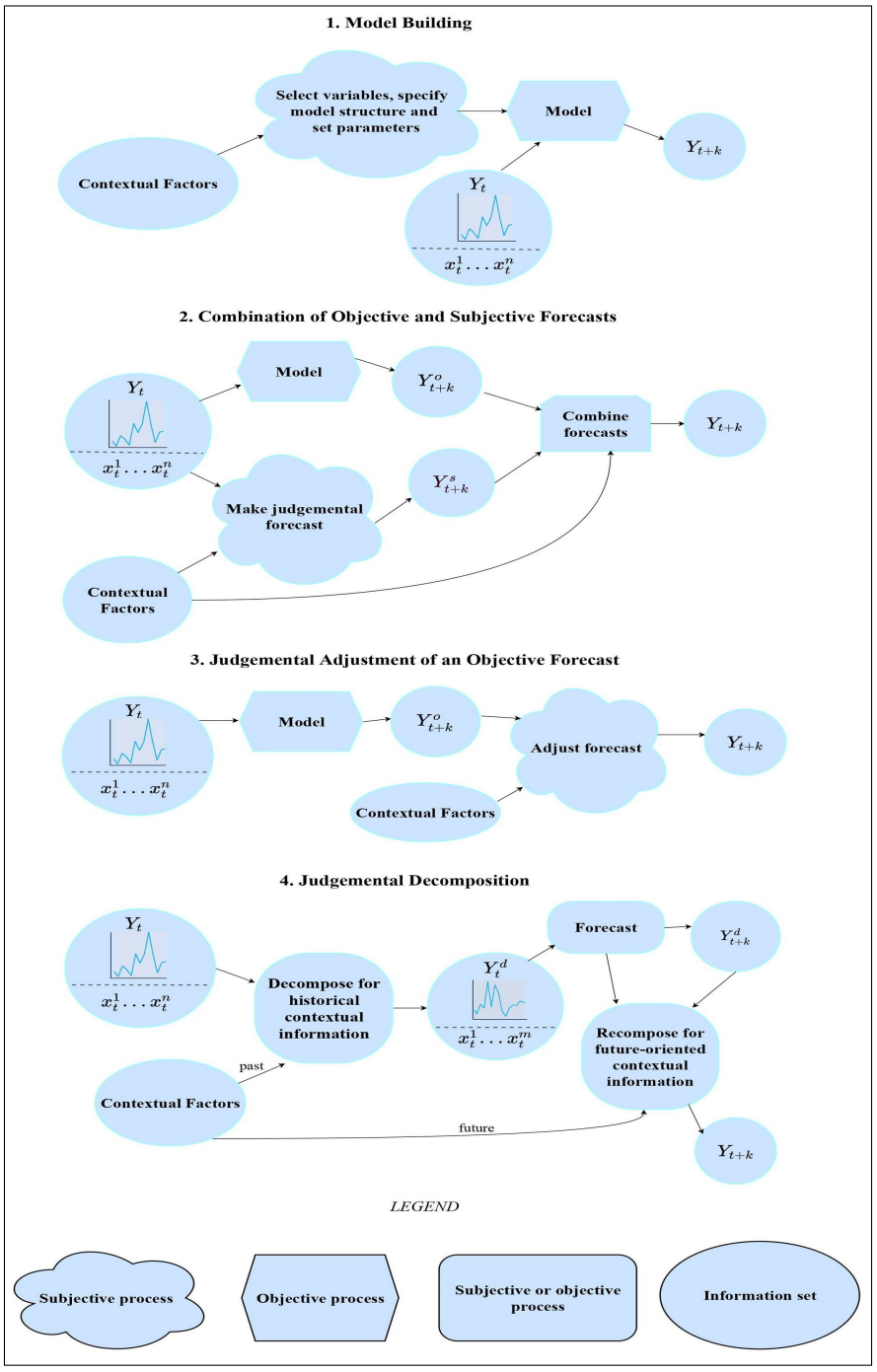

Figure 2.1: Ways to integrate judgement with objective methods in forecasting. **Adapted from**: [Webby and O'Connor, 1996]

5. The coordinator generates a forecast from the experts' analogies. This can be done by various rules such as, for example, picking the forecast from the most similar analogy.

In [Webby and O'Connor, 1996], four approaches (Figure 2.1) to **integrate judgement on quantitative forecasts** are described, these are:

- Model Building;
- Forecast Combination;
- Judgemental Adjustment;
- Judgemental Decomposition.

Judgement in **model building** can be applied without the practitioner actually perceiving it. It involves constructing or choosing the models through judgemental variable selection and data analysis, and even though there are statistical methods to determine significance, judgement is still the key reason for their selection ([Bunn and Wright, 1991]). This strategy was also validated by the research presented in [Petropoulos *et al.*, 2018], stating that selecting models judgementally consequenced in results that were as skillful as model selection through algorithms.

Combining quantitative with qualitative forecasts has proven to build a positive effect to the accuracy of predictions ([Adya *et al.*, 2001] and [Clemen, 1989]), as both classes of forecasting have properties that can benefit the forecast differently and at the same time compensate weaknesses that both may withold. Combinations can be obtained by applying simple averages or averages where forecasts are assigned different weights. However, weight assignment should be done with caution if not in a mechanical way, since research shows that practitioners place more weight on forecasts that are their own ([Harvey and Harries, 2004]).

Judgemental adjustment, in contrast to forecast combination, comprises of altering through judgement an already created objective forecast. Before adjusting a forecast, three principles should be considered ([Sanders and Ritzman, 2001]):

- Adjustment of quantitative forecasts should only happen if the involved domain knowledge is valuable to the task. Practitioners that are experienced in the specific context are usually in possession of this knowledge. These practitioners can identify characteristics that benefit the forecasting task and reduce emphasis on factors that do not;
- Judgemental adjustment can be considered if predictions demonstrate high uncertainty;
- Lastly, adjustment is an option when there are known changes to the environment being forecasted (e.g. factory malfunction).

Although the first principle is clear about the adjustments being only applied if domain knowledge is present, there has been records of non-contextual adjustments (graphical adjustment) improving accuracy when the produced statistical forecast is highly unreliable ([Willemain, 1991]). A negative critique that is given to this approach, is that practitioners apply adjustments without structuring the process, this consequences in mistakes such as adjusting too many times or trying to correct systematic observations that the quantitative model might have missed [Hyndman and Athanasopoulos, 2018].

Judgemental decomposition, such as judgemental adjustment, is an approach that completes a quantitative forecast, meaning that there is data available such as TSs. Fundamentally, this approach involves dividing the forecasting problem into smaller and easier tasks before possessing a definitive prediction, this being the main difference between this approach and judgemental adjustment. Incidentally, this division purposelly consequences in a mitigation of past effects to better evaluate the current status of the forecasting problem. In the end, all the resulting elements are to form a combined forecast. An example of a judgemental decomposition strategy with TSs is presented in [Edmundson, 1990]. The author discovered that by separately extrapolating the components of TSs and later merging the forecasts resulted in better accuracy. Nonetheless, considering this to be part of a judgemental process is quite an outdated belief as various decomposition methods are available (e.g. [Xu *et al.*, 2019]) and can be picked in an automatic manner.

2.2 Quantitative Forecasting

First, it is important to define what is considered to be a forecast. A forecast is perceived as being either a point forecast, an interval forecast (e.g. predicting sales to be included in a specific interval), or a probability forecast, the latter consists in estimating probabilities for a range of possible outcomes. All these notions are presented only for comprehension reasons since the approaches expressed in this work are restricted to point forecasting.

There are two different categories of models within the class of quantitative forecasting. One of the categories refers to **causal, relation or explanatory models**. These methods highly focus on information that regards to relationships between the variable of interest and the various factors that surround the forecasting problem, i.e., these methods try to identify influences that act on the variable that is to be forecasted. This is called multivariate forecasting and, even though, multivariance is relevant to the task of TS forecasting it will not be covered on this dissertation. The work in [Chambers *et al.*, 1971] is referenced as it provides practical descriptions of multiple causal methods.

The other category refers solely to the TS genre of data. Prior to the final task of generating forecasts on TS (i.e., executing the final forecasting model), the practitioner has been frequently advised to take certain steps regarding TS modeling and analysis[2]. Obviously, these steps present themselves as common practices and are not be considered

[2] A general approach for TS modeling can be seen in [Brockwell and Davis, 2016].

fixed or mandatory, still there is a high degree of accordance that characterizes these practices. However, ML models, especially DL models, have arised recently and altered the perspective quite significantly. The reason is that, over the years, most concepts were built on the foundation of statistical models. This work will, therefore, present an attempt at finding how concepts from the worlds of statistics and DL, on TS forecasting, can be combined and what benefits originate from such a combination.

2.2.1 Introducing Time Series

TS is a category of data that is present in many fields such as astronomy, signal processing and any subject that works with temporal measurements. TS data is frequently collected in areas that include forecasting tasks, for example, meteorology with weather forecasting, or seismology with earthquake predictions. Technically, TS is any collection of values $x(t)$ observed at particulars points in time t.

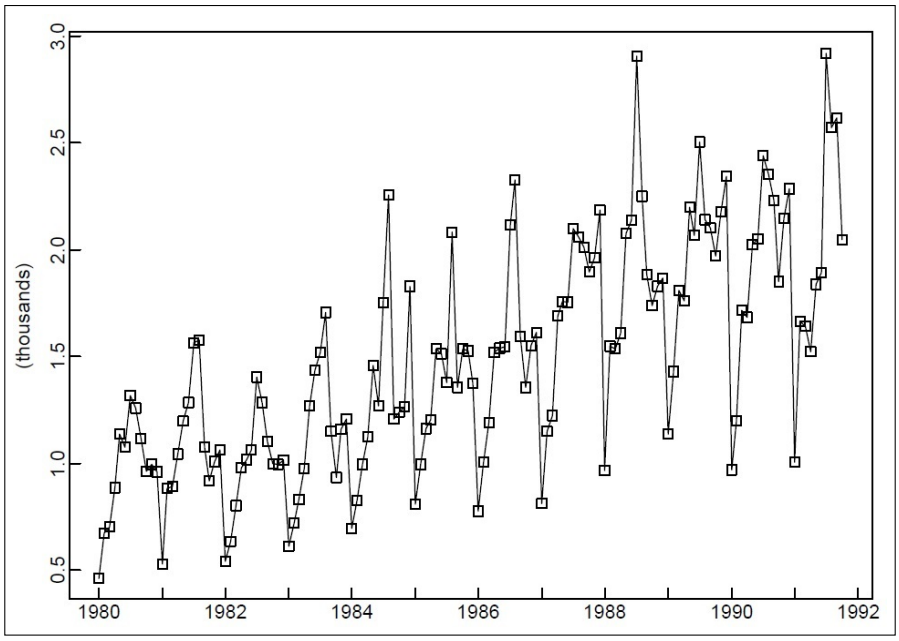

Figure 2.2: An example of a regular discrete TS: Recorded monthly sales of Australian wine between January 1980 and October 1991. **Source**: [Brockwell and Davis, 2016]

TS can be branched into two categories: **discrete** and **continuous**. A discrete TS means that its values occur at fixed moments in time, commonly having equally spaced[3] time intervals between them. In contrast, a continuous TS demonstrates a continuous record of observations, meaning that there are no intervals between them, even if their values are repeated. Figure 2.2 shows an example of the genre of TS data.

For future reference, it is important to establish that the length of time into the future for which forecasting models are prepared is specified as **forecast horizon**, denoted by

[3] A TS not having this last characteristic means that it is irregular. These inconsistencies occur frequently in real-world problems and consequence in harder forecasting tasks. Research about how to deal with irregular series is still in active development.

H and representing each step of the horizon there is $h \in \{1, ..., H\}$.

Elaborating the TS forecasting problem, let us assume that observations are regular and discrete, for example, in a product demand problem, the quantity sold $x(t)$ in the present week t and the sales $x(t-1), x(t-2), x(t-3), ...$ obtained in the previous weeks may be useful to predict the sales, for h=1,2,3,...,8 weeks ($H = 8$) in the future. Designating $\hat{x}(t + h|t)$ as the forecast of time $t + h$ made at t, then, the function $\hat{F}$ produces H predictions, one for each h, from the present and previous observations $x(t), x(t-1), x(t-2), x(t-3), ...$ is called the **forecast function**. The objective is to generate a forecast function that minimizes the mean squared errors (section 2.5) between actual values and forecasted values for all periods h.

2.3 Time Series Analysis

2.3.1 Decomposition

In order to better understand TSs, it might be useful to separate them in multiple components, namely:

- **Trend**: characterized by when there is a long-term rise or drop in the data, it does not need to be linear and it can change directions;
- **Seasonal**: seasonality takes place when the data demonstrates periodic fluctuations that usually occur in particular calendar seasons recurring every year (e.g. sales of coats in the winter);
- **Cycle**: a cyclic pattern refers to periodic fluctuations that are not of fixed period (not to be confused with seasonality). Cycles can be shorter than a calendar year but commonly describe longer term behaviours (more than a year). Cycle is usually merged with the trend component;
- **Noise** (can also be called the random, irregular or error term): represented by irregular random sources of level variations or fluctuations.

It can be important to identify and sometimes separate these patterns as they can help modeling the data for forecasting methods and, consequently, improve forecasts ([Edmundson, 1990]). There are two classical ways to decompose a TS: an **additive** model and a **multiplicative** model. If an **additive** model is considered, a TS can be described as

$$x(t) = m(t) + s(t) + \varepsilon(t) \tag{2.1}$$

Here, $m(t)$ is the trend-cycle component, $s(t)$ is the the seasonal component and $\varepsilon(t)$ the irregular or noise component, all at time $t \in [0, 1, ..., N-1]$ with N being the length of the TS. The additive model is the usual pick since it is used when seasonal variations

are constant. On the other hand, if seasonal variations increase over time then the **multiplicative model** is used, it can be specified in similar manner as

$$x(t) = m(t) \times s(t) \times \varepsilon(t), \tag{2.2}$$

and what these definitions provide are structures for basic decomposition methods. In [Hyndman and Athanasopoulos, 2018], different methods for TS decomposition are detailed.

2.3.2 Stationarity

When building TS forecasting models and for a great part of the known classical models, the presence of some statistical qualities in the data can be crucial to create skillful forecasts. **Stationarity** is one of those desired features ([Oliver and Gujarati, 1993]), as it provides some sort of **statistical balance** in TSs. Intuitively, a TS presenting **strict stationarity** means that all of its properties are unaltered by any time shift. However, before formally expressing concepts related with stationarity, figuring why statistical balance is crucial to a majority of models (especially classical statistical models) requires an understanding of the fundamental statistical processes in which TSs are included. A TS can be a realization of either a **deterministic** or a **stochastic** process. A **deterministic** TS represents a series in which its values can always be determined by some mathematical function. In contrast, Figure 2.2 shows an example of a **nondeterministic** TS and, even though patterns can be seen in the data, the values can only be described in terms of a probability distribution. In other words, this example was originated by a stochastic process. A stochastic process is defined by any statistical process that describes an evolution in time of a random phenomenon, i.e., random variables. These random variables are predetermined at fixed points in time and have a probability distribution for which the mean is not possible to calculate, since realistically only one realization can be observed. If humans had the ability go to back in time, a cross-sectional mean for each point could be calculated, that not being obviously possible, the option is to compute the mean of the obtained realization (the TS) across time. However, to converge this mean to the mean of all realizations, a statistical balance is necessary and this is achievable with **stationarity**.

A key type of stochastic processes are **stationary** processes. An important distinction to establish is that stationarity is a possible feature of stochastic processes and not of their realizations (TS). Although, for simplicity, this property will sometimes be associated to TS and not to the stochastic process. There are different types of stationarity. Strict stationarity means that the distribution of a finite sub-sequence of random variables is not altered when time is shifted along its axis. Formally, $x(t)$ is a **strictly stationary** TS if

$$\Big(x(1), ..., x(k)\Big)' \overset{d}{=} \Big(x(1+l), ..., x(k+l)\Big)' \tag{2.3}$$

for all integers l and $k \geq 1$, $\overset{d}{=}$ expresses that the two random vectors have the same joint

distribution function. This, however is quite a strong assumption as it is extremely hard to achieve and observe in the data. Instead, a less restrictive process is used which is called **weak stationarity**[4]. Weak stationarity states that the mean and variances of a stochastic process are not functions of time (they are constant). Conventionally, let $x(t)$ be a TS with $E[x(t)^2] < \infty$, the mean of $x(t)$ is

$$\mu_x(t) = E\Big[x(t)\Big]$$

and the autocovariance function of $x(t)$ is

$$\gamma_x(r,s) = \mathrm{Cov}\Big(x(r), x(s)\Big) = E\Big[\Big(x(r) - \mu_x(r)\Big)\Big(x(s) - \mu_x(s)\Big)\Big] \quad \forall\, r, s \in \mathbb{Z}$$

where r and s represent two points in time. Then, $x(t)$ is **weakly stationary** if (2.4)

1. $\mu_x(t)$ is independent of t;
2. $\gamma_x(t+l, t)$ is independent of t for each l.

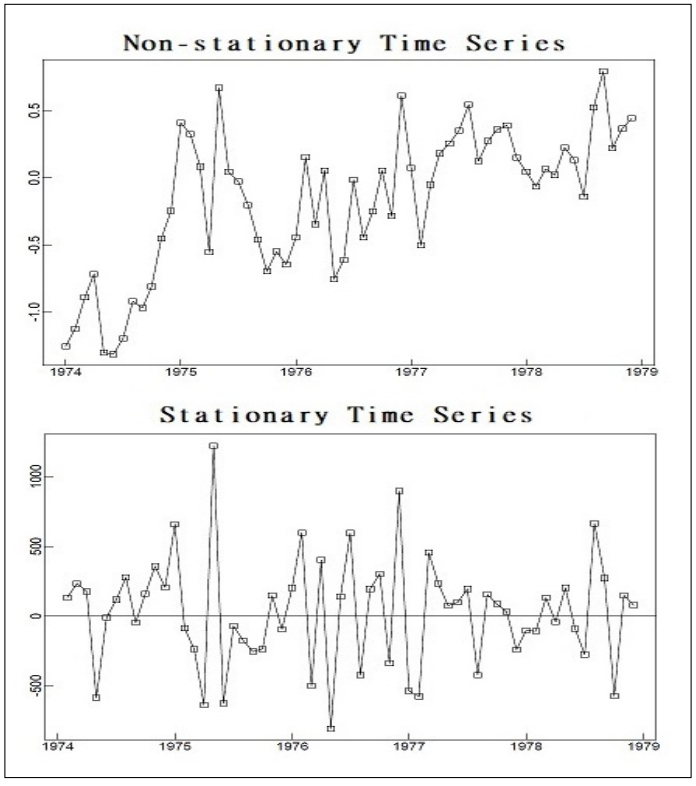

Figure 2.3: An example of stationary and non-stationary TSs. **Adapted from**: [Brockwell and Davis, 2016]

By plotting the TSs or their partial autocorrelation functions (PACF), it might be trivial to observe if there any signs of seasonality, however, that is not always the case for trends. Thus, a helpful way to know if the TS is trend-stationary is by applying statistical tests.

[4] For the sake of simplicity, in the next sections we will be referring to this as just stationarity.

One of the most popular is the unit root **augmented Dickey-Fuller** (ADF) test ([Fuller, 1996]).

2.3.3 The Augmented Dickey-Fuller Test

Unit roots refer to stochastic trends in TSs. Intuitively, unit root tests evaluate how a TS is defined by trends. When a TS shows no signs of recovery from a trend it implicates the presence of a unit root.

The null hypothesis of the ADF test states that the TS has at the very least one unit root, on the other hand, rejecting the null hypothesis means that the process is trend-stationary. In essence, the application of this test results in a t-statistic and a p-value that provide the answers. The resulting p-value is traditionally compared to a threshold of 5%, if it is lower than the threshold or the t-statistic[5] is smaller than the corresponding value in the Dickey-Fuller distribution table ([Fuller, 1996]) or the critical values ([MacKinnon, 2010]) then it means that the null hypothesis is rejected and, thus, the TS is stationary.

To achieve stationarity, the TS is transformed until the null hypothesis is rejected, and in order to ensure that, the ADF test is executed repeatedly. Formally, the ADF test works through an autoregressive model. An autoregressive model refers to when values from a TS are linearly dependent (or simply regressed) on their previous values. For example, an AR of order $p = 1$ or AR(1), i.e., for $x(t)$ on $x(t-1)$, is given by

$$x(t) = c + \beta t + \phi x(t-1) + \varepsilon(t), \tag{2.5}$$

where $x(t)$ is the variable of interest, c a constant, β the coefficient on a deterministic trend, and $\varepsilon(t)$ is the white noise term (a white noise sequence with zero mean and constant variance σ^2). Thus, for every order $p > 1$, $\phi_2 x(t-2), \phi_3 x(t-3), ..., \phi_p x(t-p)$ are added to the equation. Here, a unit root is present if $\phi = 1$. With this in mind, an early version, named the Dickey-Fuller test[6] can be described by first considering the AR(1) equation defined by

$$\Delta x(t) = x(t) - x(t-1) = c + \beta t + \gamma x(t-1) + \varepsilon(t), \tag{2.6}$$

this is turned into a regression model so that γ (which is equal to $\phi - 1$, as here a unit root is present if $\gamma = 0$) can be calculated through a linear regression. If $\gamma = 0$, then there is a **unit root**. However, if $-1 < 1 + \gamma < 1$ then the null hypothesis is rejected and the process is trend-stationary.

The **augmented Dickey-Fuller** is a more powerful test since it allows higher-order

[5] Intuitively, the t-statistic quantifies the statistical significance of the result (rejecting or not the hypothesis), in this case the more negative it is the more confidence the test has in its result.

[6] There are three more slightly different versions, which regard to the imposal of constraints on the values of c and β to verify different types of unit roots ([Hamilton, 1994]).

autoregressive processes, eliminating autocorrelation[7], by adding $\Delta x(t-p)$ to the equation. The same values for which γ is tested remain, and the process can be written as

$$\Delta x(t) = c + \beta t + \gamma x(t-1) + \delta_1 \Delta x(t-1) + ... + \delta_{p-1} \Delta x(t-p+1) + \varepsilon(t), \tag{2.7}$$

where p is given as the lag order of the autoregressive process. For this parameter, a famous statistical model quality estimator named **Akaike information criterion** (AIC) can be used to estimate p[8] ([Ng and Perron, 1995]).

2.3.4 Transformations

There are many types of TS transformations that, in combinations or individually, can accomplish stationarity. This work will be detailing two of the most popular classes, i.e., **difference** and **power** transforms, and also a single method that uses a linear regression over Fourier terms, i.e., a **harmonic regression** (HR).

Differencing is one way of transforming the data and it is usually used after a **power transform**. Differencing is a way of balancing the mean of TS by eliminating fluctuations, i.e., reducing (or even removing) trend and seasonality patterns ([Hyndman and Athanasopoulos, 2018]). It is processed through the calculation of differences between consecutive observations. Formally, it is written as

$$x'(t) = x(t) - x(t-1), \tag{2.8}$$

and, although rarely necessary, this method can be employed more than once, usually with its order notated as d (e.g., $d = 2$ represents second-order differencing[9]). Although, everytime it is executed it generates a differenced series with one less observation, this is simply because it is not possible to calculate the difference with the first observation. Since it only focuses on consecutive observations, this formula only handles trends and, as mentioned, difference transforms can be used to remove or reduce seasonal patterns. This is called **seasonal differencing**. It consists of subtracting the current observations with the corresponding previous observations of the same season. It can be written as,

$$x'(t) = x(t) - x(t-z) \tag{2.9}$$

where z corresponds to the seasonal period or frequency, and is provided depending on the TS data frequency (for example, monthly or yearly).

Power transforms alter the TS by stabilizing the variance and reducing noise, thus, usually improving the signal. Power transforms are a class of methods that use power

[7] Occurs when a variable is correlated to a lagged version of itself over periods of time.

[8] There are other methods for this, for example, the order can be attributed through a PACF plot. Technically, samples from the PACF that are significantly far from 0 may indicate useful values for p, i.e., values for p that cause the autoregressive model to fit the data better.

[9] Not to be confused with *second differencing*, which consists of subtracting the values that are placed two periods back with the present ones.

functions (e.g. a logarithmic transform). A famous method is the **Box-Cox transform** ([Box and Cox, 1964]). It only operates on TSs that are strictly positive[10], however, that does not pose a significant drawback as a constant can be added to the values. Technically, the **Box-Cox transform** is used to find the optimal power transform for the TS.

Another solution is dealing with both trend and seasonality by removing from the TS the calculated coefficients of a **HR**. A HR is simply a linear regression model that contains trigonometric regressors, in this case, the regressors are Fourier terms. Specifically, Fourier terms are sine and cosine pairs where each pair is included to represent one seasonal pattern. If a TS displays a number of seasonal patterns, then the same number of terms should be included. This can be a useful method considering that some TSs can demonstrate multiple seasonalities. The following equations explain the model step by step.

Long TSs of high-volume low-margin retail stores usually present a linear small slope trend. A linear trend can be expressed by a simple linear regression model ([Hyndman and Athanasopoulos, 2018]), such as,

$$m(t) = \beta_0 + \beta_1 t \tag{2.10}$$

with t being the regressor, or predictor variable, β_0 being the intercept, i.e., the value of $m(t)$ when $t = 0$, and β_1 being the slope. In [Taylor and Letham, 2018], it is shown that a seasonal component ($s(t)$) described by a single seasonal period (z) can be captured by a HR over Fourier terms, mathematically,

$$s(t) = \sum_{k=1}^{K} \left(a^{(k)} \cos\left(\frac{2\pi k t}{z}\right) + b^{(k)} \sin\left(\frac{2\pi k t}{z}\right) \right) \tag{2.11}$$

where K is a smoothing factor, and increasing it allows for more fluctuating seasonal patterns, and a and b are the regressors.

Considering the additive model (Equation (2.1)), the two last equations can be combined to capture the trend and seasonality components of a TS. For example, a TS $x(t)$ that is described by a linear trend and two seasonal patterns with periods z_1 and z_2, assuming $K = 1$, may be expressed by

$$x(t) = \beta_0 + \beta_1 t + \beta_2 \cos\left(\frac{2\pi t}{z_1}\right) + \beta_3 \sin\left(\frac{2\pi t}{z_1}\right) + \beta_4 \cos\left(\frac{2\pi t}{z_2}\right) + \beta_5 \sin\left(\frac{2\pi t}{z_2}\right) + \varepsilon(t) \tag{2.12}$$

Then, by removing the approximated components the forecasting model is left to capture only the residual component $\varepsilon(t)$.

An important assumption of this model is that the seasonal periods (z) are already known. As PACF plots are not suitable for TSs with multiple seasonalities, a valid approach for finding out the most relevant seasonal periods is through the application

[10] [Yeo and Johnson, 2000] present the Yeo-Johnson transformation, this is a method that accepts TS with both positive and negative observations.

of a Fast Fourier[11] transform (FFT) on the TS in question. A FFT is an algorithm that computes the Discrete Fourier transform on the TS converting it from its time domain representation to a frequency domain representation.

2.4 Time Series Forecasting

Now that some of the most fundamental concepts of TS modeling and analysis have been discussed, the next step is to widen the knowledge on what forecasting models can be used for TS forecasting. Methods for TS forecasting try to capture patterns that are inherent to the historical data on the presumption that these provide useful information about the future of the variable of interest.

With popular methods such as exponential smoothing and autoregressive moving average (ARMA) families of models, classical statistical methods have been considered the state of the art for TS forecasting for more than 50 years ([De Gooijer and Hyndman, 2006]). For a basic comprehension on how some statistical methods operate for forecasting, descriptions of the mentioned families are overviewed in the next segments.

2.4.1 Classical Models

A basic average method uses the mean of past observations as a forecast. This method implies that all the observations used to compute the mean have the same relevance or weight. However, emphasizing the most recent observations can yield in a more accurate future description. Thus, exponential smoothing methods were created, providing a balanced solution that consists in the application of *exponentially decreasing* weights on older observations. There are many versions of exponential smoothing methods that differ on complexity and in the context that they should be employed, some examples are overviewed in the following items:

- **Single Exponential Smoothing (ETS)** is suitable when the TS manifests neither trend nor seasonality. Considering $\hat{x}(t+1|t)$ to be a forecast at time $t+1$, it can be defined by

$$\hat{x}(t+1|t) = \alpha x(t) + \alpha(1-\alpha)x(t-1) + \alpha(1-\alpha)^2 x(t-2) + ..., \tag{2.13}$$

  where $0 \leq \alpha \leq 1$ is the smoothing parameter, i.e., α controls the dimishing of the weights. The smaller α is the more weight is given to older observations (Figure 2.4).

- An extension of single ETS is **Double ETS (Holt's linear trend method)** ([Holt, 1957]), this method is appropriate when a trend is present. It involves two smoothing

[11] Such analysis should not be conducted if the TS has a highly expressive trend.

equations, prior to the forecast equation, that are given by

$$\text{Level: } l(t) = \alpha x(t) + (1-\alpha)\Big(l(t-1) + b(t-1)\Big), \tag{2.14}$$

$$\text{Trend: } b(t) = \beta\Big(l(t) - l(t-1)\Big) + (1-\beta)b(t-1), \tag{2.15}$$

then, the forecast equation is written as

$$\hat{x}(t+h|t) = l(t) + hb(t) \tag{2.16}$$

where $l(t)$ is an estimate of the level of the series at time t, $b(t)$ represents an estimate of the trend also at time t, $0 \leq \alpha \leq 1$ is the smoothing parameter for the TS level, and $0 \leq \beta \leq 1$ is the smoothing parameter for the trend. The h-step-ahead forecast is the result of a sum between the last estimated level and the last estimated trend value, the latter being previously multiplied by h .

- Lastly, another extension is the **Holt-Winters ETS** ([Winters, 1960] and [Holt, 1957]). This method smoothes the TS when both trend and seasonality are present. There are two adaptations, multiplicative and additive, and one is picked by analyzing the behaviour of the seasonal component. If seasonality is modelled in an additive way then it can be defined by

$$\text{Level: } l(t) = \alpha\Big(x(t) - s(t-h)\Big) + (1-\alpha)\Big(l(t-1) + b(t-1)\Big), \tag{2.17}$$

$$\text{Trend: } b(t) = \beta\Big(l(t) - l(t-1)\Big) + (1-\beta)b(t-1), \tag{2.18}$$

$$\text{Seasonal: } s(t) = \gamma\Big(x(t) - l(t)\Big) + (1-\gamma)s(t-m), \tag{2.19}$$

then, the forecast is a result of

$$\hat{x}(t+h|t) = l(t) + b(t)h + s(t-m+h), \tag{2.20}$$

where m represents the seasonal period, m the number of periods which are forecasted ahead, and now $s(t)$ and γ are included which represent, respectively, the seasonal component and the seasonal smoothing parameter.

The exponential smoothing family of methods possesses many more variations differing in the context to use and complexity. If the reader wishes to know more about these concepts, [Gardner, 1985] is referenced.

Along with exponential smoothing methods, ARMA models are one of the most used approaches for statistical TS forecasting, with variations providing coverage on different problems, for instance on multivariate and non-stationary TS. One of the most popular versions of ARMA models is the ARIMA model. ARIMA method ([Box *et al.*, 1976]) combines differencing, to achieve stationarity, and two models, an autoregressive (AR) and a moving average (MA) model (these two models are the foundation of ARMA models).

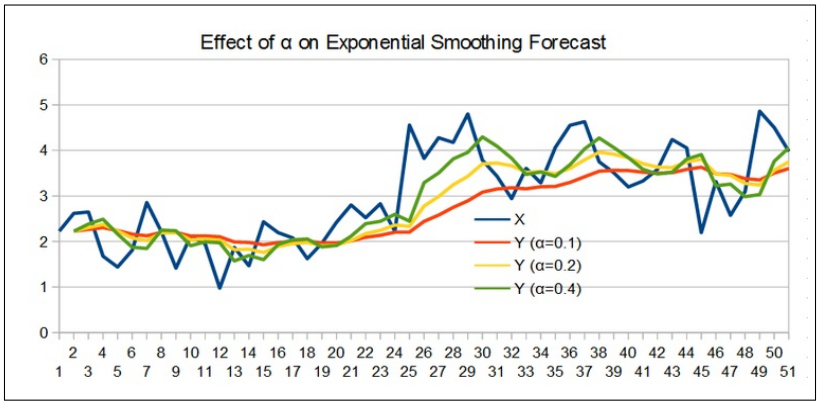

Figure 2.4: Simple exponential smoothing with three different values for α. In the smoothed signals, it is easy to observe that when $\alpha = 0.1$ there is more emphasis on older observations. **Source**: [*Forecasting - Exponential Smoothing* 2017]

As such, an ARIMA model expects three parameters:

- An order of differencing d (subsection 2.3.4);
- The autoregressive order p, for the autoregressive model (Equation (2.5));
- The moving average order q, for the moving average model (not to be confused with the moving average smoothing method). As opposed to the AR model, that uses past observations as explanatory variables, the MA focuses on past errors,

$$x(t) = c + \varepsilon(t) + \theta_1\varepsilon(t-1) + \theta_2\varepsilon(t-2) + ... + \theta_q\varepsilon(t-q), \tag{2.21}$$

  where $\varepsilon(t), ..., \varepsilon(t-q)$ represent the white noise error terms and $\theta_1, ..., \theta q$ represent the model parameters (similar form to that of Equation (2.5)). Then, the combination is called the ARIMA(p, d, q) model, which is given by,

$$x'(t) = c + \phi_1 x'(t-1) + ... + \phi_p x'(t-p) + \theta_1\varepsilon(t-1) + ... + \theta_q\varepsilon(t-q) + \varepsilon(t), \tag{2.22}$$

  where $x'(t)$ represents a TS $x(t)$ that could have been differenced once or more and ϕ are the regression coefficients.

In the past, the estimation of the optimal orders p and q would have proven to be a challenging task, however, with the creation and maintenance of statistical code libraries[12] that difficulty was reduced, since automatic methods that work to that end are included there.

[12] For example, auto.arima (forecast package) for the R language and the Python equivalent, pmdarima ([Hyndman and Khandakar, 2008] and [Smith *et al.*, 2017]).

2.5 Forecast Error Metrics

Forecast evaluation consists in assessing the suitability or quality of a forecasting method for a particular dataset. The evaluation of a quantitative forecast is usually treated as a measure of how accurate or how well a model fits or reproduces the data.

In subsection 2.2.1, it was stated that in TS forecasting the *"(...) objective is to generate a forecast function that results in minimum mean squared errors between actual values and forecasted values (...)"*. Although this being fundamentally true, in certain situations such statement can prove to be an oversimplification. In various cases, applying more than one statistical error metric is vital to obtain different kinds of information. Thus, this subsection details various metrics that are commonly used in TS forecasting. Before detailing some of the most popular metrics, defining forecast error is important as it is their basis, it can formally be written as,

$$e(t) = x(t) - \hat{x}(t), \tag{2.23}$$

with $x(t)$ being the actual observation and $\hat{x}(t)$ the forecast, both at time t. It is worth noting that this definition is allocated in a single-step forecast scenario, meaning that the forecasted value $\hat{x}(t)$ was built on a number of observations that are previous to $x(t)$. Thus, $e(t)$ is described as a one-step forecast error. However, forecasting methods usually produce forecasts with more than one step, i.e., feature a forecasting horizon of $H > 1$. Therefore, to form a valid approach, the following standard statistical accuracy metrics can be used:

$$\text{Mean Error } = \frac{1}{H}\sum_{t=1}^{H} e(t) \tag{2.24}$$

$$\text{Mean Absolute Error } = \frac{1}{H}\sum_{t=1}^{H} |e(t)| \tag{2.25}$$

$$\text{Mean Squared Error } = \frac{1}{H}\sum_{t=1}^{H} e(t)^2 \tag{2.26}$$

$$\text{Root Mean Squared Error } = \sqrt{\frac{1}{H}\sum_{t=1}^{H} e(t)^2} \tag{2.27}$$

The mean forecast error (Equation (2.24)) is simply the average of the errors (Equation (2.23)) that were calculated for all H periods. Usually, this metric is not very useful since it does not provide a clear error description when there are negative and positive values (they offset each other). However, it does provide a suggestion of the model being either, consistently, over forecasting or under forecasting, and because of that, the mean forecast error is also referred to as forecast bias. The remaining methods, however, remove this last problem by moulding the error to a positive value. Before averaging, the MAE metric turns the error into a positive value by taking its absolute value, while both

MSE and RMSE metrics conduct a transformation through squaring.

A problem that is common to the observed metrics is that they are not unit-free, i.e., they depend on the scale of the given data. Therefore, alternative metrics are required to render possible the comparison of forecast performances across TSs with different units. For that, metrics that include percentage or relative errors are the usual approach, the percentage error can be defined by

$$p(t) = \Big(\frac{x(t) - \hat{x(t)}}{x(t)}\Big) \times 100, \tag{2.28}$$

and some of metrics that use percentage errors are:

$$\text{Mean Percentage Error } = \frac{1}{H}\sum_{t=1}^{H} p(t) \tag{2.29}$$

$$\text{Mean Absolute Percentage Error } = \frac{1}{H}\sum_{t=1}^{H} |p(t)| \tag{2.30}$$

$$\text{Symmetric Mean Absolute Percentage Error } = \frac{1}{H}\sum_{t=1}^{H} \frac{|\hat{x}(t) - x(t)|}{\Big(|x(t)| + |\hat{x}(t)|\Big)/2} \tag{2.31}$$

The mean percentage error (Equation (2.29)) maintains the same problems of the mean error metric so it is usually not employed. Another option is MAPE. It is highly popular and of trivial intepretation, however, it presents a significant drawback—asymmetry. As a result of that, MAPE punishes negative errors more than positive errors, i.e., MAPE will provide better results for models that under forecast rather than for those that over forecast. In an attempt to correct this drawback, sMAPE was created (Equation (2.31)). It fixes the shortcoming by establishing a lower bound (0%) and a upper bound (200%), however, on doing this it paves the way for other problems:

- It his highly unstable when both the predicted $\hat{x}(t)$ and the actual values $x(t)$ are close to zero;
- It raises doubts on its interpretation since it presents a range of 0% to 200%;
- When $\hat{x}(t) = 0$ or $x(t) = 0$, sMAPE will always reach the upper boundary of 200%;
- Even if more delicate, another asymmetry caused by the denominator is introduced.

As such, even though its great popularity, in [Hyndman and Koehler, 2006] it is recommended that it should not be used.

Another disadvantage that is common to all percentage error metrics is the **meaningful zero**[13] problem. Frequently, retail stores show sales or client flow

[13] Percentage error metrics assume that all the variables being evaluated have the true zero characteristic, i.e., are included in a ratio scale measurement. Temperature is an example of a variable that does not possess a true zero and, as such, it cannot be evaluated by percentage error metrics. In contrast, weight is an example of variable that can have a true zero, as zero weight means no weight.

intermittence, for example, during the early morning or at lunch time. This is even more noticeable in high-frequency data, such as half-hourly TSs. This creates a major problem when using MAPE and sMAPE since observations that have a value of zero can make the error rise unrealistically. Thus, the recent unit-free mean arctangent absolute percentage error (MAAPE) metric was created ([Kim and Kim, 2016]). Fundamentally, it has the same behaviour of MAPE except for situations where the real values are zero. Instead of having the error lead to infinity it leads to $\frac{\pi}{2}$ since an arctangent function is used. Formally,

$$\text{MAAPE} = \frac{1}{H}\sum_{t=1}^{H}\arctan(|p(t)|) \tag{2.32}$$

Concluding, it is not an easy task for a practitioner to decide which metrics should be used and that is one of the reasons why sometimes multiple metrics are calculated. The mean error and the mean percentage error metrics are rarely applied as the information they provide is, as mentioned, disposable. Also, optimizing models with MAE means that the forecasts will be targeting the median, therefore, will have reduced sensitivity to outliers. In contrast, if optimization is done with RMSE then forecasts will be targeting the average of the TS and, therefore, will be more sensible to outliers ([Vandeput, 2018]).

2.6 Summary

With data storage and computational resources showing, over the years, a decline in price and a greater availabity, aditionally, with the size of TSs increasing and companies demonstrating a need for better performing forecasting methods, ML models, especially DL models, have recently drawn a great deal of attention. However, there are still doubts regarding the reliability and consistency of these models. Some authors state that ML methods overperform ([Werbos, 1988]) or at the very least can be seen as serious contestants to statistical models ([Bontempi *et al.*, 2013] and [Palit and Popovic, 2005]), while others say that ML models perform worse or require further research to reach the level of classical models ([Makridakis *et al.*, 2018a]). Yet, some of the works that state that ML methods lack in performance have obtained their conclusions through forecasting competitions ([Makridakis and Hibon, 2000], [Makridakis *et al.*, 2018b]) and not through the application of these methods in real scenarios, where practitioners have to frequently deal with crude and irregular data. In addition, linear models have been characterized as not suitable to real-world applications ([De Gooijer and Hyndman, 2006]). Thus, further research on DL models applied to TS forecasting is advantageous, even more when applied to real world scenarios such as the retail field.

In this chapter, classical models for TS analysis and forecasting were detailed. Next, this work will shift its focus entirely to a DL genre of models, specifically, ANNs.

Chapter 3

Deep Learning and Artificial Neural Networks

This chapter details the inner workings of ANNs. Firstly, a simpler representation of a neuron (also called neural unit) such as the perceptron is unveiled. Then, more advanced models, particularly, the fully-connected, convolutional and recurrent neural networks are exposed. These notions are explained while being lightly allocated to the TS forecasting context.

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are computing paradigms biologically-inspired by the processes of the brain. As such, ANNs have the ability to learn from a complex set of observations to answer specific problems. As mentioned before, ANNs can capture nonlinear relationships in the data and due to this fact are applied in many fields, for instance:

- Cybersecurity, where classification tasks are employed in order to distinguish harmless use from malicious attacks;
- Speech Recognition, for example, to enable devices to answer voice commands through the translation of spoken language into text;
- Pattern Recognition, for example, to impose identity on handwriting or through images.

Nevertheless, the question of what exactly is an ANN arises. Primarily, an ANN is a collection of connected neurons. Calculations flow through the connections that exist between neurons, the latter being organized in layers. The various predictors or input variables form the input layer and results are obtained from the output layer. To facilitate the explanation, a very simple representation of a neural network — the **perceptron**, or single layer perceptron, is provided in Figure 3.1.

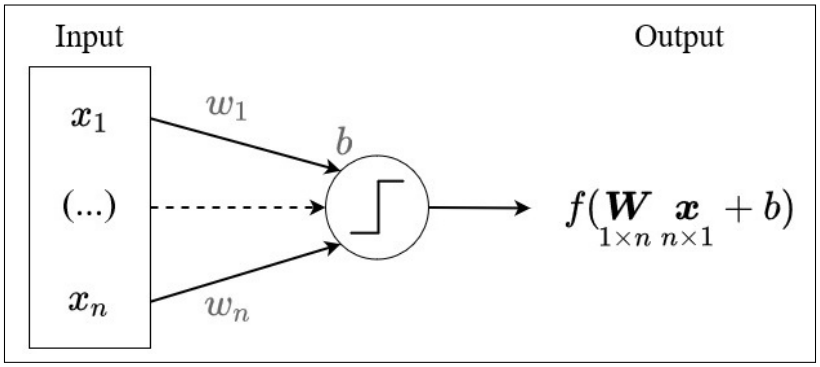

Figure 3.1: A representation of a single layer perceptron, first introduced in [Rosenblatt, 1957]. With an input layer with n units and an output layer with a single perceptron, containing **bias**, it generates an output through an activation function. Each connection between the input layer and the perceptron has an associated weight.

As demonstrated in Figure 3.1, the output of the perceptron is given by the equation ([McCulloch and Pitts, 1943]),

$$y = f(\underset{1\times n}{\boldsymbol{W}}\ \underset{n\times 1}{\boldsymbol{x}} + b), \tag{3.1}$$

where $\boldsymbol{x}$, that represents the input features $(x_1, ..., x_n)^T$, processes a dot product with $\boldsymbol{W}$, the one row matrix that represents the weights $(w_1, ..., w_n)$ for each connection, which is then added to another parameter called bias. Function f is a nonlinear function that is traditional for perceptrons, it is called the **Heaviside step function** (or unit step function), however, in the context of ANNs it is generally referred to as an **activation function**. Trivially, the unit step function can be written as

$$f(a) = \begin{cases} 1, & a \geq 0 \\ 0, & a < 0 \end{cases} \tag{3.2}$$

With all this considered, the single layer perceptron can be considered a neural network, however, its usefulness is questioned in [Minsky and Papert, 1969] where it was proved that this model cannot reproduce a XOR function, indicating that the single layer perceptron is not able to solve nonlinear relationships. Since then, the perceptron is simply known as a linear binary classifier. Still, these findings served as a foundation to the discovery of **learning algorithms**. These are algorithms that automatically tune the weights and biases, the learnable parameters of a network.

Subsequent work in the 1980s resulted in a new neural network, the first network to have some depth[1] (this is explained later) was finally able to approximate a XOR operator. It is called the Multilayer Perceptron (MLP) and it consists of a combination of neurons in some ways similar to perceptrons, as represented in Figure 3.2. Knowing this, it is worth mentioning that the modern perception of the term MLP can often be confusing. Today's perception establishes that the term perceptron is not employed in the strictest

[1] Considered shallow by today's standards.

sense of the word since the neurons in the MLP are different than the observed traditional perceptron. The only difference is that they are applied with different activation functions. The primary reason for this is that the unit step function does not work well in an ANN learning process[2] phase. Therefore, a more general term can be used—**Fully-connected Neural Network** (FNN).

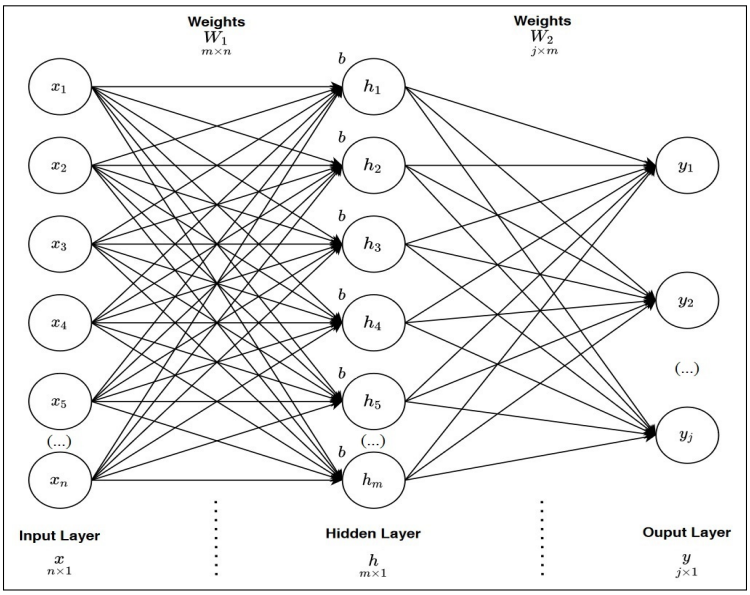

Figure 3.2: The architecture of a MLP. Now, the ANN includes a hidden layer with m perceptrons, also usually called neurons or hidden units.

By observing Figure 3.2 and transposing what was learned in Equation (3.1), the computation of the hidden neurons $h_1, ..., h_m$, i.e. the hidden layer, can be given as a vector,

$$\underset{m\times 1}{\boldsymbol{h}} = \phi\left(\underset{m\times n}{\boldsymbol{W}_1}\ \underset{n\times 1}{\boldsymbol{x}} + \underset{m\times 1}{\boldsymbol{b}_1}\right) \tag{3.3}$$

where $\boldsymbol{h}$ is a vector with each of its values corresponding to a hidden unit's output and ϕ represents any activation function.

It was discovered that the insertion of hidden layers (one or more) in combination with the presence of nonlinear activation functions is what provides the ability of an ANN to solve nonlinearly separable problems. If an ANN has a high a number of hidden layers then it is considered a Deep Neural Network DNN. The inclusion of hidden layers helped with the striking discovery (for instance, [Cybenko, 1989]) that ANNs can be universal approximators, i.e., any ANN that has at the very least some degree of **depth** (one or more hidden layers) can approximate any function. This characteristic is one of the main reasons as to why DL has been in demand in so many fields.

[2] Specifically, the derivates of HS are always zero except at $k = 0$ where it is non-differentiable, this makes it it near impossible to optimize a model using gradient methods since the weights rarely change in the updating.

3.2 Learning Designs

There are various learning designs within ML. Picking one, usually depends on the task at hand and the type of data (input and output) that the practitioner has to work on. The techniques that are most recurrent in literature are:

- **Supervised Learning**;
- **Unsupervised Learning**;
- **Reinforcement Learning**.

Unsupervised learning, is a process that is used when the data is not labelled or categorized, i.e., the data does not provide the outputs that are to be predicted by the model with the given inputs. With this, the usual tasks are to enforce the models to identify and extract underlying structures in the data. A common unsupervised learning task is **clustering**. Clustering is the process of discovering data groups through found correlations between the various data points. The evaluation of an unsupervised method will very much depend on the reason as to why the practitioner is applying it.

Reinforcement Learning algorithms have their most distinguishable characteristic being the fact that they are not used on fixed datases. This means that reinforcement learning, which is usually applied to create software agents, takes place on interactive datasets, i.e. the agents try to learn the best course of actions through feedback to maximize a cumulative reward. Games[3] are a prime example of where reinforcement learning is used (e.g. a chess agent).

3.3 Supervised Learning

Supervised learning, as the term implies, is applied when the data provides supervision through labelling, meaning that inputs have the associated labels or target predictions. Hence, the error is computable which possibilitates an iterative optimization (error minimization) through a cost/loss function,

$$J(\boldsymbol{W}) = cost(\boldsymbol{y}, \hat{\boldsymbol{y}}) = cost(\boldsymbol{y}, \hat{F}(\boldsymbol{x}, \boldsymbol{W})), \tag{3.4}$$

where y is the target value and $\hat{y}$ the predicted output obtained by an ANN approximated function $\hat{F}(\boldsymbol{x}, \boldsymbol{W})$. This iterative approach represents the ANNs' training, in which the goal is to find the values of the weights $\boldsymbol{W}$ (and biases) that correspond to the minimum of $J(\boldsymbol{W})$. Depending on the problem that is being faced, the dimensions of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\boldsymbol{x}$ are subject to change.

Supervised learning can be used in various examples, for instance, an image recognition system to distinguish cats from dogs or even various animals. In such problem, each image

[3] Google devised an agent, AlphaGo, that was capable of defeating a Go world champion: https://deepmind.com/research/case-studies/alphago-the-story-so-far.

and the corresponding label of the animal would represent each data sample. Another example is real estate price estimation. In such example, the attributes of the property (e.g. floors, number of bedrooms, commercial or not) are the input features whilst the output is the price of the property.

The two presented examples picture the main types of supervised learning, the first one is an example of categorical **classification** while the second one is a **regression** problem. The difference is the output that the predictive model is supposed to provide. In a classification problem a model's output corresponds to a categorical value (class label), on the other hand, in a regression problem it represents a real numerical value.

In the context of TS forecasting, classification approaches can be found in the literature and in some cases have shown to be advantageous. An example is the work in [Huber and Stuckenschmidt, 2020][4], where the classification methods were considered to be at least as good as the regression approach used. Nevertheless, the same authors also state that regression methods are employed more frequently and, as such, regression is considered the natural approach. For this reason, the models in this work will only be treating TS forecasting as a **supervised learning regression task**. In order that ANNs perform approximations with regression modelling, usually, the output layers linearly produce raw values, i.e., the weighted inputs plus biases of the output layer do not face nonlinear activations, that are considered to be the forecasts. Consequently, they can be directly used by cost functions. A popular example of a cost function is the MSE, which is also used for forecast evaluation (section 2.5).

3.3.1 Backpropagation and Gradient Descent Optimizers

ANNs usually learn through an iteration approach that attempts to minimize the errors produced by the cost function. To achieve this, not many alternatives have been known to surpass the famous combination of **backpropagation** algorithms with **gradient descent** optimizers. Two stages compose an iteration in the learning, or training, setting of an ANN:

- The **Forward pass**, represents the computations (the weighted input sums and activations in Equation (3.3)) traversing from the input layer to the output layer, ending with the cost function calculation;

- The **Backward pass**, which is when the backpropagation algorithm and weight updates are performed.

In the backward pass, the backpropagation algorithm is used to compute the gradient of the cost function, which is calculated by applying partial derivatives with respect to the weights (and bias), formally,

$$\nabla J(\boldsymbol{W}_i) = \frac{\partial J}{\partial \boldsymbol{W}_i} \tag{3.5}$$

[4] The classification modelling was processed through the technique of data binning.

and, in order to calculate this equation, the backpropagation algorithm makes recursive use of the mathematical **chain rule**.

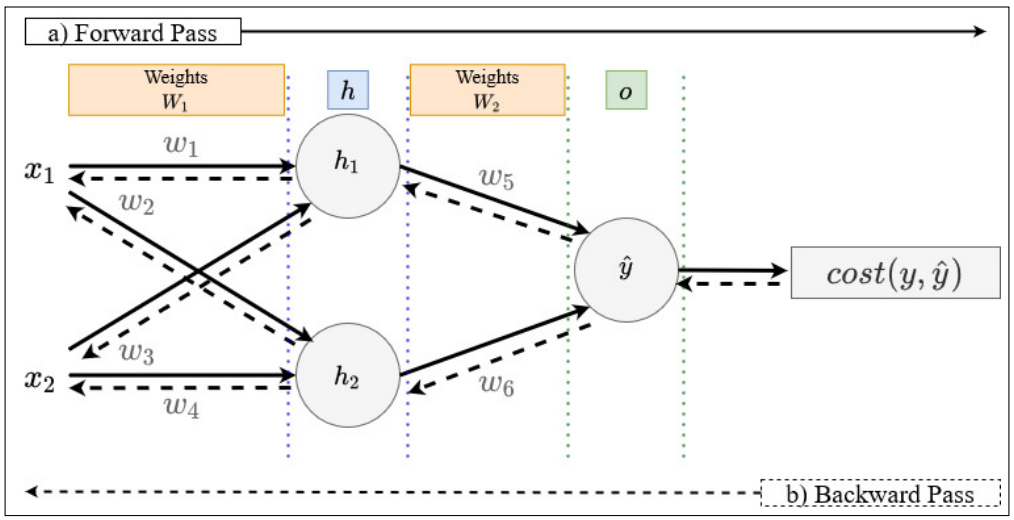

Figure 3.3: Overview of the backpropagation process. a) The forward pass consists in the activations h_1, h_2 and, assuming that the output unit is activated, $\hat{y}$ that are processed with the inputs x_1 and x_2. The cost between the final output $\hat{y}$ and the target value y is then calculated. b) In the backward pass the cost is backpropagated to calculate the gradients with respect to the weights $w_1, ...w_6$. **Adapted from**:[Güneş Baydin *et al.*, 2018]

Through the observation of the FNN in Figure 3.3, a simple matrix-based backpropagation example that exposes the chain rule can be set up ([Nielsen, 2015]). First, the network's operations (disregarding the biases) in the forward pass have to be separated,

$$\underset{2\times1}{\boldsymbol{h}_{in}} = \underset{2\times2}{\boldsymbol{W}_1}\underset{2\times1}{\boldsymbol{x}}$$

$$\underset{2\times1}{\boldsymbol{h}_{out}} = \phi(\underset{2\times1}{\boldsymbol{h}_{in}})$$

$$\underset{1\times2}{o_{in}} = \underset{1\times2}{\boldsymbol{W}_2}\underset{2\times1}{\boldsymbol{h}_{out}}$$

$$o_{out} = \phi(o_{in})$$

the $\boldsymbol{h}$ designates the calculations that refer to the hidden layer and o to the output layer, *"in"* corresponds to weighted inputs and *"out"* to activations. Secondly, in order to know how the cost function is affected by the weights, the error that is backpropagated from the output layer ($l = 2$) is introduced as,

$$\delta_2 = \frac{\partial J}{\partial o_{out}} \odot \phi'(o_{in}) \tag{3.6}$$

where $\odot$ refers to an element-wise product, the term on the left is the rate of change of J with respect to the activations and the term on right is the rate of change of ϕ at the

weighted input o_{in}. Then, the error for the hidden layer ($l = 1$) is given as

$$\boldsymbol{\delta}_1 = \left(\boldsymbol{W}_2^T \, \delta_2\right) \odot \phi'(\boldsymbol{h}_{in}) \tag{3.7}$$

where the transpose means providing, backwards, a measure of the error at the activations and the element-wise product means giving the error δ_1 at the weighted input both to the hidden layer. Were the example network to have more hidden layers and their $\boldsymbol{\delta}$ would have similar form.

With the combination of the two previous equations it is possible to calculate the originally wanted gradients. The reasoning for $\nabla J(\boldsymbol{W}_2)$, which corresponds to the gradients with respect to w_5 and w_6, is described by

$$\frac{\partial J}{\boldsymbol{W}_2} = \delta_2 \, \boldsymbol{h}_{out}^T \tag{3.8}$$

and, for $\nabla J(\boldsymbol{W}_1)$, which corresponds to the gradients with respect to w_1, w_2, w_3 and w_4, is given by

$$\frac{\partial J}{\boldsymbol{W}_1} = \boldsymbol{\delta_1} \, \boldsymbol{x}^T \tag{3.9}$$

Following the calculation of the partial derivatives, the gradients are then used by an optimizer of choice to update the weights. **Gradient descent** optimizers are commonly used for this, presenting various options. A plain gradient descent optimizer, called batch gradient descent, performs the weight updates as defined in the following equation,

$$\boldsymbol{W}_l \rightarrow \boldsymbol{W}_l^{(new)} = \boldsymbol{W}_l - \eta \nabla J(\boldsymbol{W}_l) \tag{3.10}$$

where $\boldsymbol{W}$ represents the old weights and $\boldsymbol{W}^{(new)}$ the new weights. Called the learning rate, or step size, η determines the size of the steps that are to reach the cost function's local minimum, i.e., controls how much influence each update has on the weights.

Many options for the gradient descent optimizer were developed, for example, the popular stochastic gradient descent (SGD) and Adaptive Moment Estimation (ADAM). Details on many of these methods are described in [Ruder, 2016].

3.3.2 Regularization Techniques

Commonly, when the training of a DL model is processed there is access to a training set and, as mentioned, there is a loss measure and the objective is to minimize it. However, a separate evaluation is done on a correlated test set that is referred to as test or **generalization** error. An essential issue that should be considered when developing DL algorithms is making sure that the models have good performances not only on training data but also on brand new inputs. The ability to do that implies that the model is capable of generalizing. When a model performs badly on both training and new inputs then it is **underfitting**. Underfitting is represented by contextually high loss measures meaning that the model does not learn. Typically, it is caused by the training data being

small or low-quality (e.g. irregular TSs), or the model does not have sufficient **capacity** to approximate the target function. A model's capacity is its ability to approximate a wide range of functions. This last aspect is important as it is directly related to a more complex problem: **overfitting**. Overfitting happens when the DL model has too much capacity. Because of that, it tends to fixate on specific traits from the training set too well and while it achieves small training errors it has trouble expressing knowledge with new inputs, i.e., it is unable to generalize.

Underfitting can be simply resolved by improving the training data, however, an overfitting issue may be trickier to resolve due to the already low training errors. A way to reduce the overfitting problem is through the application of regularization techniques. Regularization is defined by any modification that is applied to a learning algorithm with intent to reduce its generalization error but not its training error. Popular regularization techniques consist in limiting the capacity of the model by applying a penalty to its weights $\Omega(\boldsymbol{W})$. By redefining Equation (3.4), it is given by

$$J(\boldsymbol{W}) = cost(y_i, \hat{F}(x_i, \boldsymbol{W})) + \alpha\Omega(\boldsymbol{W}) \tag{3.11}$$

where α is an "importance" coefficient as it controls the strength of the penalty. A popular norm penalty is the ***L2*** term, formally,

$$J(\boldsymbol{W}) = cost(y_i, \hat{F}(x_i, \boldsymbol{W})) + \alpha \sum_{w \in \boldsymbol{W}}^{|\boldsymbol{W}|} w^2, \tag{3.12}$$

this way, on each learning step, the network is forced to reduce the size of the parameters and, consequently, affect the weight updates.

Another way to handle overfitting is through **early stopping**. This technique consists in monitoring the loss during training. If the loss increases for the next predefined number of iterations (*patience*) the training phase is halted and the version of the model that had the lowest loss is chosen.

Lastly, another extremely popular technique is called **dropout**. Proposed in [Hinton *et al.*, 2012], it primarily consists in ignoring or deactivating neurons in a layer with a specified probability *prob* that derives from a Bernoulli distribution. Dropout is applied during training and its effect is as if there is a different network in each iteration. As a matter of fact, since every neuron can be in an active or inactivate state there can be 2^N different network combinations. By deactivating a unit it is meant as temporarily, i.e., in that iteration, removing it from the network along with all its input and output connections (Figure 3.4).

3.4 Convolutional Neural Networks

Convolutional neural networks (CNNs), also called ConvNets, have been applied to many problems but are especially famous for their performance in problems of image

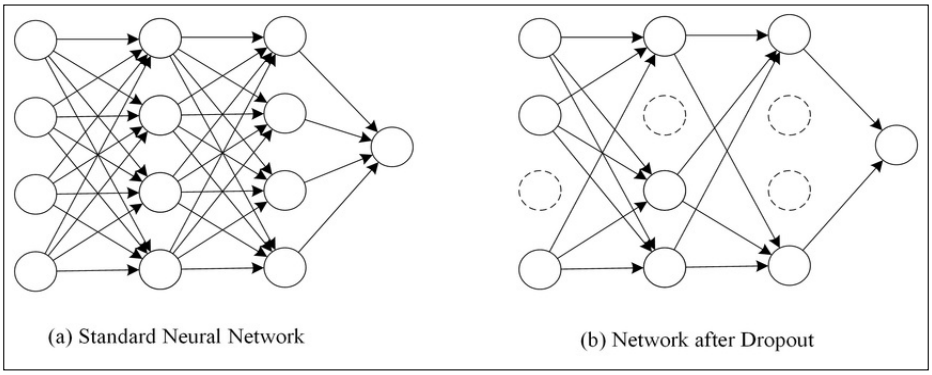

Figure 3.4: Comparison between a standard FNN and a FNN with dropout. **Source**: [Khalifa and Frigui, 2016]

classification.

Even though the creation of the CNN can be traced back to the research made in [Fukushima, 1980], the known modern architecture was first proposed in [LeCun *et al.*, 1998]. In comparison to the MLP, the architecture of a CNN enables a faster training by reducing the trainable parameters, this results in lighter computational requirements for deeper models. To accomplish this, CNNs have three main characteristics:

- Local receptive fields;
- Shared weights;
- Pooling.

In addition to these characteristics, the architecture of a CNN usually presents three types of layers: **convolutional** layers followed by ReLU activations, **pooling** layers and the already seen **fully-connected** layers (Figure 3.5).

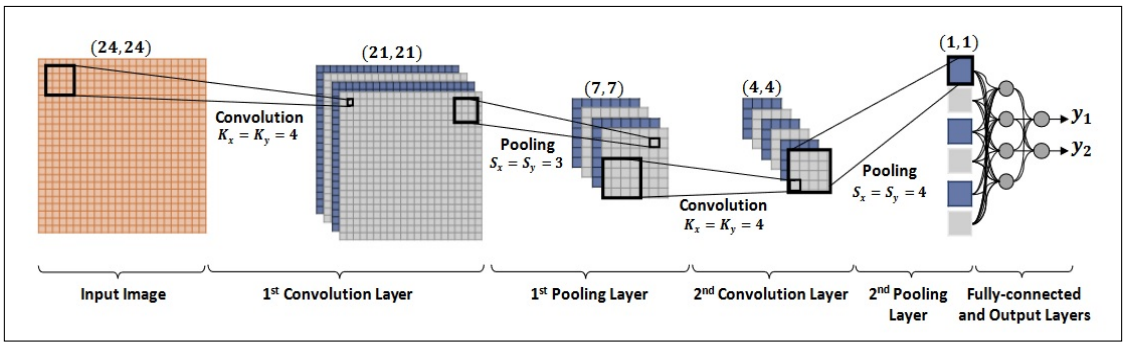

Figure 3.5: The generic architecture of a CNN. **Source**: [Kiranyaz *et al.*, 2019]

Just like MLPs, every **convolutional** layer possesses two learnable parameters, a set of **filters** (weights) and a **bias** for each filter. Their input is expected to be a *3D* input, i.e., with width (W), height (H) and depth[5] (D). A distinguishable characteristic of CNNs is **parameter sharing**, this means that many input units are able to share the

[5] Depth is usually the number of colour channels (e.g. RGB) in an image.

same weights in the processing operations. More specifically, the same weights and bias are used for each *2D* **depth slice** of the input. With this, the training is much faster than in MLPs since the number of trainable parameters is greatly reduced. Also, unlike in fully-connected layers, hidden neurons are not connected to every input unit. Instead, each neuron presents connections only to its designated **local receptive field**. These fields' shapes are equal to the preconfigured filters.

Fundamentally, a convolution operation performs dot products between the filters (weights) and the neurons' input receptive fields (Figure 3.6). Its resulting objects are input abstractions frequently called convolved features or feature maps, there are four parameters that affect them:

- The **number of filters** (F), since weights are randomly initialized they are able to capture different features from the input;
- The **size of the filters** (K);
- The **stride** (S), which dictates the size of each slide or shift the filters are to perform at each step on the receptive fields;
- Sometimes when applying the filters on the edges of the receptive fields, it is fitting to pad zeros around them, as such, the **padding** parameter (P) can be configured. Padding can ensure the output has the same shape as the input. In addition, it eases the extraction of relevant features since the edges are given more valuable interaction with the filters.

With these parameters, the dimensions ($W_{out} \times H_{out} \times D_{out}$) of an outputted volume (e.g. a set of feature maps) from a convolutional layer are given by

$$\begin{aligned} W_{out} &= \frac{(W_{in} - K + 2P)}{S + 1} \\ H_{out} &= \frac{(H_{in} - K + 2P)}{S + 1} \\ D_{out} &= F \end{aligned} \tag{3.13}$$

when a volume with dimensions $W_{in} \times H_{in} \times D_{in}$[6] is inputted.

Immediately after the built feature maps pass through the ReLU activation function, the end of the convolutional layer is reached, usually, the process moves on to what is known as a **pooling** layer. In a basic sense, pooling is an operation that simplifies and spatially reduces the information outputted from the convolutional layers. This is processed through a regional neighborhood operation in the feature maps. More specifically, an aggregation operation (e.g. mean, max or minimum) on the local receptive field is computed.

Pooling operations, as in Figure 3.7, also contribute with feature maps that are then fed to fully-connected layers. These layers are identical to the hidden layers of the traditional

[6] Or C if it corresponds to the original input.

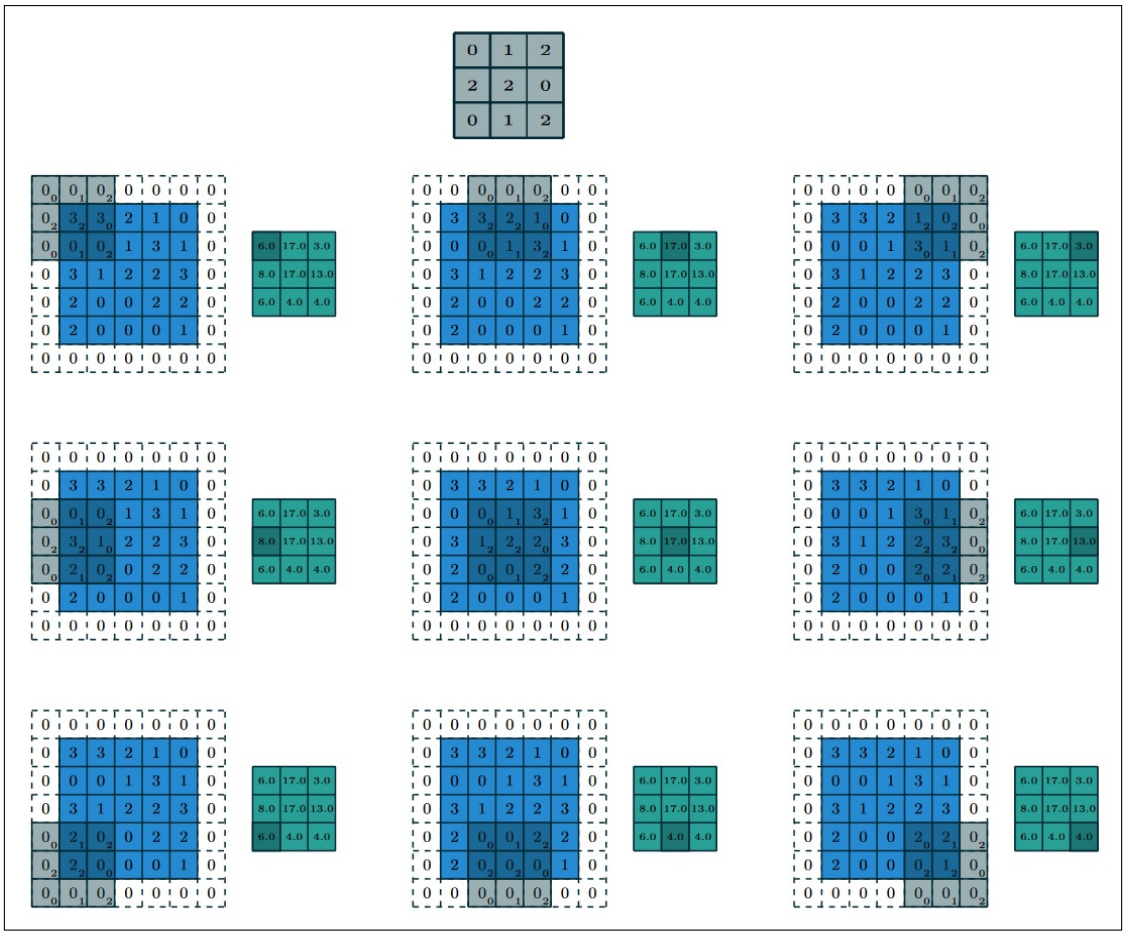

Figure 3.6: A *2D*-convolution with **padding** $P = 1$. The feature map does not have the same size as the input due to the **stride** being $S = 2$. The first matrix (gray) is the kernel, the blue matrix represents the input and the green matrix is the resulting feature map. The gray color over the blue matrix represents the **receptive field**. **Source**: [Dumoulin and Visin, 2016]

MLP. In addition, in contrast to convolutional layers, pooling layers do not have trainable weights or bias. It is also worth noting that pooling layers have recently been considered optional and some authors propose their disposal completely ([Springenberg *et al.*, 2015]). Nevertheless, that debate is not to be considered in this research.

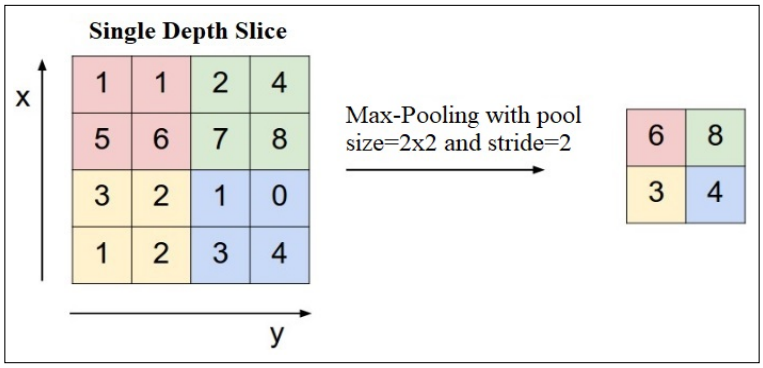

Figure 3.7: An example of max-pooling.

With all these concepts presented, the reader might question, in comparison to MLPs if there are changes in the learning phase of a CNN. And, even though it does present some differences, it is still very similar to the outlined learning algorithm (subsection 3.3.1). A detailed description of the backpropagation algorithm performed in CNNs can be found in [Kiranyaz *et al.*, 2016].

Despite the success that various CNN architectures[7] have been displaying over the years, the interpretability of the learned features has often be questioned and even criticized ([Qin *et al.*, 2018]). To better understand the intuition behind CNNs, several vizualition methods have been created ([Han *et al.*, 2016], [Sivic and Zisserman, 2003] and [Wei *et al.*, 2015] are illustrative cases). Figure 3.8 shows an example of what the filters might learn in each convolutional layer. In the first two layers the captured features were small sections, such as edges and corners. In the third layer, a texture pattern was captured. In the fourth and fifth layer, greater parts and almost even the entire class were reproduced.

3.5 Recurrent Neural Networks

The Recurrent Neural Network (RNN) is a distinctive type of ANN that was designed to model problems that have in their essence a sequential or time-related aspect. RNNs are able to "memorize" by preserving an internal state using a number of **feedback loops**. These loops occur in the neural units of the network, thus, they are called recurrent neural units. In addition to the standard connections a neuron has, a recurrent unit has a new connection leading to itself, that connection makes possible using past outputs as inputs for the calculation of new outputs (Figure 3.9). Therefore, the length of the input

[7] For example, AlexNet ([Krizhevsky *et al.*, 2017]), ZF Net by [Zeiler and Fergus, 2014] and GoogLeNet [Szegedy *et al.*, 2015] all won the ImageNet Large Scale Visual Recognition Competition (ILSVRC).

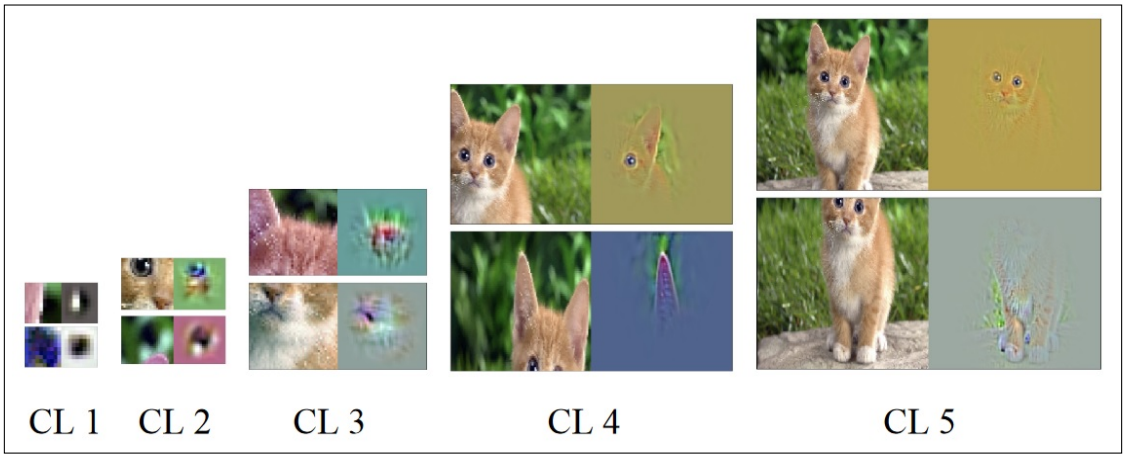

Figure 3.8: Visualization of two filters, picked randomly, in five convolutional layers of a CNN. This depiction was accomplished with a deconvolutional network ([Zeiler *et al.*, 2010]) based method ([Ranzato *et al.*, 2006]). **Source**: [Qin *et al.*, 2018]

sequences now affect the approximation process. Because of that, RNNs can operate with sequences of vectors which makes them able to solve more types of sequence problems (Figure 3.10). This contrasts with the previous architectures since, in order to perform an approximation, MLPs and CNNs follow a fixed number of computational steps that only depends on the number of layers.

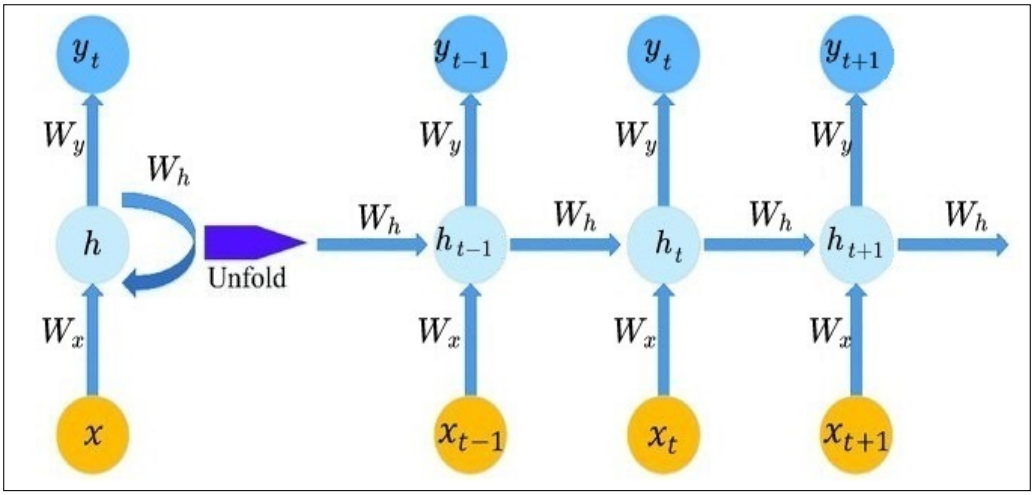

Figure 3.9: The reccurent neuron on the left can be represented by the units on the right if its processes are unfolded through time. x and y are the input and output, respectively, while W_x, W_h and W_y are the temporally shared weights for each connection. **Adapted from**: [Bao *et al.*, 2017]

As seen in Figure 3.10, RNNs combine the obtained input vector with their internal state and a learnable function. In a programming sense, this can be viewed as executing a fixed program with defined inputs and internal variables. Thus, RNNs describe programs and not just functions meaning that they are arbitrary program approximators, in other words, **Turing-Complete** ([Siegelmann, 1995]).

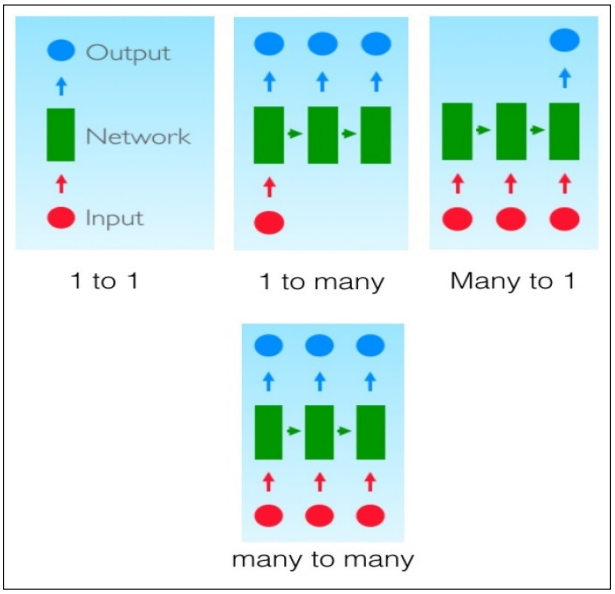

Figure 3.10: The green rectangles (captioned as networks) hold the RNNs' internal state vectors. The one-to-one sequence problem is the conventional way of solving problems with MLPs and CNNs, a fixed-sized vector input results in a fixed-sized vector output. Since the remaining possess sequences of vectors in at least one of the sections they can only be modelled by RNN architectures. **Source**: [Mosconi *et al.*, 2019]

In similar fashion of Equation (3.3), for each processing step t the activations of a layer of recurrent units, $\boldsymbol{h_t}$, can be expressed as,

$$\underset{m\times 1}{\boldsymbol{h_t}} = \begin{cases} \mathbf{0} & \text{if t=0,} \\ \phi_1\left(\underset{m\times n}{\boldsymbol{W}_x}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}_h}\ \underset{m\times 1}{\boldsymbol{h}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}_1}\right) & \textit{otherwise} \end{cases} \tag{3.14}$$

where the weights $\boldsymbol{W}_x$ and $\boldsymbol{W}_h$, which are shared in time, are associated with the inputs $\boldsymbol{x}_t$ and the hidden states $\boldsymbol{h}_{t-1}$ connections. The hidden states are the core of the **memory** feature of RNNs and represent the results of the feedback loops. The outputs can be given by,

$$\underset{m\times 1}{\boldsymbol{y_t}} = \phi_2\left(\underset{m\times m}{\boldsymbol{W}_y}\ \underset{m\times 1}{\boldsymbol{h_t}} + \underset{m\times 1}{\boldsymbol{b}_2}\right) \tag{3.15}$$

with the activation functions ϕ_1 (Equation (3.14)) and ϕ_2 not having to be the same. $\boldsymbol{W}_y$ are the weights of the output connection.

The foundation of the training process in RNNs is that the recurrent neurons' weights are adjusted recursively with the goal of learning what values that were obtained from the previous output are best to be kept in memory. The training in RNNs continues to have the same objective as in an ANN, i.e., minimizing a loss function. However, the standard backpropagation algorithm does not take into account the feedback loops. This refers to the recurrent neurons' current state depending on not only their input but also on the state of their previous step. In order to address this, an extension called the **Backpropagation**

Through Time algorithm (BPTT) was proposed in [Werbos, 1990]. Essentially, BPTT is a backpropagation algorithm executed in an unfolded RNN. The reason for that is that the gradient of the cost function at a processing step t with respect to either weight W_x or weight W_h is dependent on gradients that are calculated across every step.

3.5.1 Bidirectional RNNs

The bidirectional RNN (BRNN) was first proposed in [Schuster and Paliwal, 1997] as an extension of the traditional RNN. The main objective of this network is to remove the limitation of training only in the direction of the future, i.e., up to a predefined future time index. Since its architecture is composed of two RNNs (Figure 3.11) the BRNN performs training also towards the past[8] direction. In this manner, the network can be fed a greater volume of information and consider a broader range of features.

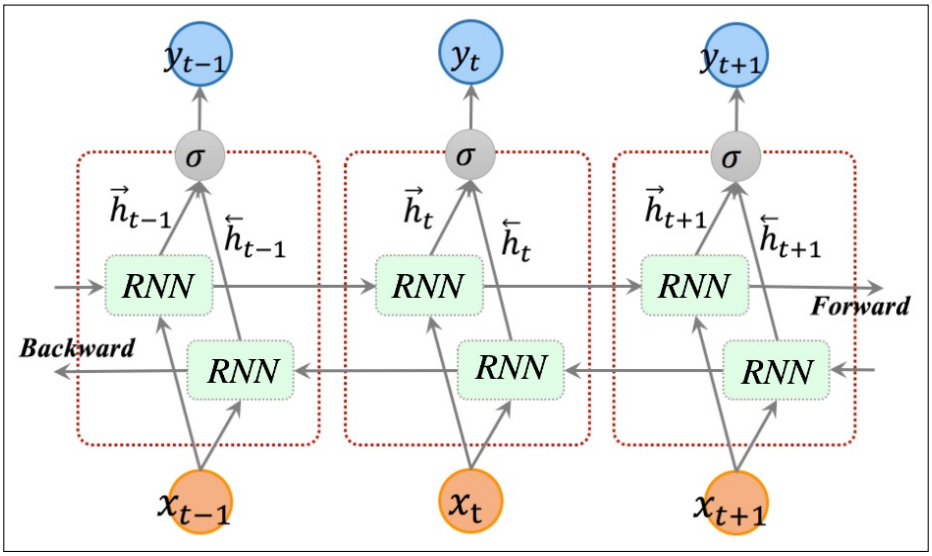

Figure 3.11: The RNNs represent the previously described recurrent units unrolled through t. The vector arrows on each $\boldsymbol{h}_t$ denote the different directions (positive and negative) that are taken to traverse the inputs for the calculations. In the end, the outputs from each unit ($\overleftarrow{\boldsymbol{h}}_t$ and $\overrightarrow{\boldsymbol{h}}_t$) are merged and activated (optional) at each step. **Adapted from**: [Cui *et al.*, 2018]

Finalizing, in addition to Equation (3.14), the equation for the computation of the *backwards recurrent layer* is given by,

$$\underset{m\times 1}{\overleftarrow{\boldsymbol{h}}_t} = \begin{cases} \mathbf{0} & \text{if t=0,} \\ \phi_1\left(\underset{m\times n}{\boldsymbol{W}_x}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}_h}\ \underset{m\times 1}{\overleftarrow{\boldsymbol{h}}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}_1}\right) & \textit{otherwise} \end{cases} \tag{3.16}$$

with the only change being the $\leftarrow$ in $\boldsymbol{h}_t$, which represents the negative direction that the calculations flow. As a result of this architecture, the number of learnable

[8] Usually, through a reverse copy of the sequence.

parameters (weights and biases) is higher and the calculations lead to longer gradient chains consequencing in a computationally expensive training.

3.5.2 Long Short-Term Memory Networks

The first form of RNNs, as it had recurrent connections, was first seen in [Hopfield, 1982] and it was called the Hopfield Network. Although, it was not until the work of [Rumelhart *et al.*, 1986] that the concept of RNNs was truly established. In that time, the popularity of these networks was short-lived. The root for this lies on the difficulties that appear during the training of ANNs which, with no counter procedures, were found to be even more frequent with traditional RNNs. Such phenomena are called vanishing and exploding gradients.

The **exploding gradients** problem, described in [Bengio *et al.*, 1994], regards to a great increase in the gradients vector norm, and when it occurs the training can be interrupted. This event is caused by the accumulation of oscillating gradients and it usually results in very large weight updates producing an unstable network. A possible approach to solve this is by applying regularization techniques (subsection 3.3.2). However, two superior solutions, explained in [Pascanu *et al.*, 2013], are **gradient norm scaling** and **gradient value clipping**. Gradient norm scaling is a mechanism that performs a normalization of the gradients whenever the vector norm goes over a threshold. Gradient value clipping constrains the gradient values to a minimum and a maximum clip value.

In comparison, the **vanishing gradients** problem presents a much more complicated resolution. It refers, as opposed to the exploding gradients problem, to the exponential reduction of the gradients vector norm to zero. This occurs while the training is handling with long-term[9] dependencies in the input sequences. The gradients become too small resulting in insignificant weight updates, thus, hindering the network's ability to learn such dependencies. Therefore, the traditional RNN is known to suffer from short-term memory. In order to reduce these problems, Long Short-Term Memory (LSTM) networks were proposed ([Hochreiter and Schmidhuber, 1997]) and, over the years, have received many improvements, e.g. [Gers *et al.*, 2000] and [Bayer *et al.*, 2009]. The overall control flow of LSTMs is equal to that of RNNs, what differs is the repeating module, i.e., the internal processes of the recurrent unit.

As seen in Figure 3.12, LSTMs have three main concepts:

- The **cell state** ($\boldsymbol{c}_t$), which presents an encoded version of information gathered from all the previously processing steps. As a result, this term is also called the **long-term state**;
- The **hidden state** and output ($\boldsymbol{h}_t$), which has the same purpose as in the traditional RNN, has the same objective of gathering information that is more focused on the previous step. Thus, it is also referred to as the **short-term state**;

[9] In [Gers *et al.*, 2000] it is declared that RNNs *"(...) fail to learn in the presence of time lags greater than 5-10 discrete time steps between relevant input events and target signals."*

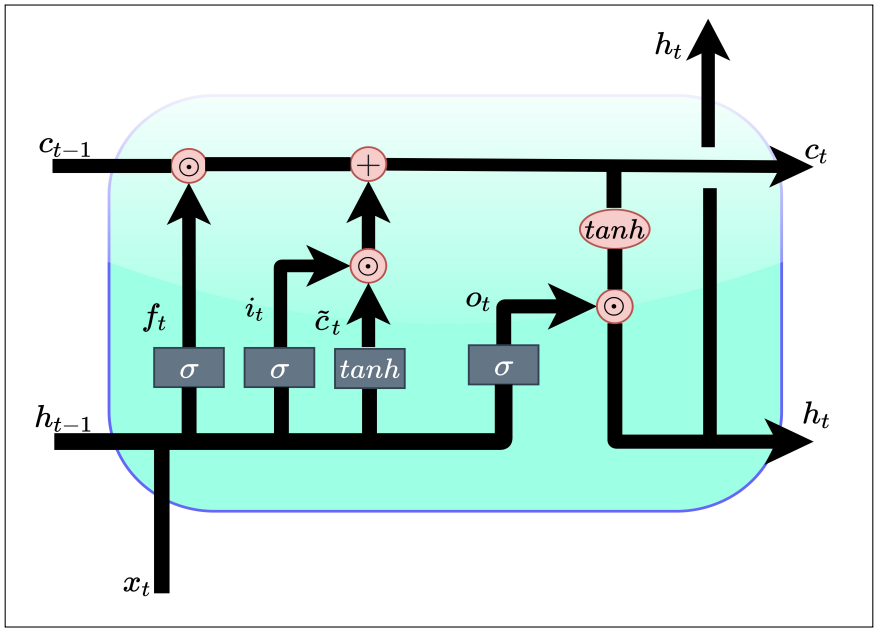

Figure 3.12: The recurrent processes in a LSTM unit. **Adapted from**: [Varsamopoulos *et al.*, 2019]

- The gating mechanisms, which are called **input gate** ($\boldsymbol{i}_t$), **output gate** ($\boldsymbol{o}_t$) and **forget gate** ($\boldsymbol{f}_t$) The **input gate** determines what values from the **candidate memory** ($\tilde{\boldsymbol{c}}_t$) are to be appended to the cell state. The **forget gate** determines what information should be removed or "forgotten" from the **cell state**. The **output gate** controls what elements from the cell state should be carried on in the unit's output and hidden state $\boldsymbol{h}_t$.

A LSTM unit can be expressed following an intuitive order of operations, where all gates and the candidate memory share the same format. They all deal with a combination of the input $\boldsymbol{x}_t$, the previous hidden state $\boldsymbol{h}_{t-1}$, and W refers to the matrices of weights of the connections. As the first step let the forget gate ($\boldsymbol{f}_t$) be defined as

$$\underset{m\times 1}{\boldsymbol{f}_t} = \sigma\left(\underset{m\times n}{\boldsymbol{W}^{(xf)}}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}^{(hf)}}\ \underset{m\times 1}{\boldsymbol{h}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}^{(f)}}\right) \tag{3.17}$$

The symbol σ refers to a specific activation function, the **sigmoid**, that transforms all values to fit the interval $]0, 1[$. The reason for this is that every gate represents a vector where its values dictate what is "remembered" and what is "forgotten". If the values are close to zero the data is to be removed, if they are close to one then it is to be conserved or added.

In the second step, the unit decides what information is to be appended to the cell state ($\boldsymbol{c}_t$). First, the input gate ($\boldsymbol{i}_t$) decides which values should be updated, and second the

new candidate values $\tilde{\boldsymbol{c}}_t$, which may be appended to the cell state, are created. Formally,

$$\underset{m\times 1}{\boldsymbol{i}_t} = \sigma\left(\underset{m\times n}{\boldsymbol{W}^{(xi)}}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}^{(hi)}}\ \underset{m\times 1}{\boldsymbol{h}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}^{(i)}}\right) \tag{3.18}$$

$$\underset{m\times 1}{\tilde{\boldsymbol{c}}_t} = tanh\left(\underset{m\times n}{\boldsymbol{W}^{(xc)}}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}^{(hc)}}\ \underset{m\times 1}{\boldsymbol{h}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}^{(c)}}\right) \tag{3.19}$$

where in Equation (3.19), *tanh* is the **hyperbolic tangent** activation function. After this, the cell state is updated from $\boldsymbol{c}_{t-1}$ to $\boldsymbol{c}_t$,

$$\underset{m\times 1}{\boldsymbol{c}_t} = tanh\left(\underset{m\times 1}{\boldsymbol{f}_t} \odot \underset{m\times 1}{\boldsymbol{c}_{t-1}} + \underset{m\times 1}{\boldsymbol{i}_t} \odot \underset{m\times 1}{\tilde{\boldsymbol{c}}_t}\right) \tag{3.20}$$

In the last step it is where the unit's hidden state and output are assembled. The output gate decides through the sigmoid function, which values of the updated cell state are to be carried on in $\boldsymbol{h}_t$. Then, the element-wise product of the output gate and the updated cell state is calculated forming $\boldsymbol{h}_t$. Mathematically, this is conjugated to

$$\underset{m\times 1}{\boldsymbol{o}_t} = \sigma\left(\underset{m\times n}{\boldsymbol{W}^{(xo)}}\ \underset{n\times 1}{\boldsymbol{x}_t} + \underset{m\times m}{\boldsymbol{W}^{(ho)}}\ \underset{m\times 1}{\boldsymbol{h}_{t-1}} + \underset{m\times 1}{\boldsymbol{b}^{(o)}}\right) \tag{3.21}$$

$$\underset{m\times 1}{\boldsymbol{h}_t} = \underset{m\times 1}{\boldsymbol{y}_t} = \underset{m\times 1}{\boldsymbol{o}_t} \odot \underset{m\times 1}{\boldsymbol{c}_t} \tag{3.22}$$

It is worth mentioning, that the BRNN (subsection 3.5.1) is modelled with LSTM cells (BiLSTM) more frequently than with the traditional recurrent units (e.g. [Graves and Schmidhuber, 2005], [Wöllmer *et al.*, 2010] and [Cui *et al.*, 2018]).

3.6 Summary

With the fundamentals of FNNs, CNNs and RNNs architectures covered, the next chapter aims in describing how these architectures are used and perform in the specific context of TS forecasting. For that goal, several studies are explored.

Chapter 4

Time Series Forecasting with Neural Networks

In retail, for reasons of labor and inventory planning it is required that models provide multi-step and not only one-step forecasts. Hence, this chapter first starts by introducing the two main ANN architectural strategies for multi-step TS forecasting.

Historically, RNNs have been used in sequence modeling, as such, given the natural interpretation of TS data as sequences of inputs and targets a variety of RNN architectures have been developed (e.g. [Salinas *et al.*, 2020] and [Lim *et al.*, 2020]). Since with CNNs that interpretation is not intuitive, concepts to adapt the traditional CNNs for TS forecasting are explored. Lastly, studies that explore TS forecasting in different fields with performance and popularity comparisons between DL, ML and classical models are overviewed.

4.1 Neural Network Multi-Step Forecasting Strategies

When long-term forecasts are required there is a good chance that one-step predictions are not enough, hence, multi-step forecasts are applied. For many classical forecasting models this ability might be trivial due to their design, yet, that is not the case for ANN models. In [Hamzaçebi *et al.*, 2009] and [Makridakis *et al.*, 2018a] there are two approaches in common:

- The recursive or iterative strategy;
- The direct strategy.

The **recursive** strategy is the oldest and most intuitive. Considering a TS that has $N + 1$ observations and disregarding the noise, let the trained one-step forecasting model

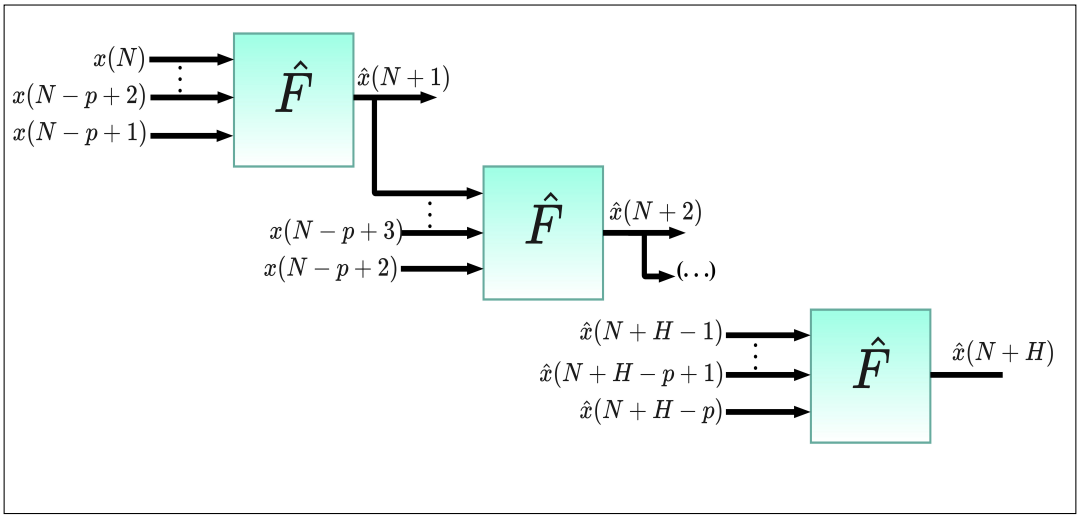

Figure 4.1: Forecasting with the recursive strategy. **Adapted from**: [An and Anh, 2015]

be $\hat{F}$, then

$$\hat{x}(N+h) = \begin{cases} \hat{F}\Big(x(N), ..., x(N-p+1)\Big), & \text{if } h = 1 \\ \hat{F}\Big(\hat{x}(N+h-1), ..., \hat{x}(N+1), x(N), ..., x(N-p+h)\Big), & \text{if } h \in \{2, ..., p\} \\ \hat{F}\Big(\hat{x}(N+h-1), ..., \hat{x}(N+h-p)\Big), & \text{if } h \in \{p+1, ..., H\} \end{cases} \tag{4.1}$$

where p is the number of lagged values used to forecast. The resulting forecast for $h = 1$ is provided back to $\hat{F}$ as input for $h = 2$, this logic is repeated H amount of times needed for a H-step forecast (Figure 4.1). This strategy has the drawback of accumulating errors and that accumulation is more severe the greater the horizon H is.

With the **direct** strategy (Figure 4.2), a different model for each $N + h$ step is built,

$$\hat{x}(N+h) = \hat{F}_h\Big(x(N), ..., x(N-p+1)\Big) \tag{4.2}$$

with $h \in \{1, ..., H\}$. A multi-step forecast is obtained through the junction of H predictions. With the way this strategy is designed, relationships that might exist between the predictions are not considered. It is, however, immune to the accumulation of errors that the recursive strategy suffers from.

The mentioned studies show contradictory results regarding the forecasting performance of the two strategies. In [Makridakis *et al.*, 2018a] the recursive strategy shows better results than the direct method, while in [Hamzaçebi *et al.*, 2009] the opposite is shown.

4.2 Convolutional Neural Networks

Due to the state-of-the-art performance of CNNs in various image-related problems its original concepts needed to be challenged for other types of data as well. Thus, these

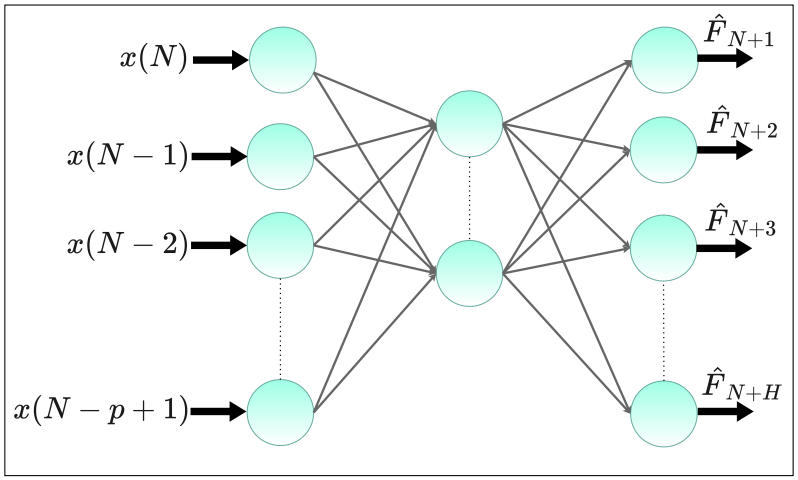

Figure 4.2: Example of an ANN architecture for a direct forecast. **Source**: [Hamzaçebi *et al.*, 2009]

networks have also been studied in different scenarios, for example, natural language processing (NLP) [Wang and Gang, 2018] and human activity recognition [Ha and Choi, 2016]. For TS data, first progress was shown in [Kiranyaz *et al.*, 2015] with the proposal of *1D* CNNs. The main differences between the *1D* and the *2D* CNNs is that the input and output data are expected to be *2D* matrices instead of *3D* and that convolutions follow in just one dimension, i.e., the filters slide only in one dimension as their height is fixed, resulting in a much lower computational task (Figure 4.3).

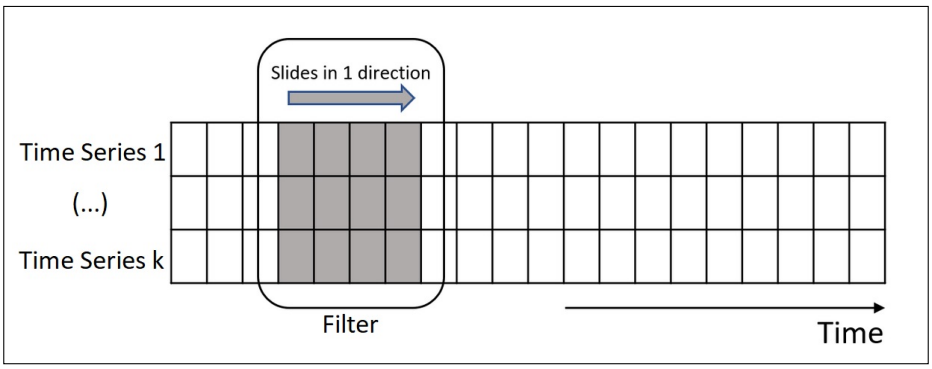

Figure 4.3: The sliding occurs only in the time dimension. Thus, performing a *1D* convolution, in this case, applied to a multivariate TS, i.e., k context-related TSs with the same length. The filter has its height fixed to the number of TSs (size $4 \times k$).

Recent advances, however, show that researchers have built CNN models with stacked convolutional layers that make use of *1D* **dilated causal convolutions** ([Lim and Zohren, 2021]). This type of convolution was first seen in the autoregressive audio generation model, **WaveNet**, proposed in [Oord *et al.*, 2016].

4.2.1 WaveNet

Causal convolutions are used to ensure that an output at time t is convolved only with elements from time t and earlier in previous layers. This means that there is no leakage from the future into the past, therefore, a prediction $\hat{x}(t+1)$ obtained by the model at t does not depend on the future $x(t+1), ..., x(N-1)$. In comparison to RNNs, causal convolutions have the advantage of not possessing recurrent connections and because of that the models benefit from faster trainings, however, they require many layers or large filters in order to increase the receptive fields, i.e., to ensure that all relevant historical data is taken into account.

Dilated convolutions fulfill that need as they are convolutions where the filters are applied over an area that is wider than their chosen length. This is because the filters skip a certain number of steps on the input sequence. The number of steps that are skipped is defined by a parameter called **dilation rate** or dilation factor d. More specifically, in dilated convolutions the filters are applied to every dth element of the input with d being usually increased by a factor of two along every layer $l = 1, ..., L$ i.e., $d \in [2^0, 2^1, ..., 2^{L-1}]$ (Figure 4.4).

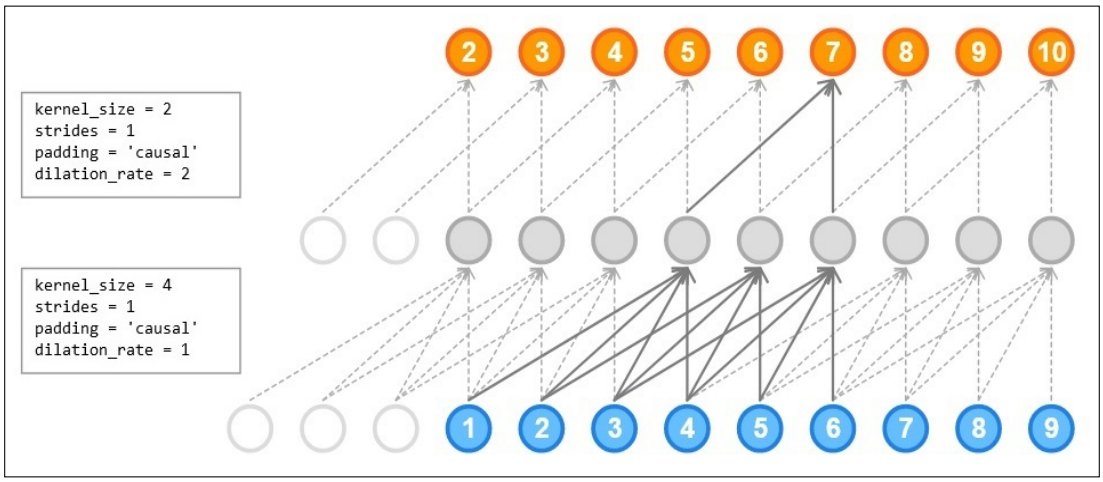

Figure 4.4: Stacked dilated causal convolutional layers. **Source**: [*Convolutions in Autoregressive Neural Networks* 2019]

As explained previously, the local receptive field of a neuron is the set of elements in the input that can alter the output of that same neuron. Nevertheless, the receptive field of the overall model may prove to be important, i.e. the number of elements in the input layer or TS that can affect the output in the final layer or the forecast. Formally, it is given by

$$r := 2^{L-1}K \tag{4.3}$$

with L being the number of layers and K being the filter size. It is worth noting that in order to preserve the causal condition it is convenient to pad the input with zeros before the TS origin (Figure 4.4). In training, this enables the model to learn how to predict the

early steps, in other words, the model learns how to start a sequence and not just how to continue it. More specifically, the padding vector is a vector of zeros where its size can be defined by,

$$s := (K-1)d \tag{4.4}$$

Similar to the LSTM, WaveNet adopted a gating mechanism, however, since vanishing gradients are not a usual problem with standard CNNs only an output gate was necessary ([Dauphin *et al.*, 2017]). It replaces the traditional ReLU functions with **gated activation units** ([Van Den Oord *et al.*, 2016]). Mathematically,

$$\boldsymbol{h} = tanh(\underset{k\times m}{\boldsymbol{W}^{f,l}} * \underset{N\times m}{\boldsymbol{x}^{(1)}}) \odot \sigma(\underset{k\times m}{\boldsymbol{W}^{g,l}} * \underset{N\times m}{\boldsymbol{x}^{(2)}}) \tag{4.5}$$

where f and g denote the filter and gate, respectively. l is the layer index, m is the number of input channels and the weights $\boldsymbol{W}$ are the learnable convolution filters. $x^{(1)}$ and $x^{(2)}$ represent the two input groups that result from a split of the feature map outputted from the previous layer. $*$ represents the dilated causal convolution operation and $\odot$ the element-wise product. Similar to the output gate in a LSTM, the second group, i.e., where the sigmoidal activation occurs, forms the gate that controls the information flow of the other group (Figure 4.5).

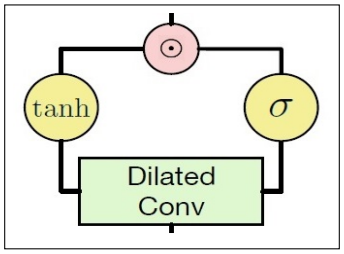

Figure 4.5: The gated convolutional layer of the WaveNet model. **Adapted from**: [Oord *et al.*, 2016]

As there are long chains of multiplication when dealing with very deep networks, which can be the case with dilated causal CNNs, training accuracy can become saturated and degrade (high training losses). In [He *et al.*, 2016], a method to reduce degradation in DNNs is presented. It is explained that if multiple nonlinear layers can map any complex function $\mathcal{H}(x)$, assuming the input and output dimensions are equal, then they can approximate a residual function $\mathcal{H}(x)-x$ as well. To prevent the training issue, the authors proposed to explicitly let the layers learn a nonlinear residual function $\mathcal{F}(x) := \mathcal{H}(x) - x$ by allowing fast passages of linear mappings x, i.e., $\mathcal{F}(x) + x$ since $\mathcal{H}(x)$ is the originally wanted function. This means that even in case of $\mathcal{F}(x) = 0$, the layers still preserve mapping x. Hypothetically, this ensures that the network cannot be worse in terms of training than if it did not have the passage of x.

Residual learning is applied in every few stacked layers. Formally, a residual block

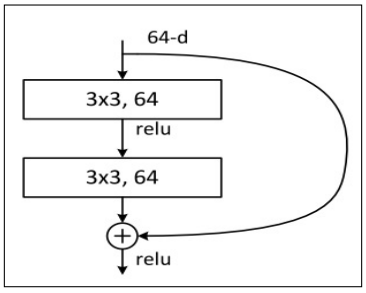

Figure 4.6: A residual connection skipping two convolutional layers. In this case, the residual function $\mathcal{F}$, which represents two convolutional layers, and the input x have matched dimensions. **Source**: [He *et al.*, 2016]

is given by

$$y = \mathcal{F}(x_i, \{\boldsymbol{W_i}\}) + x_i \tag{4.6}$$

where i is the block's index, x is the input vector (output of block $i-1$), y the output vector that is provided to the next block (i.e., x_{i+1}) and $\mathcal{F}(x_i, \{\boldsymbol{W_i}\})$ is the residual mapping to be learned. The element-wise addition $\mathcal{F} + x$ is achieved through a skip connection (can

model	top-1 err.	top-5 err.
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40

Figure 4.7: Comparison of residual CNNs with plain CNNs, all being 34-layers-deep networks. In ResNet-34 A the zero-padding strategy was used, along with the rest of the residual connections being parameter-free. B used parameterised skip connections only in situations where dimensions needed to be increased and C used parameterised skip connections at all times. **Adapted from**: [He *et al.*, 2016]

skip one or more layers), i.e., a residual connection (Figure 4.6), and it does not introduce any parameters (weights) to the network. However, in CNNs $\mathcal{F}$ and x are prone to have different dimensions due to the convolutional operations employed (for example, number of filters $F \neq$ number of input channels D). To resolve this, the authors propose two methods:

1. Perform linear projections by padding x with zeros in order to upsample it, this does not add extra learnable parameters;

2. Perform linear projections with **parameterised skip connections**, these are 1x1 convolutions ($F = 1$ and $K = 1$) that match the dimensions and increase the overall learnable parameters of the network.

Mathematically, a linear projection W_s of a skip connection is given by

$$y = \mathcal{F}(x_i, \{\boldsymbol{W_i}\}) + \boldsymbol{W_s} x_i \tag{4.7}$$

In [He *et al.*, 2016], it was shown that the second method produced slightly better results (Figure 4.7), consequently, it was adopted in WaveNet (Figure 4.8). It is worth noting that in the WaveNet architecture the skip connections lead directly to the output layer, this is to enforce that the outputs from the convolutional layers do not face any influence dilution from any further layers.

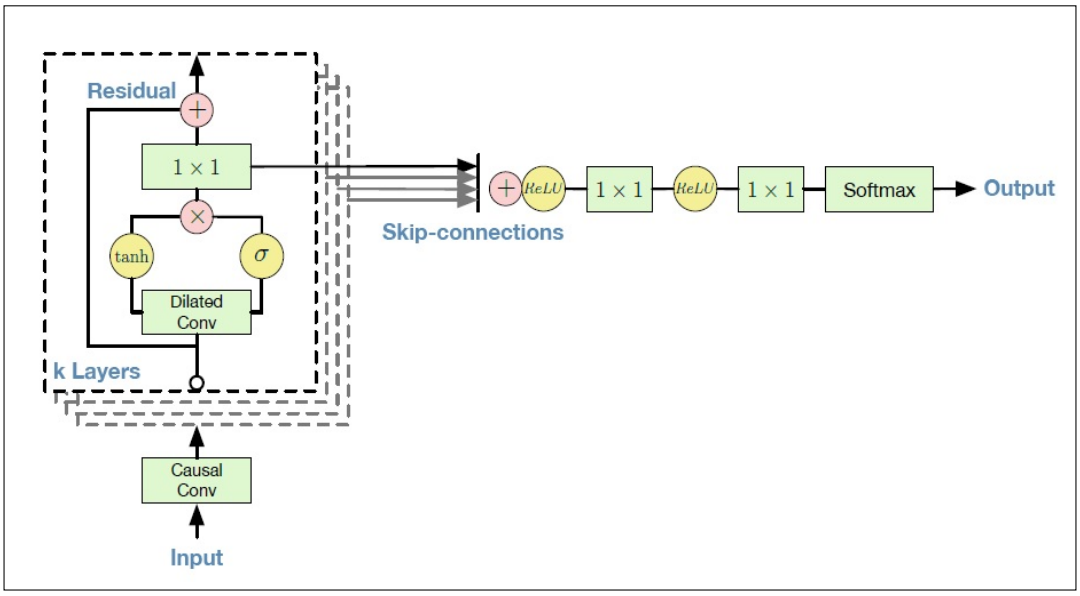

Figure 4.8: The WaveNet's architecture. **Source**: [Oord *et al.*, 2016]

4.2.2 Adapted WaveNets

In [Borovykh *et al.*, 2018] and [Papadopoulos, 2018], the WaveNet was adapted to face the problem of TS forecasting. The first, called the Augmented WaveNet, was developed for financial TSs, while the second, SeriesNet, was performed on the CIF2016 competition dataset ([*CIF - Computational Intelligence in Forecasting* 2021]).

Similar to the WaveNet architecture, both models expect an input that is given by

$$[0, ..., 0, x(0), ..., x(N-1)] \in \mathbb{R}^{N+r}$$

and the output obtained in the last layer is

$$[\hat{x}(0), ..., \hat{x}(N)] \in \mathbb{R}^{N+1}$$

with $\hat{x}(N)$ being a one-step forecast. A H-step forecast is obtained by feeding previously obtained predictions back to the model, hence, applying a recursive strategy (section 4.1).

The next points present the main architectural aspects (Figure 4.9) of the Augmented WaveNet:

- Presents four layers;
- In order to increase training efficiency and lower complexity the gating activation mechanism was not used, instead every dilated convolutional layer is followed by a standard ReLU nonlinear activation;
- Parameterised skip connections are done only in the first hidden layer (Figure 4.9) and do not lead directly to the final output layer (L);
- Standard residual connections are applied in every layer except the first and L;
- The forecasted TS results from a 1x1 convolution.

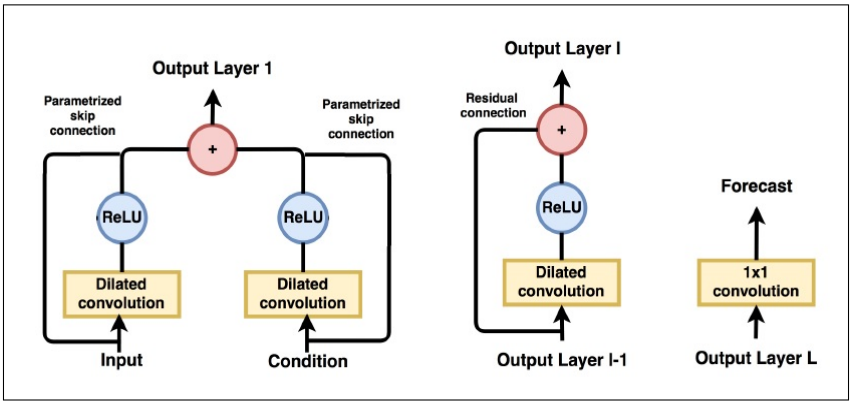

Figure 4.9: The Augmented WaveNet architecture. **Source**: [Borovykh *et al.*, 2018]

In comparison, SeriesNet shows more similarities with WaveNet (Figure 4.10), its key characteristics are:

- It has seven layers ($L = 7$);
- The gating mechanism is also not used, instead an activation function called scaled exponential linear unit (SELU) was included;
- Such as in WaveNet, parameterised skip connections lead to the final layer and standard residual connections are done in the dilated convolution blocks;
- The forecasted TS results from the sum of all parameterised skip connections followed by a ReLU activation and, lastly, a 1x1 convolution.

Both models in their dilated causal convolutional layers, just like WaveNet, present small filter sizes of $K = 2$, however, they contrast in the number of filters used. While SeriesNet demonstrated best results with $F = 32$, Augmented WaveNet was best with $F = 1$ to $F = 3$.

The L2-regularization technique, the gradient descent optimizer ADAM and the MAE

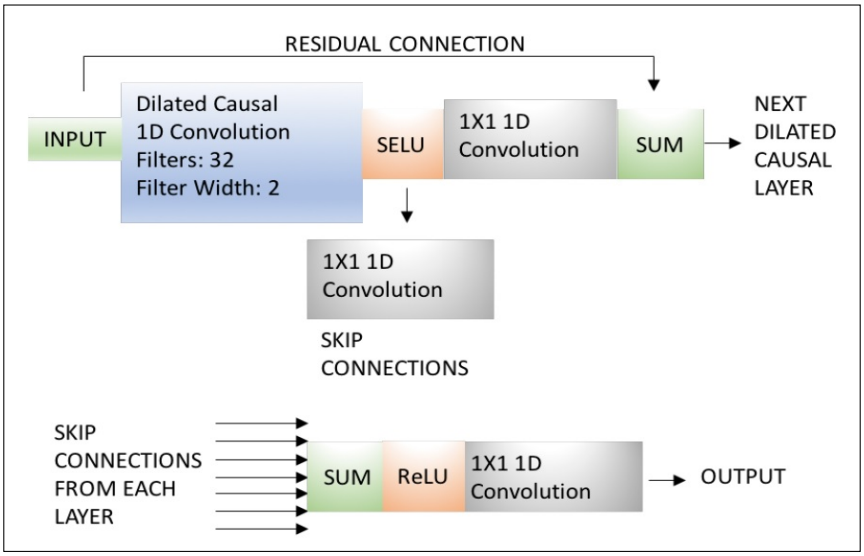

Figure 4.10: The SeriesNet architecture. **Source**: [Papadopoulos, 2018]

loss function are all applied on both models. Additionally, SeriesNet regularizes its last two layers with $prob = 80\%$ dropout (subsection 3.3.2).

The Augmented WaveNet was compared with multiple models on a daily TS, designating 750 days for training the models and 250 days for testing purposes. It showed better results than a LSTM model and held up against the classical econometric model, Vector Autoregression (VAR). Also, SeriesNet was compared with models that were in the top ranks of the CIF2016 competition. It showed better forecasting performance than a MLP, a baseline LSTM and it was very close to the two best models. Those were hybrid methods that combined LSTMs with ETS and double ETS. The data collection of CIF2016 is comprised of 72 monthly TSs with forecast horizons of 6 to 12 months.

4.3 Comparative Studies

In [Makridakis *et al.*, 2018a], an extensive comparison between classical and ML models was performed. That research was made with the objective of expanding what had already been presented in [Ahmed *et al.*, 2010], where a study with only ML models was conducted. Eight classical and ten ML models were evaluated. The ARIMA and the ETS were some of the presented classical models. As for ML models, in addition to those in [Ahmed *et al.*, 2010], RNN and LSTM networks were also included. The models had to produce 18-step forecasts on 1045 monthly TSs, a subset of the 3003 TSs belonging to the M3 competition ([Makridakis and Hibon, 2000]), and followed the multi-step forecasting strategies described in subsection 4.1 (plus a multi-neural network strategy).

For the data preprocessing stage, several methods were tested and this is due to the inconsistency the literature shows in this matter. Some authors claim that any ANN method should be able to sucessfuly model the original data ([Gorr, 1994]), while

Method	Short	Medium	Long	Average	CC
MLP Iterative	9.53	12.34	15.00	12.29	245.58
MLP Direct	10.72	13.55	16.20	13.49	438.53
MLP Multi	9.53	12.69	16.08	12.77	4006.82
BNN Iterative	9.39	12.08	14.80	12.09	141.91
BNN Multi	9.48	12.70	15.96	12.71	2046.49
Naive 2	10.78	12.46	15.08	12.77	**1.48**
SES	9.17	10.85	13.77	11.26	1.60
Holt	9.07	11.18	14.29	11.51	1.75
Damped	8.96	10.63	13.46	11.02	2.07
Comb	8.95	10.57	13.38	10.97	2.65
Theta	8.96	**10.53**	**13.19**	**10.89**	1.70
ARIMA	**8.93**	11.08	13.84	11.28	73.50
ETS	9.07	10.98	13.74	11.26	56.66

Figure 4.11: Evaluation of forecasts on various horizons (best in bold). The short-term horizon regards to 1 to 6 months ahead, medium-term 7 to 12 months ahead and long-term 13 to 18 months ahead. CC is the computational complexity, it refers to the time a model requires to train and, lastly, predict. **Source**: [Makridakis *et al.*, 2018a]

others state that preprocessing TSs is imperious[1], such as assuring the TS regularity and stationarity. The best preprocessing technique was found to be a combination of the Box-Cox transform (subsection 2.3.4) with deseasonalisation and, in certain situations, with detrending. The performance, measured with sMAPE, of the various models is presented in Figure 4.11. As seen in the figure, the RNN and the LSTM were missing in the performance measure of multi-step forecasts and this is because the authors only evaluated them for one-step forecasts (Figure 4.12).

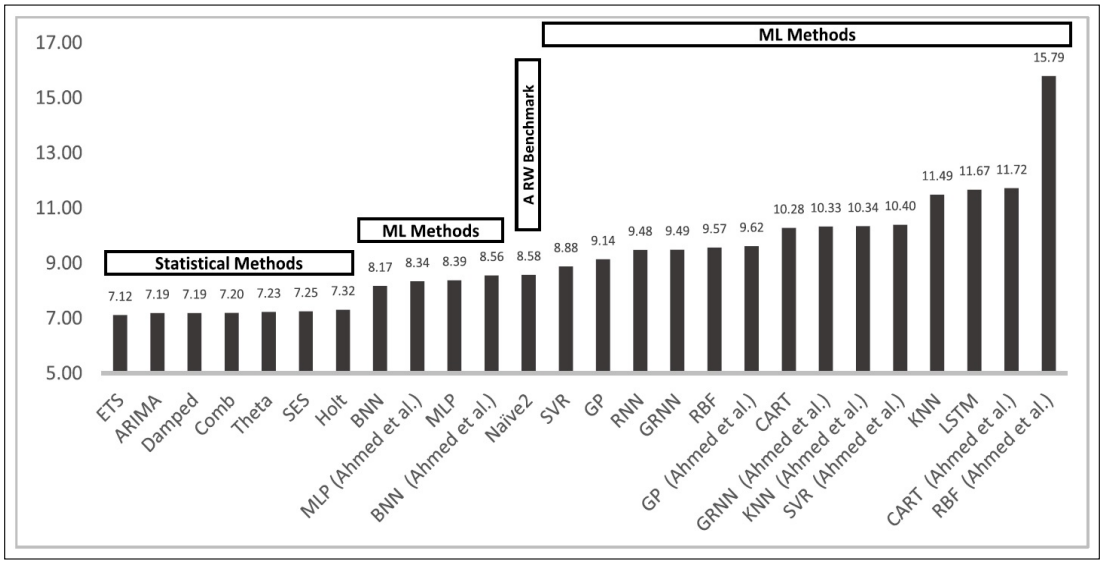

Figure 4.12: Results for one-step-ahead forecasts, now with the LSTM and others ANNs. **Source**: [Makridakis *et al.*, 2018a]

Recently, since it is a popular academic topic, a broad review for financial TS

[1] In [Zhang and Qi, 2005] it is stated that ANNs by themselves are not capable of capturing trends and seasonalities.

forecasting with ANNs was done in [Sezer *et al.*, 2020]. The number of reviewed papers was 140 and focused on areas, such as stock, cryptocurrency and other related financial prediction challenges. It was learned that RNN models, against deep MLPs, CNNs and others, were the dominant approach (Figure 4.13), particularly, with LSTMs presenting the best overall results. Despite not being mentioned the specifics of the forecasting strategies, this information is contrasted to what was seen in the previous study.

The results obtained in the M4 competition ([Makridakis *et al.*, 2018b]) are also encouraging to DL practitioners. From 49 valid participants evaluated on 100,000 TSs (more details in [Makridakis *et al.*, 2019]), the winner of the competition for both point and interval forecasts, constructed a hybrid model that mixed ETS formulas with LSTMs ([Smyl, 2020]).

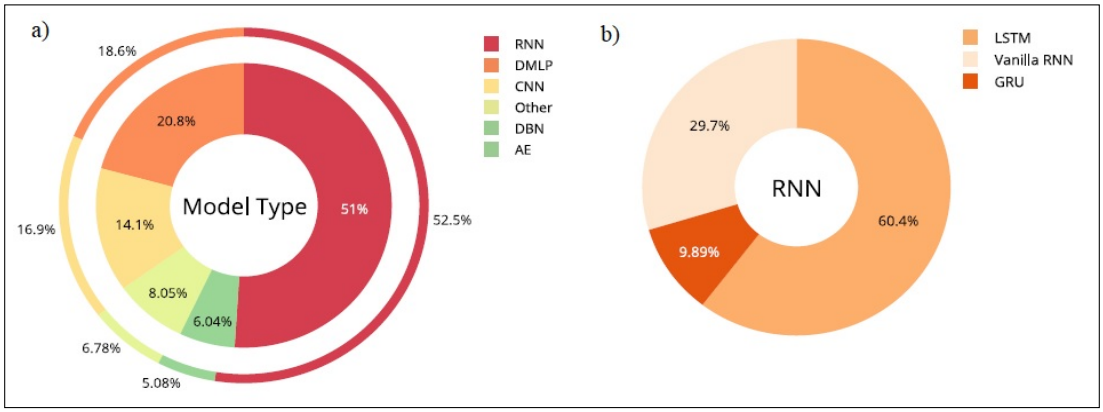

Figure 4.13: a) Models' popularity distribution. The outer circle considers studies from three years previous and up to 2019, while the inner circle considers all the studies reviewed. b) From all the RNN models LSTMs are the preferred approach. **Source**: [Makridakis *et al.*, 2018a]

The M5[2] competition is the most recent forecasting competition ([Makridakis *et al.*, 2020]) and the data focuses on multivariate daily retail sales on various aggregation levels[3]. In comparison to the M4 competition there were more participants as 5507 teams from 101 countries had to provide a 28-step forecast. Pure ML methods were, for the first time in M competitions, the top-performing approaches with the first, second, fourth and fifth places having a LightGBM ([Ke *et al.*, 2017]) model in their solution. The third place proposal had its basis on Amazon's DeepAR ([Salinas *et al.*, 2020]) model. Specifically, it consisted in 43 deep LSTMs with an encoder-decoder architecture ([Sutskever *et al.*, 2014]).

In the M competitions, in terms of performance, popularity and variety it is possible to see the great progress that ML and DL models have made along the years. Creating their way from being highly questioned in the past to being considered the present state of the art.

[2] https://www.kaggle.com/c/m5-forecasting-accuracy/overview

[3] This means that, for example, teams could sum the resulting bottom-level forecasts to perform forecasts of higher hierarchies.

4.4 Summary

Despite the present literature and works that may come out regarding the M5 competition, specific literature about retail TS forecasting using ANN models is surprisingly scarce. Additionally, no studies were found depicting high-frequency (hourly or half-hourly) retail TSs nor CNNs showed significant presence in the observed studies, which can raise questions about their performance on TS forecasting. Thus, the next chapter presents a retail forecasting experiment that combines different DL models with classical statistical analysis and data preprocessing approaches on high-frequency (half-hourly) TSs.

Chapter 5

Case Study and Experiments

5.1 Problem Definition and Datasets

The following experiments aim to help the development of TS forecasting solutions for a company that provides FSSs to a large number of retail stores throughout the world. The company's main objective is to optimize labor planning and that is achieved through client flow forecasts. Specifically, retail stores expect optimized working schedules for each employee every month. As such, an evaluation method was developed to mirror that scenario.

Multiple ANN architectures will be tested against a baseline naive model. The focus, however, will not only be on to the performance of each model but also to the effect on results of two data preprocessing methods.

The data used in the experiments are univariate half-hourly TSs that belong to three retail stores, A, B and C (Figure 5.1). Store A presents 37384 observations of client entries between 2015-01-02 and 2019-07-23. Concretely, 1649 days were recorded as 14 days are missing (e.g., Christmas days). Store B has 976 days registered between 2017-06-12 and 2020-10-18 (248 missing days) composed by 24437 observations. Lastly, store C presents 48517 observations between 2015-01-02 and 2020-10-18 with 2088 registered days (28 missing days).

Regarding the stores there are two important aspects:

1. Working schedules are not the same in every day;
2. Store B and store C both display the COVID-19 lockdown effect which can negatively influence forecasts (Figure 5.2).

5.2 Evaluation Methodology

The metrics that were used to evaluate the forecasts are MAE, RMSE and, due to client flow intermittence, the unit-free metric MAAPE.

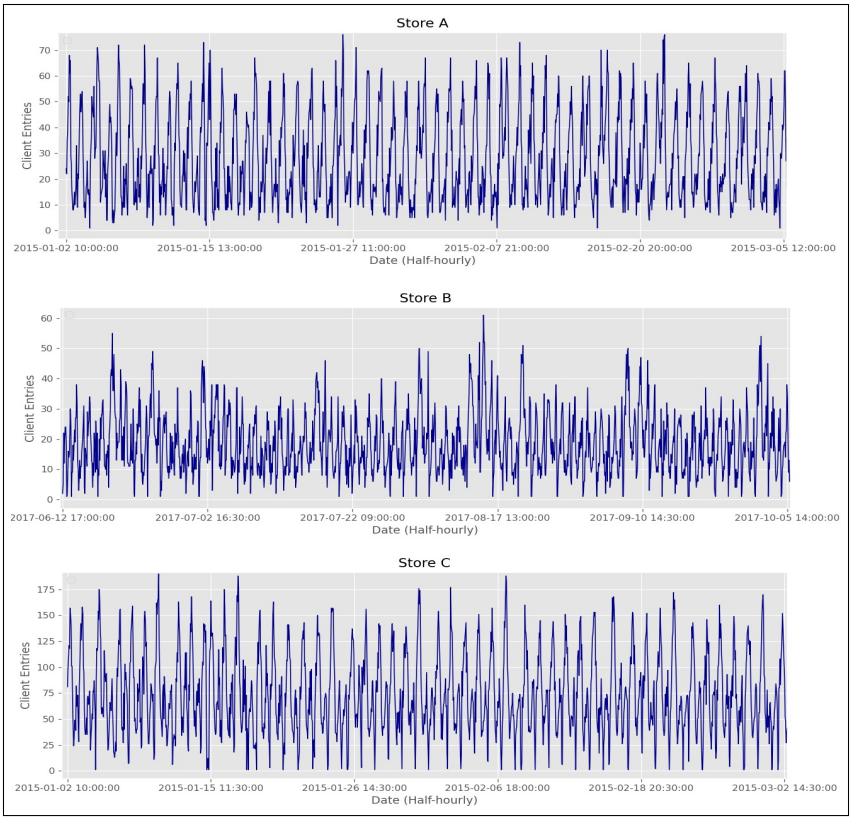

Figure 5.1: The first 1400 observations (∼2 months) of each TS.

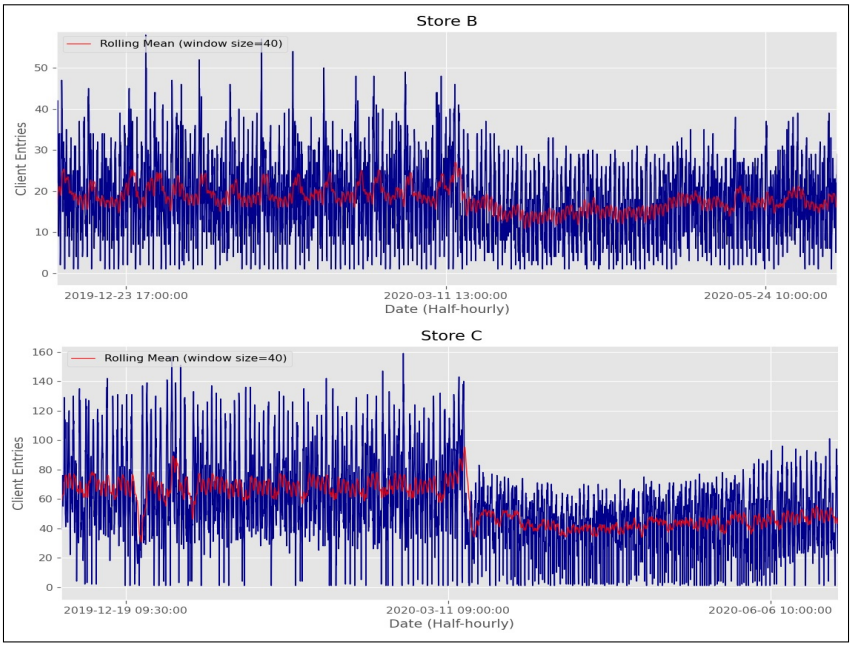

Figure 5.2: The COVID-19 lockdown effect is visible in both stores around the middle of March.

5.2.1 Time Series Cross-Validation

In every phase of the TSs forecasting process the time dependencies that are present between observations are to be preserved and the evaluation phase is not an exception. For various ML and DL problems the standard k-folds cross-validation (CV) method is valuable. The strategy consists in dividing a dataset in k folds, then, each fold is iteratively used as a test set to evaluate the model while the remaining $k-1$ are used for training. The order in which the folds are used can be random or consecutive regarding the original data sequence. However, due to the necessary preservation of dependencies between observations in TSs the strategy needs to be modified.

In [Hyndman and Athanasopoulos, 2018], a technique for TS CV is described. It consists in iteratively evaluating the model on consecutive test subsets. The end of each iteration marks the concatenation of the test subset with the training subset increasing its size for retraining. This is repeated until the totality of both train and test sets is utilized (Figure 5.3). This method is commonly called walk-forward or nested CV. A complete evaluation with the walk-forward method can be done by averaging the metrics obtained in all the iterations.

In [Varma and Simon, 2006], with the objectives of tuning and evaluating two different classifiers, nested CV produced results that were better and more reliable than other CV methods. In the following experiments, however, no hyperparameter tuning was conducted on the models. Thus, walk-forward validation is only used as an evaluation method.

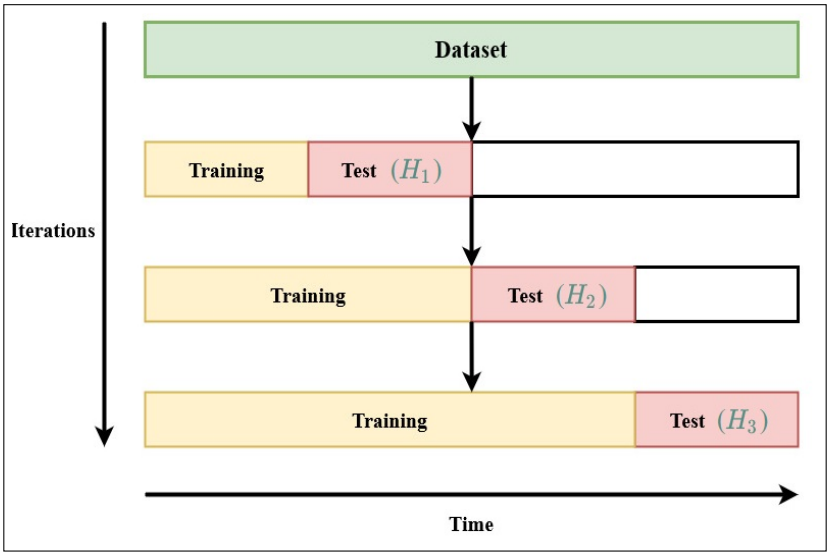

Figure 5.3: TS CV in an expanding walk-forward basis.

5.2.2 Train and Test Sets

Considering the walk-forward evaluation method, the models will be evaluated in **three** iterations. Since in each month employees need to have their working schedule available for the next month the test sets H are considered to be the number of observations that

corresponds to the last 30 days. Table 5.1 presents the size of each test set, in other words, it presents the three forecast horizons that correspond to 30 days.

Table 5.1: The sizes of each store test sets.

	Iteration 1 (H_1)	Iteration 2 (H_2)	Iteration 3 (H_3)
Store A	686	699	692
Store B	750	747	746
Store C	693	697	695

5.3 Data Preprocessing

In this experiment, three data settings are evaluated and compared:

1. The first setting has no relevant data treatment (raw);
2. The second has its daily schedules standardized through entry imputation and removal;
3. The third uses schedule standardization plus a HR transform (subsection 2.3.4).

The reason for choosing the HR method as the TS statistical transform is because it provides a greater probability of achieving stationarity than other popular methods ([Dong *et al.*, 2013]) and due to its capacity of dealing with both seasonal and trend components.

5.3.1 The Sliding Window Method

In TS forecasting, future values have their basis on past values. Because of this, the problem of forecasting is naturally classified as a regression problem. As mentioned in section 3.3, regression is a type of supervised learning problem that refers to discrete outputs. In this context, supervised learning refers to training a model by comparing the forecasted values with the target values. This adjustment is possible with a TS' past values and, as such, the training set can be framed towards that goal. In [Dietterich, 2002], a method to frame TSs as a supervised learning problem was presented and since then it has shown tremendous popularity (for example, [Paoli *et al.*, 2010], [Makridakis *et al.*, 2018a], [Li *et al.*, 2019] and [Vafaeipour *et al.*, 2014]). It is called the sliding window method and it consists in transforming the data into several fixed-size combined input and output windows with each pair composing a training sample (Figure 5.4). Within its process, three parameters are considered:

1. The stride s in which the window is to be slid;
2. The number of lagged values p in the input;

3. The number of values in the output which is equal to the pretended forecast horizon H.

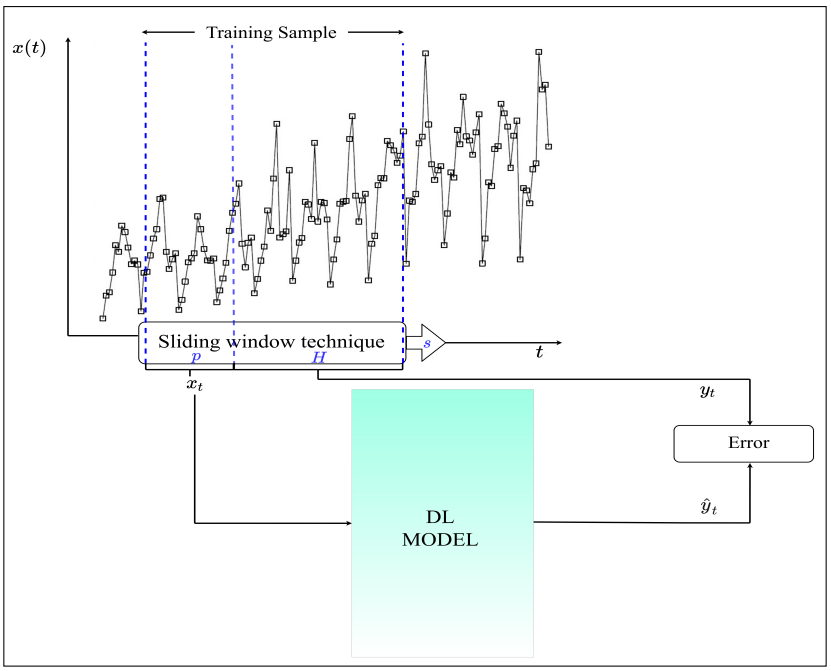

Figure 5.4: The sliding window sampling method. **Adapted from**: [Paoli *et al.*, 2010]

Generally, the input vectors $\boldsymbol{x}_t$ are made as

$$\boldsymbol{x}_t = \Big(x(t-p+1), x(t-p+2), ..., x(t)\Big) \tag{5.1}$$

with $p-1 \leq t \leq J-H-1$ using the sliding window on a univariate TS $x(t)$ with J training records. In order to train the model for a specified horizon, the output vectors $\boldsymbol{y}_t$ are given by

$$\boldsymbol{y}_t = \Big(x(t+1), ..., x(t+H)\Big) \tag{5.2}$$

Logically, just as H, p is a sensitive parameter. While windows with few lagged values may prove to possess insufficient information (e.g., seasonality), large windows can increase complexity and reduce the network's learning capabilities ([Khan *et al.*, 2019]). Thus, if the architecture of the forecasting model allows, various p values should be tested. As for the stride s, in these experiments, a value of 1 is used at all times in order to maximize the number of samples that are provided to the models.

Concluding, the number of training samples g created by the sliding window method on some TS with J training records is given by $g = J - p - H + 1$.

5.3.2 Store Schedules: Standardization

As mentioned, the schedules in the stores may vary from day to day. Since the forecasting problem is modelled as a univariate problem, no contextual information is

given including information regarding the schedules. Hence, the models may not know for what half-hours in the day they are trying to predict. Schedule standardization, i.e., constraining the TSs to a specified schedule is a solution for that issue. In its process there are two steps, missing observations are imputed and extraneous observations are removed.

So that data integrity is not greatly threatened, the chosen schedule is the schedule that is most frequent. All the stores have the same most frequent schedule, from 09:00 to 21:00 corresponding to 25 half-hourly records (Table 5.3). As it is of low complexity, linear interpolation was the chosen imputation method (Table 5.2).

Table 5.2: Number of imputations against the size of the training sets.

	Iteration 1	Iteration 2	Iteration 3
Store A	**3977** / 38250	**4048** / 39000	**4105** / 39750
Store B	**294** / 21750	**300** / 22500	**304** / 23250
Store C	**5014** / 49500	**5085** / 50250	**5156** / 51000

The standardization process, however, originates three drawbacks:

1. The model is assumed to forecast only for a specified schedule;
2. In comparison to a raw setting, H might have to be longer to conform with the artificial schedule. In this case, $H = 750$ (test set size) at all times since it corresponds to 30 days of 25 records each;
3. In the evaluation phase, performance comparisons between settings are not straightforward.

The first drawback is countered by having a second or more forecasting models filling predictions for other schedules, however, this is not explored in this work. In relation to the second and third points, in order to compare results between standardized and raw settings it should be known that forecasts aimed to the outside of the true schedule or to the outside of the restricted schedule are all invalid for evaluation. Table A.1 shows the quantity of forecasts that are valid for evaluation in each of the walk-forward iterations.

Table 5.3: The number of daily occurrences of the three most frequent schedules (MFS) on the initial training set (the order does not change with further walk-forward iterations).

	1st MFS	2nd MFS	3rd MFS	Days
Store A	*09:00-21:00*: **1006**	*09:00-21:30*: **142**	*09:30-14:00*: **104**	1559
Store B	*09:00-21:00*: **440**	*08:30-21:00*: **141**	*09:00-20:30*: **126**	886
Store C	*09:00-21:00*: **516**	*09:00-21:30*: **498**	*08:30-21:30*: **235**	1998

5.3.3 Detrending and Deseasonalizing with Harmonic Regression

In order to use the HR method presented in subsection 2.3.4, there are two prerequisites (Figure 5.5):

1. The TSs schedules must be standardized;

2. The seasonal periods must be discovered.

The first condition is satisfied with the standardization method described in the previous subsection, while the second can be met by applying FFTs to the TSs and observing the designated plots. Fourier analysis, however, requires that the signals that are being analyzed are not greatly described by trends. Therefore, in order to ensure that the FFT is used correctly ADF unit root tests are first applied.

The results in Tables 5.4, 5.5 and 5.6 indicate, with confidence, that in every iteration all the training sets are trend-stationary. As such, no transformations were required and FFTs were applied to discover any significant seasonal periods (z).

Table 5.4: ADF test results on store A training sets.

	Iteration 1	Iteration 2	Iteration 3
t-statistic	-23.042	-23.398	-23.678
Critical Value (1%)	-3.430	-3.431	-3.431
Critical Value (5%)	-2.862	-2.862	-2.862
Critical Value (10%)	-2.567	-2.566	-2.567

Table 5.5: ADF test results on store B training sets.

	Iteration 1	Iteration 2	Iteration 3
t-statistic	-14.270	-14.497	-14.778
Critical Value (1%)	-3.431	-3.431	-3.431
Critical Value (5%)	-2.862	-2.862	-2.862
Critical Value (10%)	-2.567	-2.567	-2.567

The flow of client entries displays two dominant daily and half-daily (minimum considered) seasonality (e.g. Figure 5.6) in all three stores, therefore, two Fourier pairs were chosen for the HR, specifically, the daily $z_1 = 25$ and the half-daily seasonal period $z_2 = 12.5$. For convenience, a $K = 1$ is assumed at all times.

Lastly, when this method is applied alongside the sliding window sampling method, Equation (5.1) and Equation (5.2) have $x(j)$ swapped by $\varepsilon(j)$ since it no longer represents the original TSs' values but their residuals as the fitted HRs models (Figure 5.7) are subtracted from the original training sets.

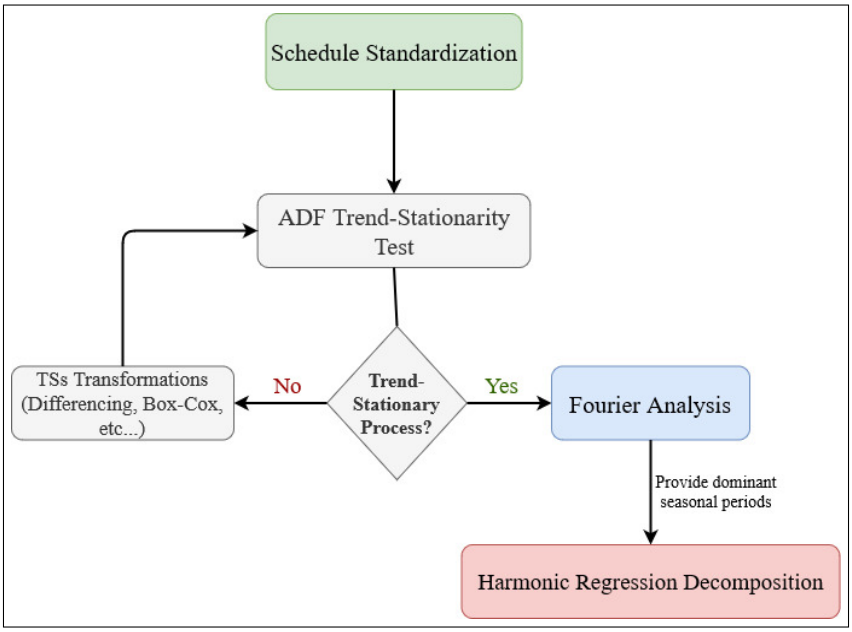

Figure 5.5: The constructed framework to apply the HR decomposition method.

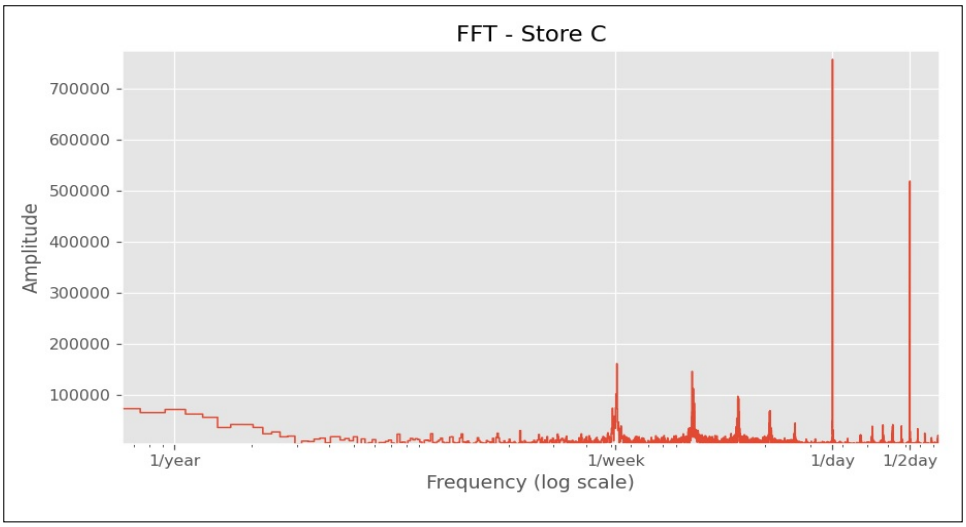

Figure 5.6: FFT plot of store C's third walk-forward iteration.

Table 5.6: ADF test results on store C training sets.

	Iteration 1	Iteration 2	Iteration 3
t-statistic	-29.747	-29.611	-29.747
Critical Value (1%)	-3.430	-3.430	-3.430
Critical Value (5%)	-2.862	-2.862	-2.862
Critical Value (10%)	-2.567	-2.567	-2.567

5.4 Technologies and Libraries

The forecasting models and related operations were developed using Python 3.7.9 with the following libraries:

- Pandas[1] which is frequently present in operations that regard to data preparation and preprocessing;
- Numpy[2] which provides efficient and optimized functions to deal with multidimensional arrays. It was used in various phases;
- Scikit-Learn[3] was used to apply the harmonic regression model, feature scaling and some of the evaluation metrics;
- TensorFlow[4], developed by the Google Brain Team, is an open-source library that acts as the primary support for many developed deep learning algorithms. It was initially implemented in *C++* but a native Python API is also available.

The Tensorflow module contains an API for the Keras[5] library which provides a user-friendly interface that allows for constructing various deep learning models. Additionally, Tensorflow includes an important algorithm to perform backpropagation called **automatic differentiation** (Autodiff). When executing the forward pass, a graph data structure that records each primitive operation is built. Then, during the backward pass a second graph that composes the gradient operations is automatically generated. The first graph is backtracked from the cost computation to the input in order to calculate the gradients with the chain rule. After each operation, the node that later computes the partial gradient for the same operation is added to the second graph (Figure 5.8).

[1] pandas.pydata.org
[2] numpy.org
[3] scikit-learn.org
[4] tensorflow.org
[5] keras.io

Figure 5.7: Fitted HR plot of store A's first walk-forward iteration training set.

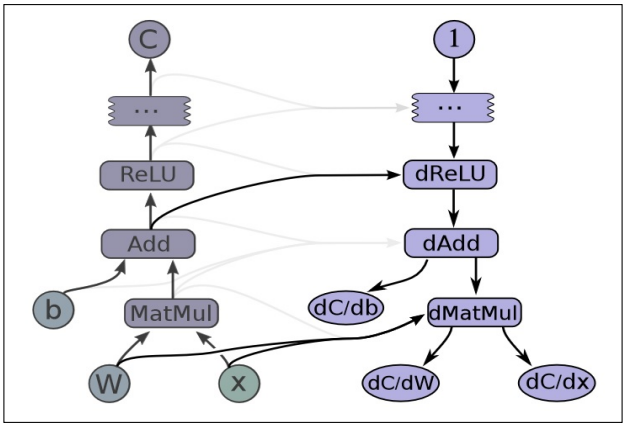

Figure 5.8: The graph on the right is the result of the Autodiff algorithm executed on the left operation (forward pass of a feedforward network). The cost C calculations are not shown for simplicity reasons. **Source**: [Abadi *et al.*, 2016]

5.5 Model Architectures and Configurations

For this experiment, the chosen models belong to the three main ANN classes (FNNs, RNNs and CNNs), these are:

1. The MLP;
2. Two LSTM architectures, a many-to-one and a many-to-many (Figure 3.10): LSTM-1 and LSTM-2;
3. The adapted WaveNet, SeriesNet.

Additionally, due to the popularity of RNNs in the forecasting field and because of the recent results presented in [Siami-Namini *et al.*, 2019], two BiLSTM architectures named BiLSTM-1 (many-to-one) and BiLSTM-2 (many-to-many) were also included.

Regarding the forecasting strategies, all the models will be following a direct strategy except for SeriesNet since the WaveNet's architecture is inherently recursive. The models that follow the direct strategy are trained with the sliding window sampling method for

50 training iterations (epochs) with early stopping ($patience = 5$) to prevent overfitting. Of those particular models, the ones that have architectures that allow it are tested with four different input vector sizes p, i.e., $\{10, 25, 100, 175\}$ (input windows of 5 hours up to a week). In parallel, SeriesNet, due to its fast computation was experimented with one more overall receptive field size r (Equation (4.3)), i.e., $\{8, 32, 128, 256, 768\}$. SeriesNet remained with the majority of its original configuration (subsection 4.2.2) having only its number of layers L and K altered to possibilitate testing with the different r. Early stopping ($patience = 500$) was also included in SeriesNet's training. The high *patience* threshold is justified by its slow training convergences and due to the training being computed for 8000 epochs.

Except when using SeriesNet[6], all training sets were scaled by subtracting the mean and diving the standard deviation. This is a common practice in many ML models as it helps gradient descent optimizers converge faster ([Ioffe and Szegedy, 2015]).

For the remaining models' hyperparameters, rules of thumb were followed and small manual adjustments were made. If a hyperparameter has not been discussed its because it was assigned with its default value as defined by Keras. The next subsections will further describe the architectures and better detail each configuration.

5.5.1 Baseline Model

The baseline is simply a naive model that uses the last H observations in each training set as the H-step forecasts.

5.5.2 MLP

The chosen MLP architecture is a three layer FNN (input, hidden and output layer) as three layers are, theoretically, sufficient to approximate any function (Figure 3.2). ReLU was the chosen activation function for the hidden layer due to its fast and simple computation ([Krizhevsky *et al.*, 2017]). As will be seen in some of the other models ahead, the output layer is a linear fully-connected layer with H output units. Therefore, a forecast $\hat{\boldsymbol{y}}_t$ (Figure 5.9) is described by,

$$\underset{H\times 1}{\hat{\boldsymbol{y}}_t} = \underset{H\times units}{\boldsymbol{W}_2}\ \boldsymbol{h} + \underset{H\times 1}{\boldsymbol{b}_2} \tag{5.3}$$

where $\boldsymbol{h}$ is given by

$$\underset{units\times 1}{\boldsymbol{h}} = ReLU\left(\underset{units\times p}{\boldsymbol{W}_1}\ \underset{p\times 1}{\boldsymbol{x}} + \underset{units\times 1}{\boldsymbol{b}_1}\right) \tag{5.4}$$

and the number of hidden units ($units$) had its value decided by a rule of thumb called the geometric pyramid rule, proposed in [Masters, 1993]. It states that the hidden layer of a three-layer FNN should have $units = \sqrt{n \times j}$ where n is the size of the input, in this

[6] In [Papadopoulos, 2018], it is stated that different scaling methods do not have any impact in forecasts.

case $n = p$, and j[7] is the size of the output ($j = H$). Knowing this, the total number of

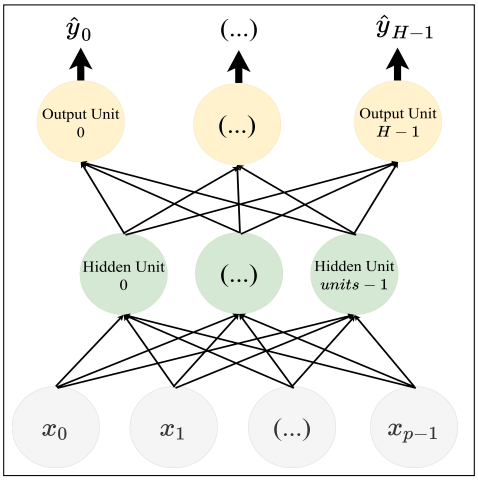

Figure 5.9: The MLP architecture for multi-step forecasting.

learnable parameters (weights and bias) of the MLP can be calculated as follows,

$$\text{Total parameters (MLP)} = p \times \sqrt{p \times H} + \sqrt{p \times H} \times H + \sqrt{p \times H} + H \tag{5.5}$$

Taking into account the horizons H shown in Table 5.1 and, for example, assuming that $p = 25$, Table 5.7 displays the number of learnable parameters of the MLP used on the raw setting. On standardized versions, since the horizon is always $H = 750$, there are 107062 ($units = 137$) weights plus biases. This was done for each of the defined p values.

Table 5.7: Number of learnable parameters of the MLP used on the raw setting.

$p = 25$	Iteration 1	Iteration 2	Iteration 3
Store A	**93958** ($units = 131$)	**96399** ($units = 132$)	**95468** ($units = 132$)
Store B	**107062** ($units = 137$)	**106648** ($units = 137$)	**106510** ($units = 137$)
Store C	**95601** ($units = 132$)	**96133** ($units = 132$)	**95867** ($units = 132$)

5.5.3 LSTM-1 and BiLSTM-1

LSTM-1 and BiLSTM-1 present three layers: an input layer, a LSTM/BiLSTM layer and a linear fully-connected output layer.

Similarly to the MLP, LSTM-1 and BiLSTM-1 have a fully-connected output layer with H units and because of that the forecasts can be described by Equation (5.3). However, $\boldsymbol{h}$ is now different since it is computed in a LSTM/BiLSTM layer:

1. In the LSTM-1 (Figure 5.10), it refers to the output gotten from the **last processing step** of a LSTM unit (Equation (3.22)), i.e., $\underset{units \times 1}{\boldsymbol{h}_t}$;

[7] Not to be confused with the j mentioned in subsection 5.3.1.

2. In the BiLSTM-1 (Figure 5.11), it refers to the output gotten from the **last processing step** of a standard LSTM unit that was element-wise concatenated[8] with the output obtained from a **backwards** LSTM unit $\underset{units\times 1}{\overleftarrow{\boldsymbol{h}}_t}$ (Equation (3.16)).

This type of architecture is trendier in classification problems, such as sentiment analysis ([Rao *et al.*, 2018]).

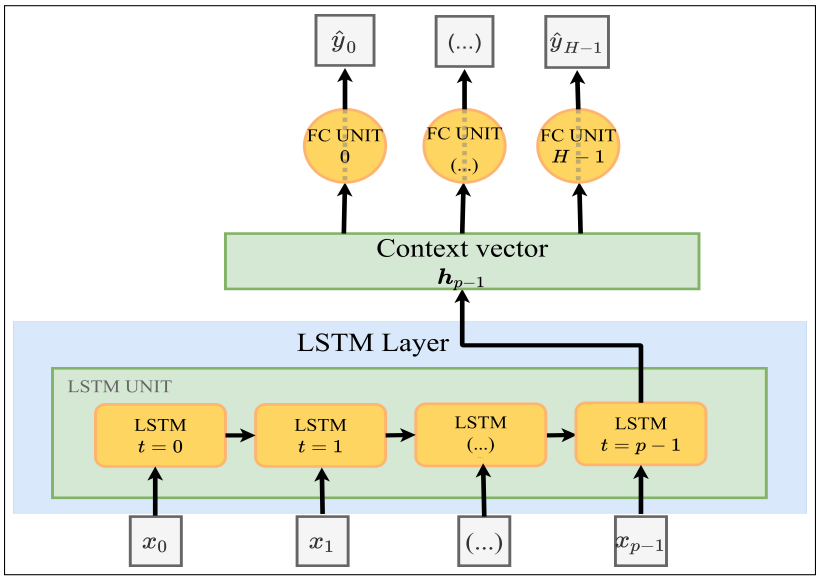

Figure 5.10: The LSTM-1 (many-to-one) architecture.

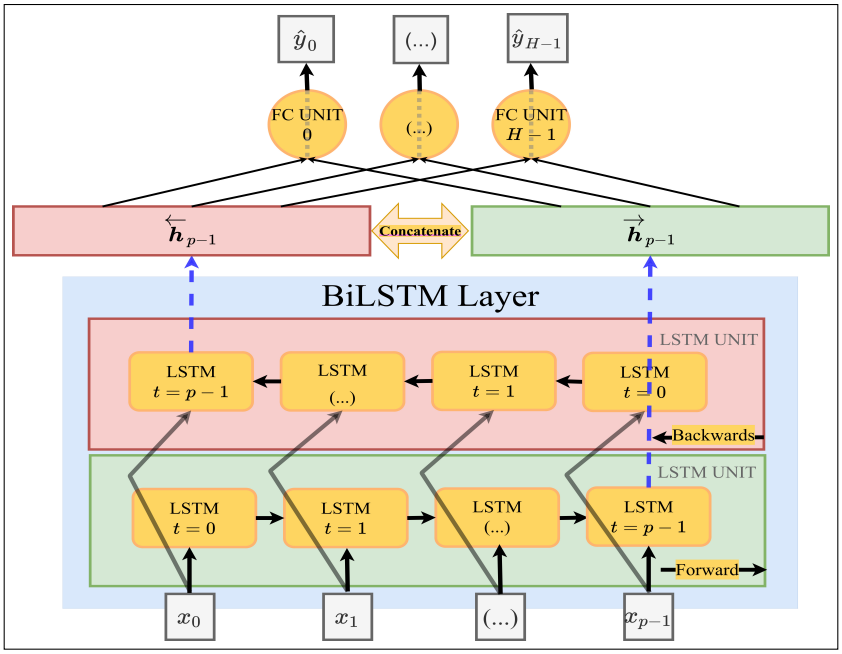

Figure 5.11: The BiLSTM-1 (many-to-one) architecture.

In RNN architectures, the parameter *units* refers to the size of the output/hidden

[8] This is Keras' default merge mode.

state ($\boldsymbol{h}_t$) and affects the memory capabilities of RNNs. In this context, the total number of weights and biases of the LSTM-1 is given by

$$\text{Total parameters (LSTM-1)} = 4 \times (2 \times units + units^2) + units \times H + H \tag{5.6}$$

and of the BiLSTM-1, since there are two LSTM units, is given by

$$\text{Total parameters (BiLSTM-1)} = 8 \times (2 \times units + units^2) + 2 \times units \times H + H \tag{5.7}$$

In order to build a comparison platform, the number of *units* was picked in an attempt to match the capacity of the MLP ([Alam, 2018]), i.e., its total number of learnable parameters. Tables A.2 and A.3 (Appendix A) show the number of parameters of both LSTM-1 and BiLSTM-1 used on the raw data setting with the same configuration ($p = 25$) that was presented in the previous subsection. On the standardized setting, the LSTM-1 has 107346 parameters ($units = 94$) and the BiLSTM-1 has 105942 ($units = 54$). As expected, this was also done for every p value.

5.5.4 LSTM-2 and BiLSTM-2

In comparison to the LSTM-1 and the BiLSTM-1, LSTM-2 (Figure 5.12) and BiLSTM-2 (Figure 5.13) have two main differences:

1. The LSTM units output their hidden state ($\boldsymbol{h}_t$) at **each step** and not only at the last;

2. The fully-connected output layer only has one unit and it is **time distributed**.

In both architectures, time distributed means that they forecast one step at a time with the output unit sharing the same weights through all steps, hence, reducing the overall number of parameters. However, this does not result in faster trainings since in order to produce a H-step forecast a constraint of $p = H$ is placed which results in larger training windows.

As expected, the total number of parameters of the LSTM-2 is given by

$$\text{Total parameters (LSTM-2)} = 4 \times (2 \times units + units^2) + units + 1 \tag{5.8}$$

and of the BiLSTM-2 is given by

$$\text{Total parameters (BiLSTM-2)} = 8 \times (2 \times units + units^2) + 2 \times units + 1 \tag{5.9}$$

To maintain the comparison platform, both architectures' number of *units* took into account the number of parameters that the MLP with $p = H$ would, theoretically, have if it forecasted one step at a time (Tables A.4, A.5 and A.6). Knowing this, in the standardized setting the LSTM-2 has always 20231 parameters ($units = 70$) and the BiLSTM-2 has 20091 parameters ($units = 49$).

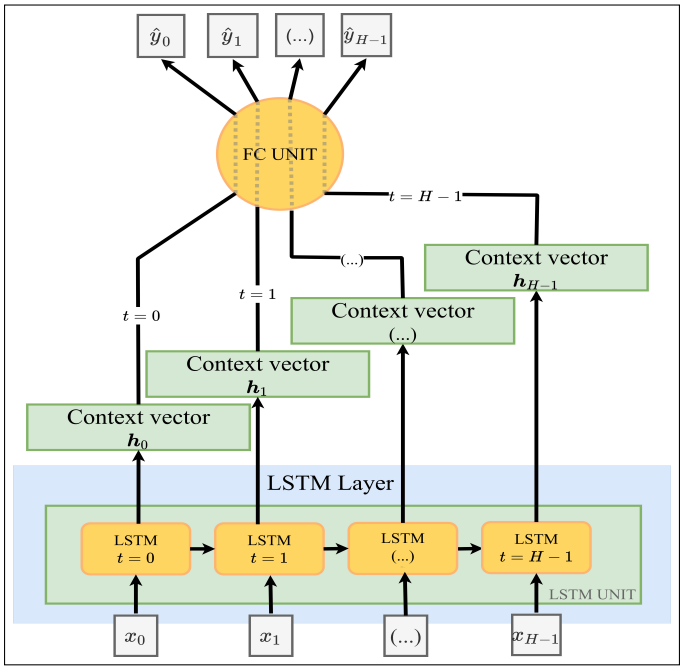

Figure 5.12: The LSTM-2 (many-to-many) architecture.

5.5.5 SeriesNet

As mentionted, SeriesNet will mostly follow its original architecture and configuration (subsection 4.2.2) only having a few differences in its number of layers L and filter sizes K when different receptive field sizes r are desired. Additionally, when $r = 8$ and $r = 32$ the dropout's *prob* is halved to 40% because of the configurations' already reduced capacity.

Similarly to the previous LSTM architectures and in contrast to the MLP, the number of parameters of SeriesNet is not directly dependent of the input and output sizes since the weights represent the preconfigured filters[9]. Additionally, SeriesNet is able to drastically reduce its amount of weights due to the parameter sharing feature CNNs present. The total number of parameters of SeriesNet is mathematically portrayed by

$$\text{Total parameters (SeriesNet)} = L \times (K + 1) \times F + (L - 1) \times F + 1 \tag{5.10}$$

Table A.7 presents, for every r, the number of layers L, the number of filters K and the total number of parameters of the SeriesNet on both unstandardized and standardized settings.

5.6 Summary

In essence, six ANN models will perform, against a baseline naive model, on three client flow TSs having three different settings by forecasting instances of 30 days. The first setting is a raw version, meaning that there is no data preprocessing. The second uses

[9] As per the original, no biases were included.

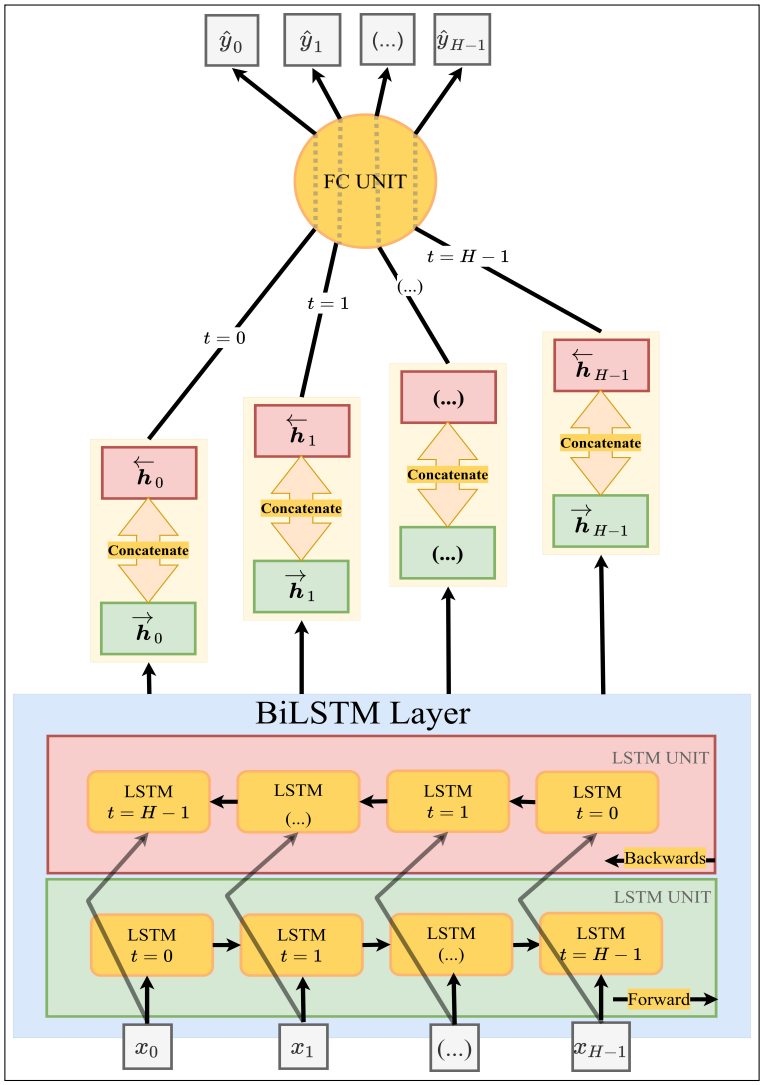

Figure 5.13: The BiLSTM-2 (many-to-many) architecture.

a standardization method for the stores' schedules. The third uses on top of the second a HR classical decomposition method.

Five of the six models, a MLP, two different LSTMs and two different BiLSTMs, use the direct multi-step strategy and project their training with the sliding window method. These are tested, therefore, with four different numbers of lagged values p as inputs. In order to compare performances, the last four models have their configurations adjusted to the number of parameters of the MLP. The sixth model, SeriesNet, has similar configuration to its original ([Papadopoulos, 2018]). It uses the recursive multi-step strategy, hence, it does not employ the sliding window method but it is tested with five different overall receptive field sizes r.

The next chapter will provide the objective results and findings of the experimentations.

Chapter 6

Results and Discussion

As stated in the previous chapter, the metrics obtained in each walk-forward iteration are averaged in order to present the overall performances. Also, the corresponding models were tested with the defined sliding window input lengths p (receptive field sizes r for the SeriesNet) along with each data setting, again, the settings are:

1. The raw or unstandardized setting (**RAW**);
2. The standardized setting (**STD**);
3. The standardized plus deseasonalization and detrending with HR (**STD+HR**) setting.

The performances are then displayed in nine tables (three settings × three store datasets). The first three tables (Table 6.1, 6.2 and 6.3) show the performances produced with the **RAW** setting. The second trio's results (Table 6.4, 6.5 and 6.6) were obtained with the **STD** setting. The third and last trio (Table 6.7, 6.8 and 6.9) has its results gotten with the **STD+HR** setting. The best models are highlighted in bold.

Table 6.1: Store A's raw setting results.

RAW	MAE	RMSE	MAAPE (%)
Naive	16.410	20.357	50.133
MLP ($p = 25$)	11.228	13.710	40.147
LSTM-1 ($p = 25$)	**10.986**	**13.661**	**38.353**
BiLSTM-1 ($p = 25$)	11.137	13.781	39.281
LSTM-2 ($p = H$)	12.670	15.065	44.336
BiLSTM-2 ($p = H$)	13.431	16.044	45.743
SeriesNet ($r = 128$)	14.859	19.029	44.994

Despite being the TS that goes through the most "gentle" standardization process

Table 6.2: Store B's raw setting results.

RAW	MAE	RMSE	MAAPE (%)
Naive	10.099	13.038	41.691
MLP ($p = 25$)	6.348	7.844	37.098
LSTM-1 ($p = 175$)	**5.992**	**7.549**	**34.326**
BiLSTM-1 ($p = 100$)	6.009	7.704	34.477
LSTM-2 ($p = H$)	7.089	8.817	39.732
BiLSTM-2 ($p = H$)	6.874	8.560	38.889
SeriesNet ($r = 768$)	6.194	7.937	34.614

Table 6.3: Store C's raw setting results.

RAW	MAE	RMSE	MAAPE (%)
Naive	25.764	31.877	42.584
MLP ($p = 175$)	**15.117**	**18.738**	29.191
LSTM-1 ($p = 100$)	15.377	19.699	**28.636**
BiLSTM-1 ($p = 175$)	16.113	20.278	29.617
LSTM-2 ($p = H$)	18.660	22.258	34.850
BiLSTM-2 ($p = H$)	20.043	23.870	37.284
SeriesNet ($r = 8$)	20.103	23.612	38.381

Table 6.4: Store A's standardized setting results.

STD	MAE	RMSE	MAAPE (%)
Naive	8.498	11.316	30.607
MLP ($p = 25$)	5.128	6.725	20.020
LSTM-1 ($p = 25$)	**5.062**	**6.617**	**19.351**
BiLSTM-1 ($p = 25$)	5.089	6.684	19.748
LSTM-2 ($p = H$)	6.547	8.633	24.620
BiLSTM-2 ($p = H$)	5.419	7.232	20.766
SeriesNet ($r = 768$)	5.238	6.898	19.508

(smallest number of imputations), the performances in store B's dataset show the worst results of all three stores. One possible reason is because of the TS having the lowest

Table 6.5: Store B's standardized setting results.

STD	MAE	RMSE	MAAPE (%)
Naive	5.712	7.319	31.471
MLP ($p = 175$)	**3.988**	**4.953**	26.118
LSTM-1 ($p = 100$)	4.206	5.248	26.966
BiLSTM-1 ($p = 100$)	4.141	5.232	26.224
LSTM-2 ($p = H$)	5.027	6.475	29.471
BiLSTM-2 ($p = H$)	4.536	5.766	28.092
SeriesNet ($r = 128$)	4.023	5.138	**24.234**

Table 6.6: Store C's standardized setting results.

STD	MAE	RMSE	MAAPE (%)
Naive	16.561	23.067	29.445
MLP ($p = 175$)	**9.149**	**12.646**	**18.793**
LSTM-1 ($p = 175$)	10.547	14.722	20.678
BiLSTM-1 ($p = 175$)	10.966	15.310	21.624
LSTM-2 ($p = H$)	13.841	19.151	25.864
BiLSTM-2 ($p = H$)	14.350	19.905	26.578
SeriesNet ($r = 768$)	13.575	18.542	25.835

Table 6.7: Store A's standardized plus HR setting results.

STD+HR	MAE	RMSE	MAAPE (%)
Naive	8.498	11.316	30.607
MLP ($p = 25$)	**5.047**	**6.625**	**19.729**
LSTM-1 ($p = 10$)	5.273	6.883	20.497
BiLSTM-1 ($p = 25$)	5.295	6.900	20.228
LSTM-2 ($p = H$)	5.672	7.344	21.831
BiLSTM-2 ($p = H$)	6.027	7.836	23.041
SeriesNet ($r = 8$)	5.353	6.980	20.617

average of client entries and being highly fluctuant. Such nature is hinted in Figure 5.1, where it can be seen that the other two stores have a more regular behaviour. Another

Table 6.8: Store B's standardized plus HR setting results.

STD+HR	MAE	RMSE	MAAPE (%)
Naive	5.712	7.319	31.471
MLP ($p = 175$)	**4.165**	**5.168**	27.049
LSTM-1 ($p = 175$)	4.341	5.472	**26.963**
BiLSTM-1 ($p = 25$)	4.265	5.319	27.219
LSTM-2 ($p = H$)	4.713	5.956	29.423
BiLSTM-2 ($p = H$)	5.130	6.475	31.104
SeriesNet ($r = 8$)	4.615	5.854	27.997

Table 6.9: Store C's standardized plus HR setting results.

STD+HR	MAE	RMSE	MAAPE (%)
Naive	16.561	23.067	29.445
MLP ($p = 175$)	**9.399**	**13.063**	**19.002**
LSTM-1 ($p = 175$)	11.844	16.090	23.298
BiLSTM-1 ($p = 175$)	9.659	13.402	19.289
LSTM-2 ($p = H$)	14.496	20.483	26.827
BiLSTM-2 ($p = H$)	13.365	17.486	25.618
SeriesNet ($r = 128$)	12.339	18.332	23.002

justification can be the fact that it is the dataset with the largest horizons/test sets (Table 5.1).

Regarding the metrics, it is worth mentioning that there are instances where the model that had both the best MAE and RMSE is different from the model that had the best MAAPE. This is because of MAAPE having low sensitivity to large forecast errors. An example is in Table 6.5, where SeriesNet despite having obtained worse MAE and RMSE it had better MAAPE than the MLP (Figure 6.1).

By shifting the focus towards the models and their configurations various conclusions can be drawn:

1. All DL models performed better than the naive model;

2. The LSTM many-to-one models had significantly better results than the many-to-many architectures (Figure 6.2) in spite of them being trained with shorter input windows ($p < H$). The reason for that may lie in the architecture of both LSTM-2 and BiLSTM-2, where every $\boldsymbol{h}_t$ outputted in the early processing steps has

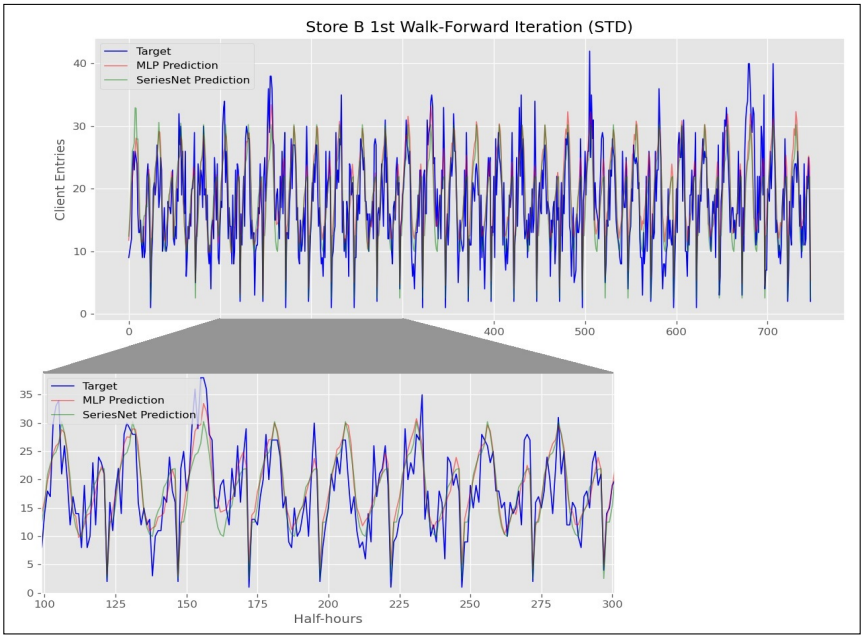

Figure 6.1: 748-step forecasts done by the best MLP and SeriesNet (as in Table 6.5).

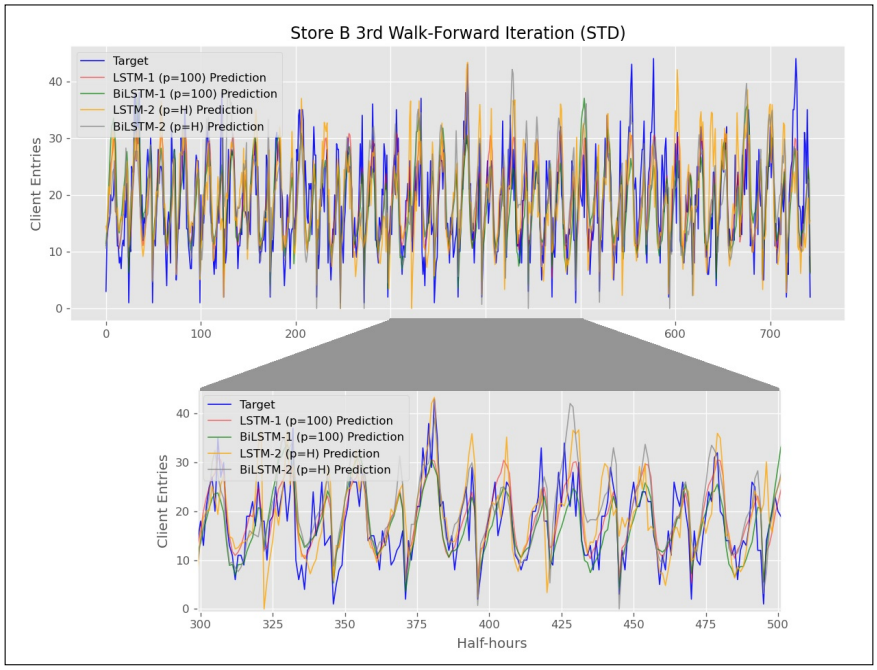

Figure 6.2: 743-step forecasts done by the LSTM many-to-many (best) and many-to-one (worst) models.

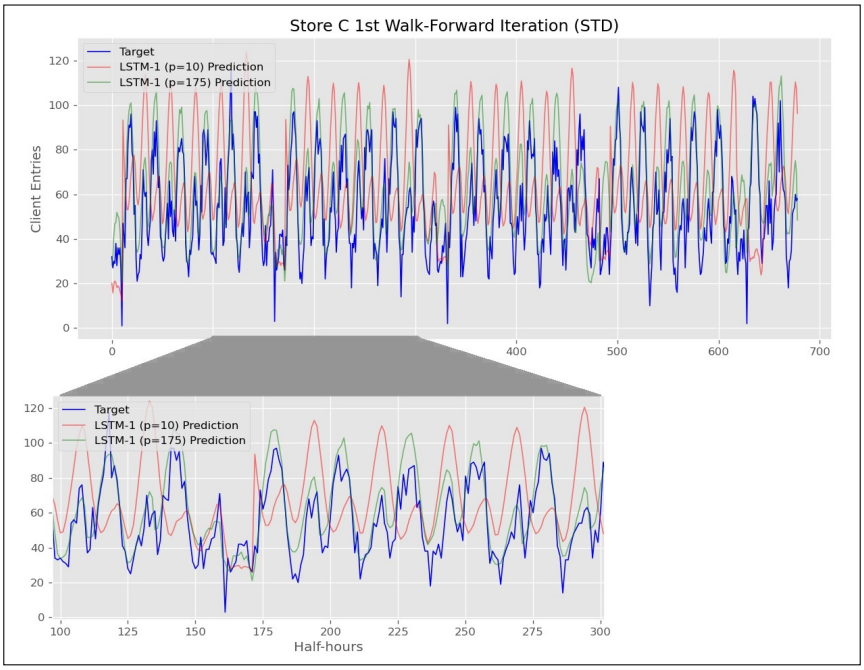

Figure 6.3: 679-step forecasts done by LSTM-1 with $p = 10$ (worst) and with $p = 175$ (best).

little to no past contextual information;

3. The MLP, LSTM-1 and BiLSTM-1 were the best models overall and consistently demonstrated close performances;

4. SeriesNet models were able to remain competitive despite them having a small amount of learnable parameters;

5. The expansion of the training sets with each walk-forward iteration does not directly translate in better performances (Table 6.10);

6. No correlation could be seen between SeriesNet's r and the sliding window input window length p, however, as the best both had smaller values in store A's STD+HR setting;

7. The MLP, LSTM-1 and BiLSTM-1 best p in store B and, especially, store C suggests that there is a longer seasonal period (probably weekly, as seen in Figure 5.6) that was not handled by the HR;

8. In alignment with the previous point, the crude deduction that p should be greater or at least equal to the longest relevant seasonal period can be made (Figure 6.3). Although, this is not necessary if the HR fits the data well (e.g., LSTM-1 having $p = 10$ as its best in Table 6.7).

The last two statements, however, require more experimentation.

Table 6.10: The best performance of BiLSTM-1 on store B's raw setting.

$p = 100$	MAE	RMSE	MAAPE
Iteration 1	6.5235	6.395	31.242
Iteration 2	6.502	8.529	35.890
Iteration 3	6.487	8.187	36.298

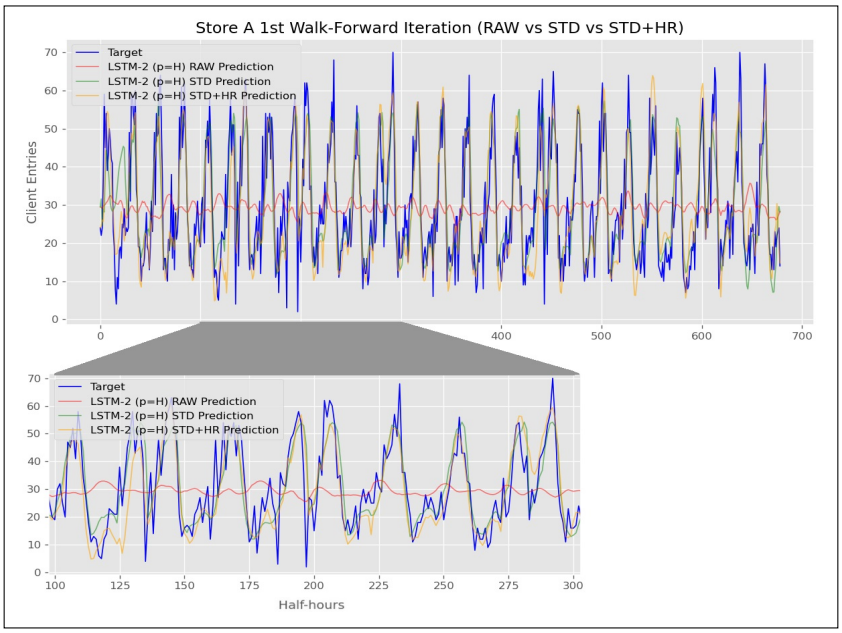

Figure 6.4: 679-step forecasts done by LSTM-2 with the three different data settings.

Finally, by cross-checking the results between the data settings it can be seen that, while the results obtained with standardized settings were greatly superior to their raw alternatives (Figure 6.4) the deseasonalization and detrending that was conducted with the HR preprocessing method was not, in general, benefitial.

Chapter 7

Conclusion and Work Ahead

As ML and DL quickly evolve and naturally emerge in many fields, this dissertation pursued to bring some of the most popular techniques to the world of retail with high-frequency client flow TS forecasting. By merging classical TS concepts with modern DL models, specifically, ANNs, it aimed at giving insight on how both areas can contribute to the problem. It also focused on comparing and thoroughly describing multiple ANN architectures with various training configurations and two fundamental multi-step forecasting strategies.

Up until Chapter 5, various works were studied to theoretically support the development and exploration of the various DL methods and classical TS techniques. Namely, ANN architectures that belong to three primary classes, such as FNNs, CNNs and RNNs. With the MLP being one of the simplest architectures and RNNs, such as the LSTM, being considered the state of the art for sequence problems, in Chapter 4 emphasis was placed on understanding how CNNs, networks that are popular for image-related problems, could be used for TS forecasting. CNN architectures that were greatly inspired in WaveNet, an audio generation model, called Augmented WaveNet and SeriesNet were detailed and the latter was tested later on.

About the classical TS techniques, stationarity and decomposition are considered fulcral notions in the classical TS theory, hence, were the chosen ones for the experiments. More specifically, a framework (Figure 5.5) was developed where a ADF trend-stationarity test gives the green light, or not, for Fourier analysis which in turn contributes with relevant seasonal periods to a HR transform method that aims at easing the models' forecasting task. Unfortunately, all the TSs present in the experiments were originated by trend-stationary processes, therefore, the framework was not fully tested.

In Chapter 5, the case study which consisted in three real world retail client flow TSs was portrayed. As expected, the data displayed many inconsistencies, especially, the stores' working schedules. In TS forecasting, it is assumed that the TSs are regular, thus, a schedule standardization method was proposed and, in Chapter 6, it was possible to see the great improvements that such process brought.

Although no extensive hypertuning was done, a rule of thumb to possibilitate the

comparison between models was followed. It consisted in constraining the architectures to have the closest number of trainable parameters. Additionally, different training configurations with the sliding window sampling method and SeriesNet's overall receptive field size were experimented. The HR transformation expressed an effect on those configurations but, in the end, it did not provide any benefit. This fact somewhat supports the claim that *"(...) the neural network model form, with its nonlinear threshold functions has the ability to fit almost any kind of nonlinearity"* in TSs ([Gorr, 1994]), such as multiple seasonalities or trends.

With the exception of SeriesNet in some cases, the tested networks are considered shallow since they possess a small number of hidden layers. Yet, all were competitive, better than the baseline naive model and in various cases able to show forecasting skill (Figure 6.1, Figure 6.2, Figure 6.3 and Figure 6.4). Having up to 8% better MAAPE than the rest, the best performing models were the MLP and two LSTM many-to-one architectures.

As predicting the future always has a degree of uncertainty, research on TS forecasting is considered to be a never ending task. Even more, when the applied technologies were not built with the problem in mind, such as DL models. Therefore, for works that may inherit this research some suggestions are given:

- Hypertuning that allows extra learning capacity and depth to the models should be done in order to obtain preciser verdicts;
- Multiple models could be trained with different working schedules and combined in order to remove the drawback of forcing the forecasts to be for a restricted schedule. Also, different interpolation methods should be experimented within the standardization process;
- The HR can be further optimized by taking into account more, or less, seasonal periods (z) discovered through Fourier analysis and, by testing with K values that allow for more flexibility with fluctuating behaviours in TSs;
- Besides SeriesNet, the recursive multi-step strategy can also be experimented with other architectures;
- The sliding window sampling method can be additionally scrutinized with many more input lengths (p) and different strides (s);
- The models should be tested on TSs that are not trend-stationary in order to see if trend-stationarity is a relevant quality for ANN forecasting models.

References

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv: 1603.04467 [cs.DC].

Adya, Monica, Fred Collopy, J Scott Armstrong, and Miles Kennedy (2001). "Automatic identification of time series features for rule-based forecasting." In: *International Journal of Forecasting* 17.2, pp. 143–157. ISSN: 01692070. DOI: 10.1016/S0169-2070(01)00079-6.
URL: www.elsevier.com/locate/ijforecast.

Ahmed, Nesreen K., Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny (2010). "An empirical comparison of machine learning models for time series forecasting." In: *Econometric Reviews* 29.5, pp. 594–621. ISSN: 07474938. DOI: 10.1080/07474938.2010.481556.

Alam, Samiul (2018). "Recurrent neural networks in electricity load forecasting." PhD thesis, p. 54.
URL: https://pdfs.semanticscholar.org/d6fa/f3f8c760a0e8a796bc6ab7dda847b83ca45d.pdf?_ga=2.94699679.240483960.1598370716-131206575.1581295419.

An, Nguyen Hoang and Duong Tuan Anh (2015). "Comparison of Strategies for Multi-step-Ahead Prediction of Time Series Using Neural Network." In: *Proceedings - 2015 International Conference on Advanced Computing and Applications, ACOMP 2015*, pp. 142–149. DOI: 10.1109/ACOMP.2015.24.

Bao, Wei, Jun Yue, and Yulei Rao (2017). "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." In: *PLoS ONE* 12.7. ISSN: 19326203. DOI: 10.1371/journal.pone.0180944.

Bayer, Justin, Daan Wierstra, Julian Togelius, and Jürgen Schmidhuber (2009). "Evolving Memory Cell Structures for Sequence Learning." In: *Proceedings of the 19th International Conference on Artificial Neural Networks: Part II*. ICANN '09. Limassol, Cyprus: Springer-Verlag, pp. 755–764. ISBN: 9783642042768. DOI: 10.1007/978-3-642-04277-5_76.
URL: https://doi.org/10.1007/978-3-642-04277-5_76.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). "Learning Long-Term Dependencies with Gradient Descent is Difficult." In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. ISSN: 19410093. DOI: 10.1109/72.279181.

Bontempi, Gianluca, Souhaib Ben Taieb, and Yann-Aël Le Borgne (2013). "Machine Learning Strategies for Time Series Forecasting." In: *eBISS 2012. Lecture Notes in Business Information Processing*. Aufaure M, vol. 138. Brussels, Belgium: Springer, Berlin, Heidelberg, pp. 62–77. DOI: 10.1007/978-3-642-36318-4_3.
URL: http://mlg.ulb.ac.be.

Borovykh, Anastasia, Sander Bohte, and Cornelis W. Oosterlee (2018). "Dilated convolutional neural networks for time series forecasting." In: *Journal of Computational Finance* March 2017. ISSN: 14601559. DOI: 10.21314/jcf.2019.358.

Box, G. E. P. and D. R. Cox (1964). "An Analysis of Transformations." In: *Journal of the Royal Statistical Society: Series B (Methodological)* 26.2. DOI: 10.1111/j.2517-6161.1964.tb00553.x.

Box, G.E.P., G.M. Jenkins, and H. Day (1976). *Time Series Analysis: Forecasting and Control*. Holden-Day series in time series analysis and digital processing. San Francisco: Holden-Day. ISBN: 9780816211043.

Brockwell, Peter J. and Richard A. Davis (2016). *Introduction to Time Series and Forecasting*. 3rd Edition. Springer-Verlag. ISBN: 978-3-319-29852-8. DOI: 10.1007/978-3-319-29854-2.

Bunn, Derek and George Wright (1991). "Interaction of Judgemental and Statistical Forecasting Methods: Issues & Analysis." In: *Management Science* 37.5, pp. 501–518. ISSN: 0025-1909. DOI: 10.1287/mnsc.37.5.501.

Chambers, John C., Satinder K. Mullick, and Donald D. Smith (1971). "How to choose the right forecasting technique." In: *Harvard Business Review* 49.4, pp. 45–70. ISSN: 00178012.

CIF - Computational Intelligence in Forecasting (2021).
URL: https://irafm.osu.cz/cif2015/main.php (visited on 04/12/2021).

Clemen, Robert T (1989). "Combining forecasts: A review and annotated bibliography." In: *International Journal of Forecasting* 5.4, pp. 559–583. ISSN: 01692070. DOI: 10.1016/0169-2070(89)90012-5.

Convolutions in Autoregressive Neural Networks (2019).
URL: https://theblog.github.io/post/convolution-in-autoregressive-neural-networks/ (visited on 03/28/2021).

Cui, Zhiyong, Ruimin Ke, Ziyuan Pu, and Yinhai Wang (2018). "Stacked bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction." In: *arXiv*, pp. 1–11. ISSN: 23318422. arXiv: 1801.02143.

Cybenko, G. (1989). "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals, and Systems* 2.4, pp. 303–314. ISSN: 09324194. DOI: 10.1007/BF02551274.

Dauphin, Yann N., Angela Fan, Michael Auli, and David Grangier (2017). "Language modeling with gated convolutional networks." In: *34th International Conference on Machine Learning, ICML 2017* 2, pp. 1551–1559. arXiv: 1612.08083.

De Gooijer, Jan G. and Rob J. Hyndman (2006). "25 years of time series forecasting." In: *International Journal of Forecasting* 22.3, pp. 443–473. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2006.01.001.

Dietterich, Thomas G. (2002). "Machine Learning for Sequential Data: A Review." In: *Structural, Syntactic, and Statistical Pattern Recognition*. Ed. by Terry Caelli, Adnan Amin, Robert P. W. Duin, Dick de Ridder, and Mohamed Kamel. Berlin, Heidelberg:

Springer Berlin Heidelberg, pp. 15–30. ISBN: 978-3-540-70659-5.

Dong, Zibo, Dazhi Yang, Thomas Reindl, and Wilfred M. Walsh (2013). "Short-term solar irradiance forecasting using exponential smoothing state space model." In: *Energy* 55, pp. 1104–1113. ISSN: 03605442. DOI: 10.1016/j.energy.2013.04.027. URL: http://dx.doi.org/10.1016/j.energy.2013.04.027.

Dumoulin, Vincent and Francesco Visin (2016). "A guide to convolution arithmetic for deep learning." In: pp. 1–31. arXiv: 1603.07285. URL: http://arxiv.org/abs/1603.07285.

Edmundson, R H (1990). "Decomposition; a strategy for judgemental forecasting." In: *Journal of Forecasting* 9.4, pp. 305–314. ISSN: 1099131X. DOI: 10.1002/for.3980090403.

Fisher, Marshall and Ananth Raman (Sept. 2018). "Using Data and Big Data in Retailing." In: *Production and Operations Management* 27.9, pp. 1665–1669. DOI: 10.1111/poms.12846.

Forecasting - Exponential Smoothing (2017). URL: http://www.edscave.com/forecasting---exponential-smoothing.html (visited on 12/16/2020).

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological Cybernetics* 36.4, pp. 193–202. ISSN: 03401200. DOI: 10.1007/BF00344251.

Fuller, Wayne (1996). *Introduction to Statistical Time Series*. Ed. by N. Fisher et al. V. Barnet R. Bradley. 2nd Edition. Iowa State University: Wiley-Interscience. ISBN: 0-471-55239-9.

Gardner, Everette S. (1985). "Exponential smoothing: The state of the art." In: *Journal of Forecasting* 4.1, pp. 1–28. DOI: 10.1002/for.3980040103. URL: https://doi.org/10.1002/for.3980040103.

Gaur, Vishal, Saravanan Kesavan, and Ananth Raman (2014). "Retail Inventory: Managing the Canary in the Coal Mine!" In: *California Management Review* 56.2, pp. 55–76.

Gers, Felix A., Jürgen A. Schmidhuber, and Fred A. Cummins (Oct. 2000). "Learning to Forget: Continual Prediction with LSTM." In: *Neural Comput.* 12.10, pp. 2451–2471. ISSN: 0899-7667. DOI: 10.1162/089976600300015015. URL: https://doi.org/10.1162/089976600300015015.

Gorr, Wilpen L. (1994). "Editorial: Research prospective on neural network forecasting." In: *International Journal of Forecasting* 10.1, pp. 1–4. ISSN: 0169-2070. DOI: https://doi.org/10.1016/0169-2070(94)90044-2. URL: http://www.sciencedirect.com/science/article/pii/0169207094900442.

Graves, Alex and Jürgen Schmidhuber (2005). "Framewise phoneme classification with bidirectional LSTM and other neural network architectures." In: *Neural Networks* 18.5-6, pp. 602–610. ISSN: 08936080. DOI: 10.1016/j.neunet.2005.06.042.

Green, Kesten C. and J. Scott Armstrong (2007). "Structured analogies for forecasting." In: *International Journal of Forecasting* 23.3, pp. 365–376. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2007.05.005.

Güneş Baydin, Atılım, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind (2018). *Automatic differentiation in machine learning: A survey*. arXiv: 1502.05767.

Ha, S. and S. Choi (2016). "Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors." In: *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 381–388. DOI: 10.1109/IJCNN.2016.7727224.

Hamilton, James D. (1994). *Time Series Analysis*. 1st Edition. Princeton University Press, pp. 528–529. ISBN: 0691042896.

Hamzaçebi, Coşkun, Diyar Akay, and Fevzi Kutay (2009). "Comparison of direct and iterative artificial neural network forecast approaches in multi-periodic time series forecasting." In: *Expert Systems with Applications* 36.2 PART 2, pp. 3839–3844. ISSN: 09574174. DOI: 10.1016/j.eswa.2008.02.042.

Han, Song, Huizi Mao, and William J. Dally (2016). "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding." In: *4th International Conference on Learning Representations, ICLR 2016 - Conference*

Track Proceedings, pp. 1–14. arXiv: 1510.00149.

Harvey, Nigel and Claire Harries (July 2004). "Effects of judges' forecasting on their later combination of forecasts for the same outcomes." In: *International Journal of Forecasting* 20.3, pp. 391–409. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2003.09.012.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2016-December, pp. 770–778. ISSN: 10636919. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov (2012). "Improving neural networks by preventing co-adaptation of feature detectors." In: pp. 1–18. arXiv: 1207.0580.
URL: http://arxiv.org/abs/1207.0580.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory." In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735.
URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Hogarth, Robin M. and Spyros Makridakis (Feb. 1981). *Forecasting and planning: An evaluation*. DOI: 10.1287/mnsc.27.2.115.

Holt, Charles C. (1957). "Forecasting seasonals and trends by exponentially weighted moving averages." In: *International Journal of Forecasting* 20.1, pp. 5–10. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2003.09.015.
URL: http://www.sciencedirect.com/science/article/pii/S0169207003001134.

Hopfield, J. J. (1982). "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the National Academy of Sciences of the United States of America* 79.8, pp. 2554–2558. ISSN: 00278424. DOI: 10.1073/pnas.79.8.2554.

Huber, Jakob and Heiner Stuckenschmidt (2020). "Daily retail demand forecasting using machine learning with emphasis on calendric special days." In: *International Journal of Forecasting*. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2020.02.005.

Hyndman, Rob J and George Athanasopoulos (2018). *Forecasting: Principles and Practice*. URL: https://otexts.com/fpp2/ (visited on 10/23/2020).

Hyndman, Rob J and Yeasmin Khandakar (2008). "Automatic time series forecasting: the forecast package for R." In: *Journal of Statistical Software* 26.3, pp. 1–22. URL: https://www.jstatsoft.org/article/view/v027i03.

Hyndman, Rob J. and Anne B. Koehler (2006). "Another look at measures of forecast accuracy." In: *International Journal of Forecasting* 22.4, pp. 679–688. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2006.03.001. URL: http://www.sciencedirect.com/science/article/pii/S0169207006000239.

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree." In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., pp. 3149–3157. ISBN: 9781510860964.

Khalifa, Amine Ben and Hichem Frigui (2016). "Multiple Instance Fuzzy Inference Neural Networks." In: *CoRR* abs/1610.04973. arXiv: 1610.04973. URL: http://arxiv.org/abs/1610.04973.

Khan, Irfan Ahmad, Adnan Akber, and Yinliang Xu (May 2019). "Sliding Window Regression based Short-Term Load Forecasting of a Multi-Area Power System." In: *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE, pp. 1–5. ISBN: 978-1-7281-0319-8. DOI: 10.1109/CCECE.2019.8861915. URL: https://ieeexplore.ieee.org/document/8861915/.

Kim, Sungil and Heeyoung Kim (2016). "A new metric of absolute percentage error for intermittent demand forecasts." In: *International Journal of Forecasting* 32.3, pp. 669–679. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2015.12.003. URL: http://dx.doi.org/10.1016/j.ijforecast.2015.12.003.

Kiranyaz, S., T. Ince, and M. Gabbouj (2016). "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks." In: *IEEE Transactions on Biomedical Engineering* 63.3, pp. 664–675. DOI: 10.1109/TBME.2015.2468589.

Kiranyaz, S., T. Ince, R. Hamila, and M. Gabbouj (2015). "Convolutional Neural Networks for patient-specific ECG classification." In: *2015 37th Annual International*

Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 2608–2611. DOI: 10.1109/EMBC.2015.7318926.

Kiranyaz, Serkan, Onur Avci, Osama Abdeljaber, Turker Ince, Moncef Gabbouj, and Daniel J. Inman (2019). "1D convolutional neural networks and applications - A survey." In: *arXiv*, pp. 1–20.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2017). "ImageNet Classification with Deep Convolutional Neural Networks." In: *Commun. ACM* 60.6, pp. 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.
URL: https://doi.org/10.1145/3065386.

Lawrence, Michael, Paul Goodwin, Marcus O'Connor, and Dilek Önkal (2006). "Judgmental forecasting: A review of progress over the last 25 years." In: *International Journal of Forecasting* 22.3, pp. 493–518. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2006.03.007.

LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner (1998). "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11, pp. 2278–2323. ISSN: 00189219. DOI: 10.1109/5.726791.

Li, Ling, Sida Dai, and Zhiwei Cao (2019). "Deep long short-term memory (LSTM) network with sliding-window approach in urban thermal analysis." In: *2019 IEEE/CIC International Conference on Communications Workshops in China, ICCC Workshops 2019*, pp. 222–227. DOI: 10.1109/ICCChinaW.2019.8849965.

Lim, Bryan and Stefan Zohren (Apr. 2021). "Time-series forecasting with deep learning: A survey." In: 379.2194. ISSN: 1364503X. DOI: 10.1098/rsta.2020.0209. arXiv: 2004.13408.
URL: http://arxiv.org/abs/2004.13408.

Lim, Bryan, Stefan Zohren, and Stephen Roberts (2020). "Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction." In: *Proceedings of the International Joint Conference on Neural Networks*. DOI: 10.1109/IJCNN48605.2020.9206906. arXiv: 1901.08096.

MacKinnon, James (2010). "Critical Values for Cointegration Tests." In: *Working Paper* 1227. DOI: 10.22004/ag.econ.273723.

Makridakis, Spyros (1988). *Metaforecasting: Ways of Improving Forecasting Accuracy and Usefulness*. Tech. rep., p. 467.

Makridakis, Spyros and Michele Hibon (2000). "The M3-Competition: results, conclusions and implications." In: *International Journal of Forecasting* 16, pp. 451–476. URL: www.elsevier.com/locate/ijforecast.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2018a). "Statistical and Machine Learning forecasting methods: Concerns and ways forward." In: *PLoS ONE* 13.3. ISSN: 19326203. DOI: 10.1371/journal.pone.0194889.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2018b). "The M4 Competition: Results, findings, conclusion and way forward." In: *International Journal of Forecasting* 34.4, pp. 802–808. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2018.06.001.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2019). "The M4 Competition: 100,000 time series and 61 forecasting methods." In: *International Journal of Forecasting* 36.1, pp. 54–74. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2019.04.014.
URL: https://doi.org/10.1016/j.ijforecast.2019.04.014.

Makridakis, Spyros, Evangelos Spiliotis, and Vassilios Assimakopoulos (2020). "The M5 Accuracy Competition: Results, Findings and Conclusions." In: October, pp. 1–44. URL: https://www.researchgate.net/publication/344487258.

Masters, Timothy (1993). "10 - Designing Feedforward Network Architectures." In: *Practical Neural Network Recipies in C++*. Ed. by Timothy Masters. San Francisco (CA): Morgan Kaufmann, pp. 173–185. ISBN: 978-0-08-051433-8. DOI: https://doi.org/10.1016/B978-0-08-051433-8.50015-X.
URL: https://www.sciencedirect.com/science/article/pii/B978008051433850015X.

McCulloch, Warren S. and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity." In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133. ISSN: 00074985. DOI: 10.1007/BF02478259.

Minsky, M. and S. Papert (1969). "Perceptrons - an introduction to computational geometry." In: MIT Press.

Mosconi, Francesco, Ari Lerner, and Nate Murray (2019). *Zero to Deep Learning with Keras and Tensorflow*. Ed. by Nate Murray and Ari Lerner. 1st Edition. Fullstack.io.

Ng, Serena and Pierre Perron (1995). "Unit root tests in ARMA models with data-dependent methods for the selection of the truncation lag." In: *Journal of the American Statistical Association* 90.429, pp. 268–281. ISSN: 1537274X. DOI: 10.1080/01621459.1995.10476510.

Nielsen, M.A. (2015). *Neural Networks and Deep Learning*. Determination Press. URL: http://neuralnetworksanddeeplearning.com.

Oliver, F. R. and D. Gujarati (1993). "Essentials of Econometrics." In: *Journal of the Royal Statistical Society. Series A (Statistics in Society)* 156.2. ISSN: 09641998. DOI: 10.2307/2982744.

Oord, Aaron van den, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu (2016). "WaveNet: A Generative Model for Raw Audio." In: pp. 1–15. arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

Palit, Ajoy K. and Dobrivoje Popovic (2005). *Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications. Advances in Industrial Control*. Secaucus: Springer-Verlag New York. ISBN: 978-1-84628-184-6. DOI: 10.1007/1-84628-184-9.

Paoli, Christophe, Cyril Voyant, Marc Muselli, and Marie Laure Nivet (2010). "Forecasting of preprocessed daily solar radiation time series using neural networks." In: *Solar Energy* 84.12, pp. 2146–2160. ISSN: 0038092X. DOI: 10.1016/j.solener.2010.08.011. URL: http://dx.doi.org/10.1016/j.solener.2010.08.011.

Papadopoulos, Krist (2018). "SeriesNet: A Dilated Causal Convolutional Neural Network for Forecasting." In: April, pp. 1–22. URL: https://github.com/kristpapadopoulos/seriesnet/blob/master/seriesnet-Krist-Papadopoulos-v1.pdf.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). "On the difficulty of training recurrent neural networks." In: *30th International Conference on Machine*

Learning, ICML 2013. eprint: 1211.5063.

Petropoulos, Fotios (July 2015). *Forecasting Support Systems: ways forward*. Tech. rep. Cardiff, UK: Cardiff Business School.

Petropoulos, Fotios, Nikolaos Kourentzes, Konstantinos Nikolopoulos, and Enno Siemsen (May 2018). "Judgmental selection of forecasting models." In: *Journal of Operations Management* 60, pp. 34–46. ISSN: 02726963. DOI: 10.1016/j.jom.2018.05.005.

Qin, Zhuwei, Fuxun Yu, Chenchen Liu, and Xiang Chen (2018). *How convolutional neural networks see the world - A survey of convolutional neural network visualization methods*. DOI: 10.3934/mfc.2018008. eprint: 1804.11191.

Ranzato, Marc'Aurelio, Christopher Poultney, Sumit Chopra, and Yann LeCun (2006). "Efficient Learning of Sparse Representations with an Energy-Based Model." In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, pp. 1137–1144.

Rao, Guozheng, Weihang Huang, Zhiyong Feng, and Qiong Cong (Sept. 2018). "LSTM with sentence representations for document-level sentiment classification." In: *Neurocomputing* 308, pp. 49–57. ISSN: 09252312. DOI: 10.1016/j.neucom.2018.04.045.
URL: https://doi.org/10.1016/j.neucom.2018.04.045%20https://linkinghub.elsevier.com/retrieve/pii/S092523121830479X.

Rosenblatt, F. (Jan. 1957). *The perceptron - A perceiving and recognizing automaton*. Tech. rep. 85-460-1. Ithaca, New York: Cornell Aeronautical Laboratory.

Rowe, Gene and George Wright (1999). "The Delphi technique as a forecasting tool: Issues and analysis." In: *International Journal of Forecasting* 15.4, pp. 353–375. ISSN: 01692070. DOI: 10.1016/S0169-2070(99)00018-7.
URL: www.elsevier.com/locate/ijforecast.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms." In: pp. 1–14. arXiv: 1609.04747.
URL: http://arxiv.org/abs/1609.04747.

Rumelhart, D. E., P. Smolensky, J. L. McClelland, and G. E. Hinton (1986). "Schemata and Sequential Thought Processes in PDP Models." In: *Parallel Distributed*

Processing: Explorations in the Microstructure, Vol. 2: Psychological and Biological Models. Cambridge, MA, USA: MIT Press, pp. 7–57. ISBN: 0262631105.

Salinas, David, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski (2020). "DeepAR: Probabilistic forecasting with autoregressive recurrent networks." In: *International Journal of Forecasting* 36.3, pp. 1181–1191. ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2019.07.001.
URL: https://www.sciencedirect.com/science/article/pii/S0169207019301888.

Sanders, Nada R. and Larry P. Ritzman (2001). "Judgmental Adjustment of Statistical Forecasts." In: pp. 405–416. DOI: 10.1007/978-0-306-47630-3_18.

Schuster, Mike and Kuldip K. Paliwal (1997). "Bidirectional recurrent neural networks." In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681. ISSN: 1053587X. DOI: 10.1109/78.650093.

Sezer, Omer Berat, Mehmet Ugur Gudelek, and Ahmet Murat Ozbayoglu (May 2020). "Financial time series forecasting with deep learning: A systematic literature review: 2005–2019." In: *Applied Soft Computing Journal* 90. ISSN: 15684946. DOI: 10.1016/j.asoc.2020.106181.

Siami-Namini, Sima, Neda Tavakoli, and Akbar Siami Namin (Dec. 2019). "The Performance of LSTM and BiLSTM in Forecasting Time Series." In: *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 3285–3292. ISBN: 978-1-7281-0858-2. DOI: 10.1109/BigData47090.2019.9005997.
URL: https://ieeexplore.ieee.org/document/9005997/.

Siegelmann, Hava T. (1995). "Computation beyond the turing limit." In: *Science* 268.5210, pp. 545–548. ISSN: 00368075. DOI: 10.1126/science.268.5210.545.

Sivic and Zisserman (2003). "Video Google: a text retrieval approach to object matching in videos." In: *Proceedings Ninth IEEE International Conference on Computer Vision*, 1470–1477 vol.2. DOI: 10.1109/ICCV.2003.1238663.

Smith, Taylor G. *et al.* (2017). *pmdarima: ARIMA estimators for Python*. [Online; accessed 2020-11-29].
URL: http://www.alkaline-ml.com/pmdarima.

Smyl, Slawek (Jan. 2020). "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting." In: *International Journal of Forecasting* 36.1, pp. 75–85. ISSN: 01692070. DOI: 10.1016/j.ijforecast.2019.03.017.

Springenberg, Jost Tobias, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller (2015). "Striving for simplicity: The all convolutional net." In: *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pp. 1–14. arXiv: 1412.6806.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (Sept. 2014). "Sequence to sequence learning with neural networks." In: *Advances in Neural Information Processing Systems*. Vol. 4. January, pp. 3104–3112. arXiv: 1409.3215.
URL: http://arxiv.org/abs/1409.3215.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich (2015). "Going deeper with convolutions." In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June-2015, pp. 1–9. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

Taylor, Sean J and Benjamin Letham (2018). "Forecasting at Scale." In: *American Statistician* 72.1, pp. 37–45. ISSN: 15372731. DOI: 10.1080/00031305.2017.1380080.
URL: https://doi.org/10.7287/peerj.preprints.3190v2.

Vafaeipour, Majid, Omid Rahbari, Marc A. Rosen, Farivar Fazelpour, and Pooyandeh Ansarirad (2014). "Application of sliding window technique for prediction of wind velocity time series." In: *International Journal of Energy and Environmental Engineering* 5.2-3, pp. 1–7. ISSN: 22516832. DOI: 10.1007/s40095-014-0105-5.

Van Den Oord, Aäron, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu (2016). "Conditional image generation with PixelCNN decoders." In: *Advances in Neural Information Processing Systems*, pp. 4797–4805. ISSN: 10495258. arXiv: 1606.05328.

Vandeput, Nicolas (Nov. 2018). *Data Science for Supply Chain Forecast*, pp. 15–26. ISBN: 978-1730969430.

Varma, Sudhir and Richard Simon (Feb. 2006). "Bias in error estimation when using cross-validation for model selection." In: *BMC Bioinformatics* 7, p. 91. ISSN: 14712105. DOI: 10.1186/1471-2105-7-91.
URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1397873/.

Varsamopoulos, Savvas, Koen Bertels, and Carmen Garcia Almudever (Nov. 2019). "Comparing Neural Network Based Decoders for the Surface Code." In: *IEEE Transactions on Computers* 69.2, pp. 300–311. ISSN: 2326-3814. DOI: 10.1109/tc.2019.2948612.
URL: http://dx.doi.org/10.1109/TC.2019.2948612.

Wang, W. and J. Gang (2018). "Application of Convolutional Neural Network in Natural Language Processing." In: *2018 International Conference on Information Systems and Computer Aided Education (ICISCAE)*, pp. 64–70. DOI: 10.1109/ICISCAE.2018.8666928.

Webby, Richard and Marcus O'Connor (1996). "Judgemental and statistical time series forecasting: A review of the literature." In: *International Journal of Forecasting* 12.1, pp. 91–118. ISSN: 01692070. DOI: 10.1016/0169-2070(95)00644-3.

Weber, Mary Margaret and S. Prasad Kantamneni (2002). "POS and EDI in retailing: An examination of underlying benefits and barriers." In: *Supply Chain Management* 7.5, pp. 311–317. DOI: 10.1108/13598540210447755.

Wei, Donglai, Bolei Zhou, Antonio Torrabla, and William Freeman (2015). "Understanding Intra-Class Knowledge Inside CNN." In: 6.2, pp. 6–12. arXiv: 1507.02379.
URL: http://arxiv.org/abs/1507.02379.

Werbos, Paul J (1990). "Backpropagation Through Time: What It Does and How to Do It." In: *Proceedings of the IEEE* 78.10, pp. 1550–1560. ISSN: 15582256. DOI: 10.1109/5.58337.

Werbos, Paul J. (1988). "Generalization of backpropagation with application to a recurrent gas market model." In: *Neural Networks* 1.4, pp. 339–356. ISSN: 08936080. DOI: 10.1016/0893-6080(88)90007-X.

Willemain, Thomas R (1991). "The effect of graphical adjustment on forecast accuracy." In: *International Journal of Forecasting* 7.2, pp. 151–154. ISSN: 01692070. DOI:

10.1016/0169-2070(91)90049-2.

Winters, Peter R. (Apr. 1960). "Forecasting Sales by Exponentially Weighted Moving Averages." In: *Management Science* 6.3, pp. 324–342. DOI: 10.1287/mnsc.6.3.324. URL: https://doi.org/10.1287/mnsc.6.3.324.

Wöllmer, Martin, Florian Eyben, Alex Graves, Björn Schuller, and Gerhard Rigoll (2010). "Bidirectional LSTM Networks for Context-Sensitive Keyword Detection in a Cognitive Virtual Agent Framework." In: *Cognitive Computation* 2.3, pp. 180–190. ISSN: 18669956. DOI: 10.1007/s12559-010-9041-8.

Xu, Wenquan, Hui Hu, and Wei Yang (2019). "Energy Time Series Forecasting Based on Empirical Mode Decomposition and FRBF-AR Model." In: *IEEE Access* 7, pp. 36540–36548. DOI: 10.1109/ACCESS.2019.2902510.

Yeo, I. N.Kwon and Richard A. Johnson (2000). "A new family of power transformations to improve normality or symmetry." In: *Biometrika* 87.4. ISSN: 00063444. DOI: 10.1093/biomet/87.4.954.

Zeiler, M. D., D. Krishnan, G. W. Taylor, and R. Fergus (2010). "Deconvolutional networks." In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2528–2535. DOI: 10.1109/CVPR.2010.5539957.

Zeiler, Matthew D. and Rob Fergus (2014). "Visualizing and understanding convolutional networks." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS.PART 1, pp. 818–833. ISSN: 16113349. DOI: 10.1007/978-3-319-10590-1_53. arXiv: 1311.2901.

Zhang, G. Peter and Min Qi (2005). "Neural network forecasting for seasonal and trend time series." In: *European Journal of Operational Research* 160.2, pp. 501–514. ISSN: 03772217. DOI: 10.1016/j.ejor.2003.08.037.

Appendices

Appendix A

Evaluation and Parameterization

Table A.1: Number of forecasts that are valid for evaluation.

	Iteration 1	Iteration 2	Iteration 3
Store A	679	693	679
Store B	748	746	743
Store C	679	679	679

Table A.2: Number of learnable parameters of the LSTM-1 on the unstandardized setting ($p = 25$).

$p = 25$	Iteration 1	Iteration 2	Iteration 3
Store A	**94136** ($units = 89$)	**96729** ($units = 90$)	**96092** ($units = 90$)
Store B	**107346** ($units = 94$)	**107061** ($units = 94$)	**106966** ($units = 94$)
Store C	**96183** ($units = 90$)	**96547** ($units = 90$)	**96365** ($units = 90$)

Table A.3: Number of learnable parameters of the BiLSTM-1 (unstandardized) ($p = 25$).

$p = 25$	Iteration 1	Iteration 2	Iteration 3
Store A	**94494** ($units = 52$)	**95859** ($units = 52$)	**95124** ($units = 52$)
Store B	**105942** ($units = 54$)	**105615** ($units = 54$)	**105506** ($units = 54$)
Store C	**95229** ($units = 52$)	**95649** ($units = 52$)	**95439** ($units = 52$)

Table A.4: Number of learnable parameters of the MLP with $p = H$ if it had one output unit (unstandardized).

$p = H$	Iteration 1	Iteration 2	Iteration 3
Store A	**17889** ($units = 26$)	**18227** ($units = 26$)	**18045** ($units = 26$)
Store B	**20305** ($units = 27$)	**20224** ($units = 27$)	**20197** ($units = 27$)
Store C	**18071** ($units = 26$)	**18175** ($units = 26$)	**18123** ($units = 26$)

Table A.5: Number of learnable parameters of the LSTM-2 (unstandardized).

$p = H$	Iteration 1	Iteration 2	Iteration 3
Store A	**18019** ($units = 66$)	**18019** ($units = 66$)	**18019** ($units = 66$)
Store B	**20305** ($units = 70$)	**20305** ($units = 70$)	**20305** ($units = 70$)
Store C	**18019** ($units = 66$)	**18019** ($units = 66$)	**18019** ($units = 66$)

Table A.6: Number of learnable parameters of the BiLSTM-2 (unstandardized).

$p = H$	Iteration 1	Iteration 2	Iteration 3
Store A	**17757** ($units = 46$)	**18519** ($units = 47$)	**17757** ($units = 46$)
Store B	**20091** ($units = 49$)	**20091** ($units = 49$)	**20091** ($units = 49$)
Store C	**17757** ($units = 46$)	**18519** ($units = 47$)	**17757** ($units = 46$)

Table A.7: Configuration and number of learnable parameters of the SeriesNet.

	L	K	Parameters
$r = 8$	3	2	**353**
$r = 32$	5	2	**609**
$r = 128$ (original)	7	2	**865**
$r = 256$	8	2	**993**
$r = 768$	9	3	**1409**