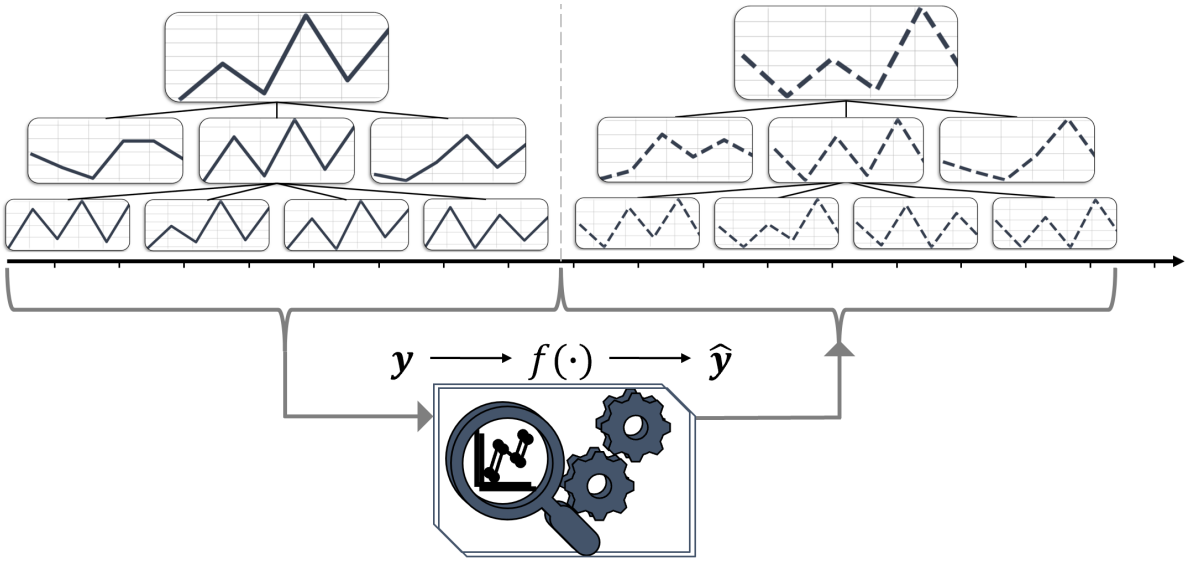


Small-Scale Demand Forecasting:

Exploring the Potential of Machine Learning and Hierarchical Reconciliation

Master's thesis in Complex Adaptive Systems

Viking Zandhoff Westerlund

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Small-Scale Demand Forecasting: Exploring the Potential of Machine Learning and Hierarchical Reconciliation

VIKING ZANDHOFF WESTERLUND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Small-Scale Demand Forecasting:
Exploring the Potential of Machine Learning and Hierarchical Reconciliation

VIKING ZANDHOFF WESTERLUND

Examiner: Mats Granath, Department of Physics

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Illustration of hierarchical time series forecasting. Designed with Python and PowerPoint.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Small-Scale Demand Forecasting: Exploring the Potential of Machine Learning and Hierarchical Reconciliation

VIKING ZANDHOFF WESTERLUND
Department of Physics
Chalmers University of Technology

Abstract

Demand forecasting plays an important role in facilitating data-driven decision-making for businesses, particularly in domains such as inventory planning and resource allocation. While traditional forecasting models such as exponential smoothing and autoregressive models have long been prevalent in the time series forecasting domain, recent research has been increasingly focused on more complex machine learning-based models. These complex models offer great potential and flexibility, but they require large amounts of data to achieve optimal performance. In this thesis, I explored viable approaches for constructing accurate forecasting models for a young company in the industrial production industry who wants to predict their future demand, while facing the challenge of limited data availability. The analysis in this thesis involved comparing the predictive performance of state-of-the-art machine learning models, such as the Temporal Fusion Transformer (TFT) and the LightGBM to an exponential smoothing state-space model. Furthermore, I investigated whether the hierarchical structure of the time series data could be exploited through forecast reconciliation to further increase forecasting accuracy. My findings indicate that both the TFT and LightGBM demonstrate superior forecasting accuracy, improving the average forecast accuracy with 43.1 % and 33.2 % respectively, compared to the exponential smoothing model. However, the TFT displayed inconsistent performance results, suggesting its unreliability. Moreover, the results show that while hierarchical forecast reconciliation does not enhance forecast accuracy, it corrects incoherency between forecasts without compromising accuracy, which is of great value.

Keywords: Time series forecasting, Demand forecasting, Hierarchical time series, Temporal Fusion Transformer, LightGBM, Minimum trace reconciliation

Acknowledgements

I would like to extend my sincere gratitude to the examiner of this thesis, Mats Granath, for his valuable feedback. I would also like to thank Adam Söderholm, the opponent of this thesis, for challenging me to improve my work by providing insightful thoughts and comments.

Viking Zandhoff Westerlund, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BU	Bottom-Up Reconciliation
ES	Exponential Smoothing
GBM	Gradient Boosting Machine
MinT	Minimum Trace
OLS	Ordinary-Least-Squares
PSEL	Percentage Sum of Errors per Level
TFT	Temporal Fusion Transformer
WLS_s	Weighted-Least-Squares applying structural scaling
WMASE	Weighted Mean Absolute Scaled Error

Contents

List of Figures

List of Tables

1

Introduction

The ability to foresee the future would equate to an unparalleled competitive advantage for any business in any industry. Strategic decisions would always be well-informed, resources could be allocated efficiently, and products and services could proactively be adapted to shifting customer demand. Consequently, predicting the future, also known as forecasting, has always been of central importance for businesses [1].

Some events are possible to anticipate quite precisely, such as the time of the sunrise tomorrow. However, in most business applications the future remains uncertain, as the past never really repeats itself [1]. Businesses operate within a complex landscape that is influenced by external factors such as competition, technological advancement, and the state of the global economy, making it virtually impossible to precisely predict the future. The goal is then to build a forecasting model that is as accurate as possible.

"*All models are wrong, some are useful*" - George E.P Box.

The availability of historical data is vital for any forecasting task. With the ever-increasing amount of data that is being collected by businesses and organizations, developing accurate time series forecasting models to support decision-making is becoming crucial for companies in order to stay competitive [2].

1.1 Background

A time series is a sequence of observations of some quantity over a period of time. A time series forecasting model is a mathematical model designed to identify patterns in a time series and extrapolate those patterns to make predictions about the future. Time series forecasting is of central importance in many domains, one prominent example – which will also be the focus of this thesis – is demand forecasting [3], [4].

Two of the most common time series forecasting models are exponential smoothing (ES) models and ARIMA models [5, ch. 9]. However, these traditional models have been shown to be inferior to more advanced machine learning-based models in terms of forecasting accuracy [6], [7]. In the recent M5 accuracy competition [4], all of the top 50 forecasting models were based purely on machine learning. Furthermore,

global tech giants such as Google [7] and Amazon WS [8] are replacing traditional models with more advanced machine learning models for many practical forecasting problems, and they claim to have seen significant improvements in forecast accuracy.

Models such as ETS and ARIMA are *model-driven*, meaning that they make assumptions about the characteristics of the underlying data generating process of a time series, such as trend and seasonality [9]. This makes model-driven forecasting methods more limited, but they can be very efficient on small data sets. Conversely, most (but not all) machine learning models are *data-driven* [9]. Data-driven models make no assumptions regarding the data generating process, instead they learn patterns directly from the historical data. Thus, data-driven methods are more flexible than model-driven methods, but they require a large amount of data in order to learn meaningful patterns and produce accurate forecasts without overfitting.

Consequently, large data sets are essential for companies and organizations that want to leverage the advantages of machine learning-based forecasting models. However, newly launched start-up companies or companies with recently introduced products may not have the several years of time series data required to train data-driven machine learning models effectively. To overcome this challenge, one approach is to train the models across multiple time series simultaneously, a method referred to as *cross-learning* [10]. With cross-learning, advanced machine learning models such as neural networks can be successfully implemented without having very lengthy time series data. Additionally, training models across multiple time series enables the models to capture complex patterns and relationships between correlated time series, which is not possible in series-to-series methods.

Although using cross-learning can facilitate the use of ML-based forecasting algorithms on shorter time series data, it can be challenging to reap the potential benefits if the time series are uncorrelated [4]. Fortunately, businesses in the industrial production and manufacturing industry often have multiple correlated time series, such as historical sales data for different products.

Time series of sales can often be structured by country, city, store and product category. Time series that adhere to such a structure are referred to as *hierarchical times series* [11]. Hierarchical time series forecasting is a special case of forecasting, where the forecasts of the time series are required to be *coherent*. The coherency constraint means that the sum of the forecasts of the disaggregated time series must be equal to the forecast of the most aggregated time series. To ensure coherence, *reconciliation* methods can be applied as a post-processing step to reconcile the base forecasts. A thorough description of hierarchical time series and reconciliation methods is provided in Section 3.

1.2 Problem Statement

The private data set that I will be using in this thesis contains historical demand data for a relatively young company in the industrial production industry, that can

be structured as hierarchical time series. Of special interest is forecasting the total sales volume and the volume for each individual product. Due to confidentiality agreements, the company will remain nameless throughout this thesis, and be referred to simply as The Company. Since The Company is young, the data set is small in terms of number of observations. However, the company wants to leverage demand forecasting to streamline its supply chain planning and resource allocation. Thus, it is important to explore possible ways to build forecasting models that can generate accurate predictions, despite having a limited amount of data.

The potential advantage of leveraging cross-learning for this task is two-fold. Firstly, it enables the use of ML-based forecasting models on shorter time series by training across multiple time series. Secondly, because the machine learning models train on multiple time series simultaneously they can potentially capture correlations between the time series, an especially important challenge in the field of hierarchical forecasting [6].

At another end, reconciliation methods have proven to increase forecasting accuracy for hierarchical time series by incorporating the correlation between time series [11]. However, in most papers on reconciliation methods the base forecasts are produced with traditional forecasting models such as ES and ARIMA [3], [11]–[13], models that are not capable of utilizing cross-learning.

This raises the question; to what extent can ML-based forecasting models, trained across all time series simultaneously, and reconciliation methods be combined to maximize forecasting accuracy on a small data set?

1.3 Purpose

The purpose of this thesis is to explore viable methods for implementing time series forecasting models to predict future demand for a young company in the industrial production industry, facing the inherent challenge of limited data availability.

To investigate this, the predictive performance of the machine learning-based forecasting models Temporal Fusion Transformer and Light Gradient Boosting Machine will be compared to a traditional exponential smoothing model. Additionally, the effect of applying hierarchical reconciliation methods will be examined.

1.4 Delimitations

In this thesis, the forecast accuracy of the three forecasting models (1) exponential smoothing state-space model, (2) Light Gradient Boosting Machine (LightGBM), (3) Temporal Fusion Transformer (TFT) will be compared. The underlying algorithms of these models are different, and therefore the learning phase is different. The differences in the resulting forecast accuracy between the models will be addressed, however I will not perform any in-depth comparison of the models and their technical details.

The analysis conducted in this thesis is limited to the private data set provided by The Company. Since different data sets have different characteristics, results obtained and conclusions drawn in this thesis may not be generalizable. Furthermore, per request of The Company, the forecasts in this thesis will be at a fixed length of 4 steps with weekly resolution. The predictive performance of the models on different time scales might vary. For example, error accumulation in recursive models may be more prominent for longer horizons.

1.5 Ethical Considerations

Time series forecasting is used in a wide range of applications. Thus, this section will consider the societal, ecological and ethical aspects of forecasting as a practice, rather than the aspects pertaining specifically to the domain of demand forecasting.

The purpose of forecasting demand is to facilitate efficient allocation of resources and plan production output. If this is done accurately, it would mean that companies can reduce waste by not over-producing. This could be especially impactful in fast moving retail businesses such as clothing. As an example of what can happen if the demand is misjudged, the fashion giant H&M had $4.3 billion worth of unsold clothes [14] in 2018.

When using forecasting within organizations, issues of moral hazard may arise [15]. Such situations arise when the manager or decision-maker within the organization have personal incentives to skew or affect the results of a forecast to serve his or her purpose, rather than giving an unbiased prediction of the future. This is a special case of the well-known principal/agent problem.

In a blog post called " 'Blame the algorithm' is the new 'Don't blame me. I just work here' ", the author illuminates the problem of treating an algorithm as a possible scapegoat for organizational mistakes [16]. The author refers to a case where a woman's medical home-care was cut in half based on a decision made entirely by an algorithm. The responsible organization was successfully sued for having an "unconstitutional" algorithmic allocating system.

Whereas this instance does not pertain to forecasting, similar problems could arise if practitioners do not assume responsibility for the forecasts produced by an algorithm. Machine learning algorithms are not flawless, and unanticipated real-world events may arise that causes forecasts to be completely wrong. In such scenarios, the organization in question must take appropriate actions and/or be held responsible.

2

Theory

This chapter contains the theoretical background and notations that are relevant for becoming familiar with time series forecasting and understanding the forecasting models. The first section covers notation and terminology pertaining to time series forecasting. Sections 2.2 and 2.3 provide the background and research that led up to the TFT and LightGBM, respectively. The last two sections cover model selection and evaluation as well as some common accuracy metrics used for assessing the predictive performance of the models.

2.1 Notation in Time Series Forecasting

Time series forecasting entails predicting the future value of some target variable y, that is observable over time. The *forecast horizon* h determines how many time steps from the *forecast origin* the forecast should reach. The simplest case is where $h = 1$, which is called a *one-step ahead* forecast. Traditional forecasting models such as ETS and ARIMA utilize past observations of the target variable to predict future values. As such, the one-step ahead forecast with origin at time t is defined as

$$\hat{y}_{t+1} = f(y_1, y_2, ..., y_t), \tag{2.1}$$

where $\hat{y}_{t+1}$ is the predicted value of y at time $t + 1$, $f(\cdot)$ is the forecasting model, and $\{y_1, y_2, ..., y_t\}$ are past observations of y. In many practical applications it is more informative to produce multi-step ahead forecasts, such as predicting the sales for each week over the next month. Forecasting multiple time steps is called *multi-horizon* forecasting, indicating that forecast horizon h is greater than one step. Given a forecast horizon $h > 1$, the multi-horizon forecast is defined by

$$\hat{y}_{t+\tau} = f(y_1, y_2, ..., y_t), \tag{2.2}$$

for $\tau \in \{1, ..., h\}$, which produces the set $\{\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h}\}$ of predicted values.

2.1.1 Recursive vs. Direct Forecasting

There exists several methods for producing multi-horizon forecasts, two of which are the *recursive* method [17], and the *direct*[1], also called the *multi-output* method [18]. Recursive forecasting is done by iteratively creating one-step ahead forecasts

$$\begin{aligned} \hat{y}_{t+1} &= f(y_1, y_2, ..., y_t), \\ \hat{y}_{t+2} &= f(y_1, y_2, ..., y_t, \hat{y}_{t+1}) \\ &\vdots \\ \hat{y}_{t+h} &= f(y_1, y_2, ..., y_t, \hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h-1}), \end{aligned} \tag{2.3}$$

where each preceding forecast is iteratively fed back to the model in order to produce subsequent forecasts. Models used for recursive forecasting are therefore optimized to produce one-step ahead forecasts. Despite this, recursive forecasting has been successfully implemented in multi-horizon forecasting [17].

Conversely, the direct forecasting method consists of directly forecasting the entire horizon.

$$\hat{y}_{t+1:t+h} = f(y_1, y_2, ..., y_t), \tag{2.4}$$

where $\hat{y}_{t+1:t+h} = \{\hat{y}_{t+1}, \hat{y}_{t+2}, ..., \hat{y}_{t+h}\}$. Theoretically, direct forecasting is better at modeling temporal dependencies between forecasted values [17]. Furthermore, the direct forecasting method does not suffer from the accumulation of errors that may arise when using the recursive approach.

2.1.2 Incorporating Multiple Sources of Data

Several machine learning-based forecasting models can incorporate data from sources other than past observations of the target variable [18]. Furthermore, utilizing cross-learning facilitates training ML-based models across multiple time series.

To define this mathematically, let I be a collection of entities, such as products of a store. Each individual product i is associated with a time-varying scalar target $y_{i,t}$, static data $\boldsymbol{s}_i$, time-varying observable exogenous inputs $\boldsymbol{z}_{i,t}$, and time-varying inputs $\boldsymbol{u}_{i,t}$ that are known in advance. Let $\boldsymbol{x}_{i,t} = (\boldsymbol{z}_{i,t}, \boldsymbol{u}_{i,t}, \boldsymbol{s}_i)$ denote the collection of all predictor inputs for entity i at time t. Given these heterogeneous sources of data, the τ-step ahead forecast takes the form

$$\hat{y}_{i,t+\tau} = f(y_{i,1:t}, \boldsymbol{z}_{i,1:t}, \boldsymbol{u}_{i,1:t+\tau}, \boldsymbol{s}_i, \tau), \tag{2.5}$$

where $y_{i,1:t} = \{y_{i,1}, ..., y_{i,t}\}$ is the set past observations of the target variable of entity i, $\boldsymbol{z}_{i,1:t} = \{\boldsymbol{z}_{i,1}, ..., \boldsymbol{z}_{i,t}\}$ is the set of past observations of exogenous explanatory variables, $\boldsymbol{u}_{i,1:t+\tau} = \{\boldsymbol{u}_{i,1}, ..., \boldsymbol{u}_{i,t+\tau}\}$ are time-varying inputs that are known for

[1]The term *direct* forecasting has historically been used to describe the process of fitting h different models, one for each time step $\tau \in \{1, ..., h\}$. However, following the terminology in [18] I will consider direct forecasting as fitting a single model that produces forecasts for all time steps $\tau \in \{1, ..., h\}$.

the entire forecast horizon, and $\boldsymbol{s}_i$ are the static variables related to the target variable[2]. Note that in Equation 2.5, $\tau = 1$ equates to the one-step ahead forecast, and $\tau \in \{1, ..., h\}$ would comprise a multi-horizon forecast.

To provide a relevant example, assume that y is the total daily sales of the product SKU01, and the task is to forecast the total sales for the upcoming 5 days $\{y_{t+1}, ..., y_{t+5}\}$. Then the forecast horizon $h = 5$, $y_{1:t}$ are the historical total daily sales, $\boldsymbol{z}_{1:t}$ could be the target $y_{1:t}$ with some applied transformation, or it could be historical daily sales of a different product SKU02, or some exogenous variable such as the inflation. Furthermore, $\boldsymbol{u}_{1:t+h}$ could be the specific weekdays in the forecast horizon h which are known beforehand, and $\boldsymbol{s}$ could be the product category of SKU01.

2.2 Deep Learning for Modeling Time Series Data

This section will give a brief description of the evolution of sequential data modeling that laid the foundations for the Temporal Fusion Transformer (TFT) [19]. The TFT will be described in more detail in Section 4.2.

An artificial neural network (ANN) is a machine learning algorithm that is widely adopted for a range of different tasks such as classification, regression, natural language processing, as well as forecasting. The ANN algorithm is inspired by the human brain, building on computational units called *neurons* [20, ch. 1].

When dealing with sequential data such as text sentences or time series, the most prevalent ANN architectures are based on the idea of *recurrent neural networks* (RNN) [21]. Figure 2.1 shows a schematic view of an "unfolded" RNN with a single hidden neuron. The fundamental principle of recurrent neural networks is that there are recurrent connections in the network architecture. These connections allow outputs from previous states to be re-used, such that the output of the network at time t takes into account both the input $\boldsymbol{x}_t$, as well as the preceding *hidden state* $\boldsymbol{h}_{t-1}$ of the network[3]. Through this recurrent mechanism, the network retains information from past time steps, facilitating the learning of sequential dependencies. Hence, RNNs are able to capture temporal dependencies in time series data.

Although recurrent neural networks are suitable for modeling sequential data, they suffer from a drawback known as the vanishing/exploding gradient problem which makes RNNs infeasible for long sequences. To handle this problem, the Long Short-Term Memory (LSTM) architecture was developed [22].

The LSTM network is based on the same fundamental idea that the hidden state of the network should be propagated recurrently through time to memorize information from past time steps. The difference between an RNN and an LSTM network is that the computational unit is no longer a neuron, but an LSTM cell. Figure 2.2

[2]To simplify notation, the subscript i will be omitted throughout the report unless necessary.
[3]The first hidden state $\boldsymbol{h}_0$ must be initialized manually, e.g by zero or random initialization.

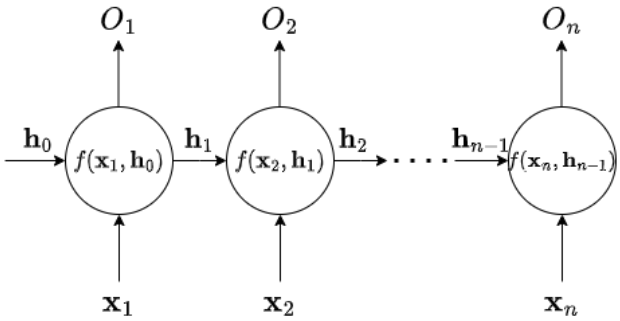


Figure 2.1: Schematic view of an unfolded recurrent neural network with one hidden neuron. Arrows indicate flow of information.

illustrates the high level structure of an LSTM cell, and how information flows through it.

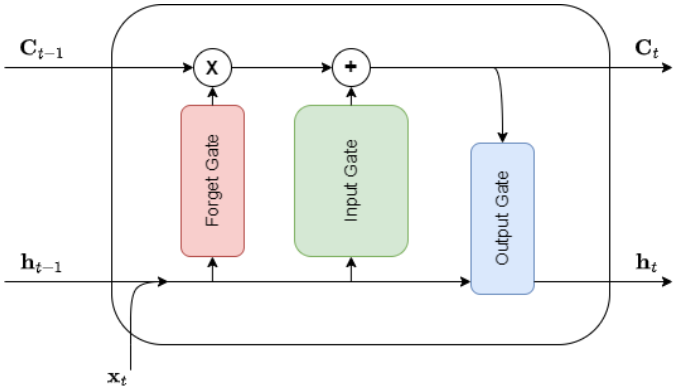


Figure 2.2: Schematic view of an LSTM cell. Arrows indicate flow of information.

The LSTM cell recurrently propagates two states to subsequent time steps, the hidden state $\boldsymbol{h}_t$ which is referred to as the short-term memory, and the *cell state* $\boldsymbol{C}_t$. The cause of the vanishing/exploding gradient problem in RNNs is that the gradients become exponentially dependent on the weights of the recurrent connections during backpropagation through time [22]. As depicted in Figure 2.2, the cell state $\boldsymbol{C}_t$ is not affected by any weights, effectively mitigating the vanishing/exploding gradient problem. Hence, the cell state serves as the long-term memory of the network.

Fundamentally, the previous hidden state $\boldsymbol{h}_{t-1}$ and the input $\boldsymbol{x}_t$ are fed to the "Forget Gate". The forget gate then computes what fraction of the previous cell state $\boldsymbol{C}_{t-1}$ that should be kept. The hidden state $\boldsymbol{h}_{t-1}$ and the input $\boldsymbol{x}_t$ are then fed to the "Input Gate" that computes a new contribution that is added to $\boldsymbol{C}_{t-1}$ to form $\boldsymbol{C}_t$. Finally, the new cell state $\boldsymbol{C}_t$, the previous hidden state $\boldsymbol{h}_{t-1}$ and the input $\boldsymbol{x}_t$ are fed to the "Output Gate" where the new hidden state $\boldsymbol{h}_t$ is computed. The new hidden state $\boldsymbol{h}_t$ is also the output value of the LSTM cell.

When handling sequential data, there are different forms of input/output mappings such as one-to-one and many-to-one. Another common mapping is many-to-many,

where a model takes a sequence of input values and produces a sequence of output values, also referred to as sequence-to-sequence problems [23]. Two prominent examples of sequence-to-sequence problems are natural language processing and time series forecasting.

Sequence-to-sequence problems can efficiently be solved with *Encoder-Decoder* models [23]. An encoder-decoder model consists of an encoder that encodes information from an input sequence into a fixed-size context vector v, and a decoder that uses the information in v to produce an output sequence. Encoder-decoder models are frequently based on LSTMs due to the effectiveness of LSTMs in handling sequential data. Although encoder-decoder models were originally developed for natural language processing, the use of LSTM-based encoder-decoder models has become widely adopted for time series forecasting [18], [19], [21], [24].

A conceptual image of an LSTM-based encoder-decoder model is shown in Figure 2.3. In a time series context, the encoder is given a sequence of historical observations of predictors $\{\boldsymbol{z}_{t-k}, ..., \boldsymbol{z}_t\}$, past known inputs $\{\boldsymbol{u}_{t-k}, ..., \boldsymbol{u}_t\}$, static features $\boldsymbol{s}$, and historical observations of the targets $\{y_{t-k}, ..., y_t\}$ for a pre-specified fixed *look-back window* of k time steps [18]. The encoder summarizes the input data in the context vector $\boldsymbol{v}_t$ that is passed to the decoder. The decoder combines the summarized information in $\boldsymbol{v}_t$ with the future known inputs $\{\boldsymbol{u}_{t+1}, ..., \boldsymbol{u}_{t+h}\}$ and produces a sequence of forecasted values $\{\hat{y}_{t+1}, ..., \hat{y}_{t+h}\}$ for a pre-specified forecast horizon h. The forecasts take the form

$$\hat{y}_{t+1:t+h} = f(y_{t-k:t}, \boldsymbol{z}_{t-k:t}, \boldsymbol{u}_{t-k:t+h}, \boldsymbol{s}_i), \tag{2.6}$$

where $\hat{y}_{t+1:t+h} = \{\hat{y}_{t+1}, ..., \hat{y}_{t+h}\}$ and $f(\cdot)$ is the encoder-decoder prediction model. Thus, such sequence-to-sequence models are *direct* forecasting models.

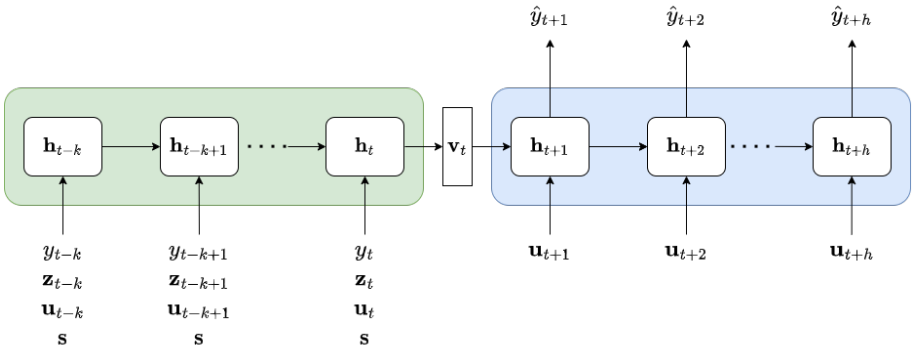


Figure 2.3: Schematic view of an encoder-decoder model.

Although the vanishing gradient problem is alleviated by using LSTM based architectures, it remains difficult to capture very long-term temporal dependencies in the data, since the cell state of the LSTM network is continuously updated with every iteration [21]. To enhance the ability of the models to capture long term dependencies, the encoder-decoder models can be augmented by incorporating attention mechanisms [21], [25].

2.3 Gradient Boosted Decision Trees

Gradient boosting is a powerful machine learning technique commonly used for classification and regression, originally introduced by Friedman [26]. The gradient boosting technique combines multiple weak learners using an ensemble method to create a strong model. The algorithm described in this section comprises the fundamental details of the LightGBM [27]. The LightGBM, which is an advanced version of a gradient boosted decision tree, will be described in Section 4.3.

Given a set of data points $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$ where $\boldsymbol{x}$ are the predictor variables and y is the target variable, and a differentiable loss function $\mathcal{L}$, the gradient boosting algorithm consists of finding an estimation of the function $F^* : \boldsymbol{x} \to y$ that minimizes $\mathcal{L}$, by iteratively adding so called *weak learners*, or *base learners* to the function estimate F. The aim of an individual weak learner is not to capture all of the dynamics in the data, but rather to capture some of the dynamics. The ensemble of weak learners is then used to make strong predictions. The function estimate at iteration m can be expressed on the form

$$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + f_m(\boldsymbol{x}), \tag{2.7}$$

where $F_{m-1}(\boldsymbol{x})$ is the function estimate from the previous iteration, and $f_m(\boldsymbol{x})$ is the function estimation increment, also called "boost". In gradient boosting, each boost $f_m(\boldsymbol{x})$ is fit to minimize the negative pseudo-residuals

$$\tilde{y}_{i,m} = -\frac{\partial \mathcal{L}(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\Big|_{F(\boldsymbol{x})=F_{m-1}(\boldsymbol{x})}. \tag{2.8}$$

The weak learners can have different functional forms, one common implementation is to use decision trees. When decision trees are used as weak learners, the ensemble model is referred to as *gradient boosted decision trees* (GBDT) [28].

In a GBDT, the weak learners are decision trees with a limited number L of terminal regions, also called leaves. The weak learner decision tree function is defined as $h(\boldsymbol{x}; \{R_{l,m}\}_1^L)$, where $R_{l,m}$ is the l-th terminal region of the decision tree at iteration m. The decision tree is fit to the negative pseudo-residuals $\tilde{y}_{i,m}$ from Equation 2.8 using a least squares criterion [28]. Once all pseudo-residuals have been partitioned into terminal regions, the optimal output value of the weak learner for each terminal region is computed by finding the value $\gamma_{l,m}$ that minimizes the loss in that region, where

$$\gamma_{l,m} = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{l,m}} \mathcal{L}(y_i, F_{m-1}(\boldsymbol{x}_i) + \gamma), \tag{2.9}$$

taking the previous predictions $F_{m-1}(\boldsymbol{x}_i)$ into account. Equation 2.9 clarifies the fundamental principle of gradient boosting, that the new weak learner is fit to minimize the errors of the current function estimate F_{m-1}. The function increment of iteration m is separately defined for each terminal region

$$f_m(\boldsymbol{x}) = \eta \cdot \gamma_{l,m} \mathbf{I}(\boldsymbol{x}_i \in R_{l,m}), \tag{2.10}$$

where η is the learning rate that controls the size of the contribution of each weak learner, and $\mathbf{I}(\cdot)$ is the indicator function. Finally, the function estimate $F_m(\boldsymbol{x})$ is updated as described in Equation 2.7.

The initial function estimate for each terminal region is computed by

$$F_0(\boldsymbol{x}) = \arg\min_{\gamma} \sum_{\boldsymbol{x}_i \in R_{l,0}} \mathcal{L}(y_i, \gamma). \tag{2.11}$$

The procedure of fitting a decision tree to the negative pseudo-residuals requires iterating over the entire data set to find the optimal split points, which is a computationally expensive process. To alleviate the computational cost of fitting the weak learners, Friedman modified his original boosting algorithm using "bagging". Bagging is an acronym for "bootstrap aggregation", a technique that was introduced by Breiman [29].

Bagging entails randomly subsampling the training data and fitting the weak learner on the subset, rather than the entire data set. In the GBDT algorithm the sampling is done without replacement. Bagging speeds up the training process, and according to Breiman [29], the randomness that is introduced in the subsampling procedure can improve the generalizability of the predictor model.

2.4 Model Selection and Evaluation

Two main concerns in any supervised learning task are to select the best model possible, and assess the generalization ability of that model [30]. The problem of selecting the best model is two-fold. The first dimension involves finding the best hyperparameters for a particular type of model, and the second dimension involves comparing different models and choosing the one best-suited for the specific task.

A straightforward approach to selecting and evaluating models is to split the available data into three distinct subsets called the training, validation and test set. The models are fit to the training data, using the validation error to find the optimal hyperparameters. Then, the predictive performance of the models are evaluated and compared on the unseen test data. This approach is referred to as the *three-way holdout method* [30].

The three-way holdout method has a major downside when working with small data sets [30]. Since the validation error used during hyperparameter tuning and training is computed for a specific split, the computed validation error is just a point estimate of the true generalization error of the model. In other words, it is possible that a selected model is optimized for on a validation fold that is not representative of the true data distribution. Thus, the predictive performance of a model may be sensitive to the particular data split. As a visualized example, in Figure 2.4, the data in the validation set is not representative of the entire distribution, and this split will benefit models that are prone to pessimistic forecasts.

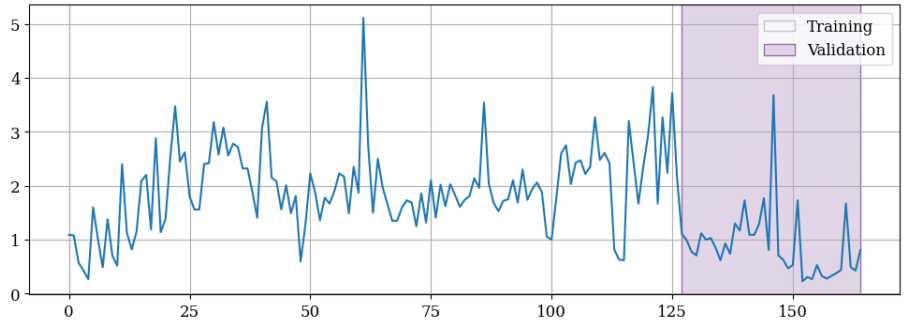


Figure 2.4: Time series data split using a training and validation set.

To reduce the effect of the randomness associated with the particular data split, in classification and regression tasks it is common to use a more robust method called k-fold cross-validation [30]. This method involves splitting the data into k disjoint folds, training the model on $k-1$ folds and using the last fold as a validation set. The validation set is then rotated, and the validation accuracy is summed across all folds. Figure 2.5 depicts a 5-fold cross validation split. Note that an additional test set is used for out-of-sample predictions.

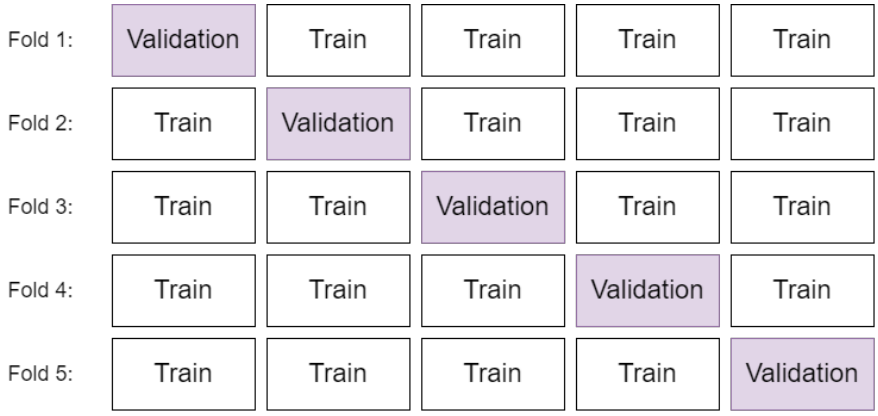


Figure 2.5: 5-fold cross-validation.

From a statistical perspective, this method computes the average of k point estimators of the generalization error, rather than relying on one point estimate.

The k-fold cross-validation method relies on the assumption that all data points are independent. However, in time series forecasting all data points have a temporal relationship. Thus, traditional k-fold cross validation is not a viable option for time series forecasting [31]. The simplest approach to tackle this problem is the *time series split* [31], depicted in Figure 2.6. Note that the fourth fold would correspond to the situation shown in Figure 2.4, where the amount of training data is maximized.

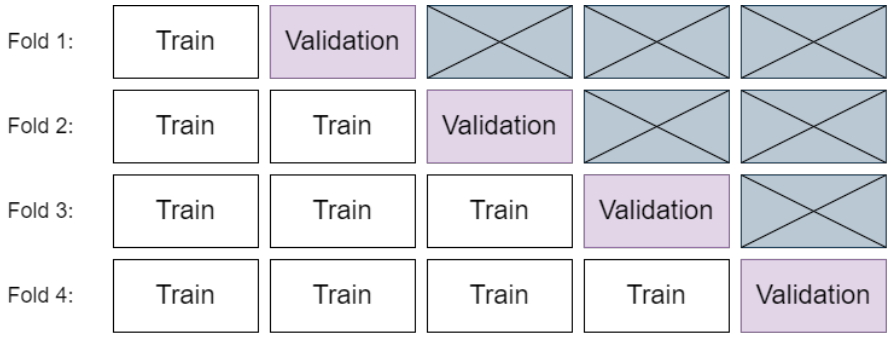


Figure 2.6: Time series split cross validation.

The time series split cross-validation method has its drawbacks. Since each fold contains different amounts of data, hyperparameters optimized for one fold may not be optimal for another [31]. Furthermore, the earliest data points, which may also be the least important, are used several times whereas the most recent data is only used once. However, it is a more robust cross-validation method than the three-way holdout method.

Once the hyperparameters of the model are chosen, the problem of comparing the generalization ability of different models still remains. Using a single hold-out test set would induce the data split randomness discussed for the validation set. However, if the objective is to establish a ranking order between different models, it suffices to evaluate their relative performance [30]. Thus, as long as the bias induced by the data split randomness affects all models similarly, the comparison is not affected.

Conversely, if the objective is to assess the generalization ability of any particular forecasting model, having a representative test set is of central importance.

2.5 Forecast Accuracy Metric

When evaluating the predictive performance of different forecasting models, there are several ways to measure accuracy. Some common metrics are the Mean Absolute Error (MAE), the Root Mean Squared Error (RMSE), the Mean Absolute Percentage ERROR (MAPE) [32]. However, no metric is perfect, and these metrics have undesired properties that make them inappropriate for evaluating forecasting accuracy. The MAE is defined by

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|, \tag{2.12}$$

where y_t is the true value and $\hat{y}_t$ is the predicted value at time t. The MAE, and the similarly defined RMSE are scale-dependent, which means that they cannot be used to compare forecast accuracy across time series of different scale. Continuing, the MAPE is defined by

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|100\frac{y_t - \hat{y}_t}{y_t}\right|, \tag{2.13}$$

which is closely related to the MAE, but it is scaled by the true value y_t. Thus, it alleviates the problem of scale dependency, but it becomes undefined if a single value y_t is zero. To avoid these unwanted properties, Hyndman and Koehler [33] propose a metric called the Mean Absolute Scaled Error (MASE) defined by

$$\text{MASE} = \frac{\frac{1}{h}\sum_{t=n+1}^{n+h}|y_t - \hat{y}_t|}{\frac{1}{n-1}\sum_{t=2}^{n}|y_t - y_{t-1}|}, \tag{2.14}$$

where the numerator is the MAE computed for each step in the forecast horizon h, and the denominator is the mean absolute lag-1 difference of the in-sample data.

The MASE is indifferent to the scale of the time series data, which means that results are comparable between time series of different magnitudes, and it can handle zero-values [33]. The M5 competition used a variety of the MASE called the Root Mean Squared Scaled Error (RMSSE), which is similarly defined [4].

While the MASE has the desired properties, its interpretation is not as straight forward as the MAE, RMSE or MAPE [32]. To best understand it, the denominator in Equation 2.14 can be interpreted as the mean absolute error (MAE) of the naïve one-step ahead in-sample forecast [33]. Thus, the MASE can be interpreted as the relative forecast accuracy compared to the average naïve one-step in-sample forecast error. If the MASE is smaller than 1, then the forecasts are better, otherwise they are worse. In simple terms, a lower MASE indicates a higher accuracy.

3

An Introduction to Hierarchical Time Series

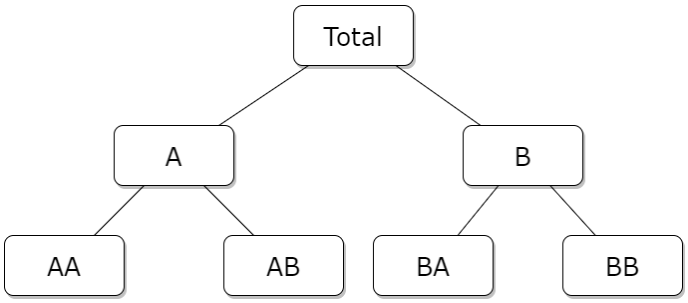
This chapter will serve as an introduction to hierarchical time series. The first section will provide an explanation of what hierarchical time series are, and give examples on where they might appear. The second section will give a review of reconciliation methods, what they are and why they are used. In Section 3.3, I define an accuracy metric that captures certain aspects of the forecasting performance that are relevant for the hierarchical forecasting in this thesis.

3.1 Hierarchical Time Series

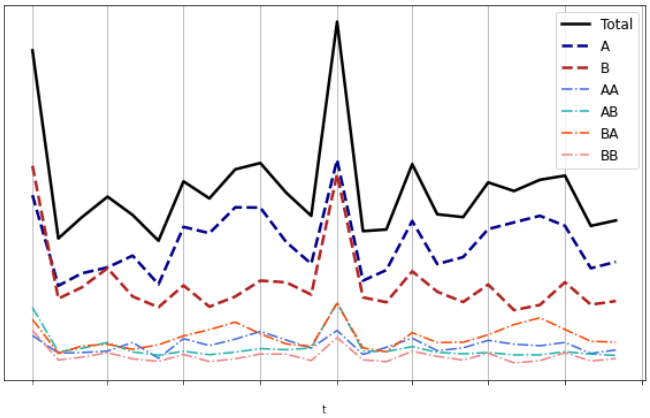
Hierarchical time series are a special type of multivariate time series where the individual time series are related in a hierarchical structure [11]. An example of such a structure is shown in Figure 3.1a. The top node represents some total quantity. The children of the top node represent the total quantity split by some parameter, which for example could be geographical location, product category or store-id. The children are further split in a similar way, until the bottom level time series are reached. The hierarchy in Figure 3.1a has two levels, but in general hierarchical time series can have any number of levels. All levels must adhere to the constraint that the sum of the children nodes must be equal to the parent node. That is, $A = AA + AB$, $B = BA + BB$ and $Total = A + B$. Figure 3.1b shows an example of a time series representation of this hierarchy.

Hierarchical time series appear in many different domains, such as tourism flow [12], electricity consumption [13], and sales in the retail industry [11]. As an example, in Figure 3.1 the top level time series, $Total$, could represent the total sales of a company, with intermediate level time series A and B being the total sales in country A and B respectively. Finally, the bottom level time series AA, AB, BA and BB could represent total sales from two different stores in each of the countries.

Specifically, following the notation in [11], consider a set of N time series of length T that adhere to a hierarchical structure. Let $\boldsymbol{y}_t \in \mathbb{R}^N$ be a vector containing the observation of each individual time series at time step t, and let $\boldsymbol{b}_t \in \mathbb{R}^M$ be a vector containing the observations of the M bottom-level time series at time t. The



(a) A two-level hierarchical time series represented as a tree diagram.



(b) Illustration of hierarchical time series. Image is for visual purposes and is not made to scale.

Figure 3.1: Tree-diagram and line graph visualizations of hierarchical time series.

hierarchical structure can then be represented in matrix form as

$$\boldsymbol{y}_t = \boldsymbol{S}\boldsymbol{b}_t, \tag{3.1}$$

where $\boldsymbol{S} \in \mathbb{R}^{N \times M}$ is the summing-matrix that aggregates the bottom-level time series to the upper levels. In the example presented in Figure 3.1 we have $\boldsymbol{y}_t = [y_{Total,t}, y_{A,t}, y_{B,t}, y_{AA,t}, y_{AB,t}, y_{BA,t}, y_{BB,t}]^T$, $\boldsymbol{b}_t = [y_{AA,t}, y_{AB,t}, y_{BA,t}, y_{BB,t}]^T$, and the summing-matrix

$$\boldsymbol{S} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ & & \boldsymbol{I}_4 & \end{bmatrix},$$

where $\boldsymbol{I}_4 \in \mathbb{R}^{4 \times 4}$ is the identity matrix.

Strictly defined, a collection of time series that are structured in a natural hierarchy and disaggregates in a unique manner, such as a geographical hierarchy, is referred to as *hierarchical* time series [11]. However, a collection of time series that can be aggregated by shared attributes, but does not disaggregate in a unique manner is

called *grouped* time series. An example of grouped time series is when total sales can be disaggregated by product category and geographical region. In this case there would be no difference in disaggregating the time series by region and then by product category, or vice versa. It is also possible to disaggregate total sales by category and region in parallel, in which case the aggregation structure is no longer hierarchical. However, the same principles apply. Throughout this paper I will use the term hierarchical time series to refer to both hierarchical and grouped time series, but highlight important differences when they arise.

3.2 Forecast Reconciliation Methods

As discussed in the previous section, hierarchical time series must adhere to the constraint that the sum of the children nodes is equal to the parent node. The same must be true for the forecasts of the time series. If the forecasts adhere to this constraint, the forecasts are *coherent* [11]. Generally, base forecasts of the time series will not be coherent, which means that some additional method is required to ensure coherence. Such methods are commonly referred to as reconciliation methods.

The most common reconciliation methods are linear operations [11]. To define a linear reconciliation, let $\hat{\boldsymbol{y}}_{t+h}$ be the h-step multi-horizon base forecasts of all time series in the hierarchy, and $\boldsymbol{S}$ be the summing-matrix. Every linear reconciliation can be written as

$$\tilde{\boldsymbol{y}}_{t+h} = \boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_{t+h}, \tag{3.2}$$

for an appropriate choice of the mapping matrix $\boldsymbol{P} \in \mathbb{R}^{M \times N}$, where $\tilde{\boldsymbol{y}}_{t+h}$ is the set of reconciled forecasts.

3.2.1 Top-Down and Bottom-Up Reconciliation

Two simple reconciliation methods are the Bottom-Up (BU) and Top-Down (TD) [11]. The BU approach is to create base forecasts for all time series in the bottom level, and then aggregate them to the higher levels. For example, The Company is interested in forecasting sales per product as well as total sales. Using the BU approach, one would produce base forecasts for all individual products and aggregate the predicted volume to produce a forecast for the total volume.

In the BU approach, the mapping matrix $\boldsymbol{P} = [\mathbf{0}_{M\times(N-M)} \mid \boldsymbol{I}_M]$ where $\mathbf{0}_{M\times(N-M)}$ is the $M \times (N-M)$ null-matrix and $\boldsymbol{I}_M$ is the $M \times M$ identity matrix. The downside of this approach is that time series at the lowest levels often have low signal-to-noise ratio which makes forecasting difficult.

The TD approach is to forecast the top level time series and disaggregate to the lower levels in the hierarchy. In this approach, the matrix $\boldsymbol{P} = [\boldsymbol{p} \mid \mathbf{0}_{M\times(N-1)}]$, where $\boldsymbol{p} = [p_1, p_2, ..., p_M]^T$ is the set of proportions used to disaggregate the top

level forecast. The downside of the TD approach is that it is difficult to estimate an accurate set of proportion factors.

3.2.2 Minimum Trace (MinT) Reconciliation

Both the BU and TD reconciliation methods suffer from information loss because the forecasts of the entire hierarchy are based only on forecasting the bottom level or top level. To exploit information from all levels in the hierarchy, Wickramasuriya *et al.* [11] suggested an approach where an optimal mapping matrix $\boldsymbol{P}$ is derived by minimizing the trace of the covariance matrix of the reconciled forecast errors.

Let

$$\hat{\boldsymbol{e}}_t(h) = \boldsymbol{y}_{t+h} - \hat{\boldsymbol{y}}_{t+h} \tag{3.3}$$

be the base forecast errors. Assuming that the base forecasts are unbiased, then $\mathbb{E}[\hat{\boldsymbol{e}}_t(h)] = 0$ which implies $\mathbb{E}[\boldsymbol{y}_{t+h}] = \mathbb{E}[\hat{\boldsymbol{y}}_{t+h}]$. Let $\hat{\boldsymbol{b}}_{t+h}$ be the bottom level base forecasts with $\mathbb{E}[\hat{\boldsymbol{b}}_{t+h}] = \boldsymbol{\beta}_{t+h}$. The relation in Equation 3.1 gives $\mathbb{E}[\hat{\boldsymbol{y}}_{t+h}] = \boldsymbol{S}\boldsymbol{\beta}_{t+h}$. Given the unbiasedness assumption, this implies that $\mathbb{E}[\boldsymbol{y}_{t+h}] = \boldsymbol{S}\boldsymbol{\beta}_{t+h}$. Thus, the set $\tilde{\boldsymbol{y}}_{t+h}$ of reconciled forecasts is unbiased if and only if

$$\begin{aligned}
\mathbb{E}[\tilde{\boldsymbol{y}}_{t+h}] &= \mathbb{E}[\boldsymbol{y}_{t+h}] \\
\mathbb{E}[\boldsymbol{S}\boldsymbol{P}\hat{\boldsymbol{y}}_{t+h}] &= \boldsymbol{S}\boldsymbol{\beta}_{t+h} \\
\boldsymbol{S}\boldsymbol{P}\boldsymbol{S}\boldsymbol{\beta}_{t+h} &= \boldsymbol{S}\boldsymbol{\beta}_{t+h} \\
\boldsymbol{S}\boldsymbol{P}\boldsymbol{S} &= \boldsymbol{S} \\
\boldsymbol{S}'\boldsymbol{S}\boldsymbol{P}\boldsymbol{S} &= \boldsymbol{S}'\boldsymbol{S} \\
\boldsymbol{P}\boldsymbol{S} &= \boldsymbol{I}_N,
\end{aligned} \tag{3.4}$$

where $\boldsymbol{I}_N \in \mathbb{R}^{N \times N}$ is the identity matrix. Furthermore, let the h-step reconciled forecast errors be defined as

$$\tilde{\boldsymbol{e}}_t(h) = \boldsymbol{y}_{t+h} - \tilde{\boldsymbol{y}}_{t+h}. \tag{3.5}$$

Wickramasuriya et al. [11] proved that for any $\boldsymbol{P}$ such that $\boldsymbol{P}\boldsymbol{S} = \boldsymbol{I}_N$, the covariance matrix of $\tilde{\boldsymbol{e}}_T(h)$ is given by

$$\mathrm{Var}[\tilde{\boldsymbol{e}}_t(h)] = \boldsymbol{S}\boldsymbol{P}\boldsymbol{W}_h\boldsymbol{P}'\boldsymbol{S}', \tag{3.6}$$

where $\boldsymbol{W}_h \in \mathbb{R}^{N \times N}$ is the covariance matrix of the h-step multi-horizon base forecast errors. The goal of the minimum trace reconciliation method is to find the mapping matrix $\boldsymbol{P}$ that satisfies $\boldsymbol{P}\boldsymbol{S} = \boldsymbol{I}_N$, that minimizes the trace of the matrix in Equation 3.6. Minimizing the trace of the covariance matrix is equivalent to minimizing the sum of the variances of the reconciled forecast errors. This problem has the closed form solution

$$\boldsymbol{P} = (\boldsymbol{S}'\boldsymbol{W}_h^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}'\boldsymbol{W}_h^{-1}. \tag{3.7}$$

Since the base forecast errors of the h-step multi-horizon base forecasts cannot be computed beforehand, the matrix $\boldsymbol{W}_h$ must be estimated. Some methods for estimating $\boldsymbol{W}_h$ suggested in [11] are listed below.

1. **OLS**: Set $\boldsymbol{W}_h = \boldsymbol{I}_N$. This method is equivalent to the ordinary-least-squares estimator suggested in [12]. Using this method, the mapping matrix becomes

$$\begin{aligned} \boldsymbol{P} &= (\boldsymbol{S}'\boldsymbol{I}_N^{-1}\boldsymbol{S})^{-1}\boldsymbol{S}'\boldsymbol{I}_N^{-1} \\ &= \boldsymbol{S}'\boldsymbol{S}^{-1}\boldsymbol{S}' \end{aligned} \tag{3.8}$$

   which means that the optimal mapping matrix is independent of the data.

2. **WLS$_s$**: Set $\boldsymbol{W}_h = \boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ is a diagonal matrix whose diagonal elements are given by $\boldsymbol{S}\mathbf{1}$, where $\boldsymbol{S}$ is the summing matrix and $\mathbf{1} \in \mathbb{R}^{M\times 1}$ is the unit vector. This method constructs the mapping matrix $\boldsymbol{P}$ in a way such that the weights of $\boldsymbol{P}$ correspond to how many bottom-level time series that are aggregated to obtain a given time series. For example, the weight of the top-level time series will be M and for all bottom-level time series the weights will be 1. Thus, it is referred to as WLS$_S$ (weighted-least-squares applying structural scaling).

3. **WLS$_V$**: Set $\boldsymbol{W_h} = \text{diag}(\hat{\boldsymbol{W}}_1)$, where

$$\hat{\boldsymbol{W}}_1 = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{e}_t\boldsymbol{e}_t' \tag{3.9}$$

   and $\boldsymbol{e}_t$ is a vector of the one-step ahead base forecast residuals for each time series at time t. Using this method, the mapping matrix P consists of weights based on the variance of the forecast residuals and is therefore referred to as WLS$_V$ (weighted-least-squares using variance scaling).

4. **Shrink**: Set $\boldsymbol{W_h} = \hat{\boldsymbol{W}}^*_{1,D}$, defined by

$$\hat{\boldsymbol{W}}^*_{1,D} = \lambda_D \text{diag}(\hat{\boldsymbol{W}}_1) + (1-\lambda_D)\hat{\boldsymbol{W}}_1, \tag{3.10}$$

   where λ_D is a shrinkage parameter. The idea of the MinT Shrink method is that off-diagonal elements of $\hat{\boldsymbol{W}}_1$ are shrunk toward zero whereas the diagonal elements remain unchanged.

A problem with the MinT reconciliation methods is that it does not guarantee non-negative reconciled forecasts. This poses a problem for applications where the target variable is sales volume. To overcome this using a non-negativity, instead of having a closed form solution as in Equation 3.7 the non-negative reconciliation is formulated as a quadratic programming problem, imposing non-negative constraints [34].

Note that these reconciliation methods are implemented as a post-processing step, and they put no constraint on the methods used to produce the base forecasts [3].

3.3 Metric for Evaluating Forecast Coherency

For comparing the total predicted volume at each hierarchical level accumulated across the entire forecast horizon, I define the metric Percentage Sum of Errors per Level (PSEL) as

$$\text{PSEL}_j = \sum_{i \in \Phi(j)} \frac{\sum_{t=1}^{n} \hat{y}_{i,t} - y_{i,t}}{\sum_{t=1}^{n} y_{i,t}}, \tag{3.11}$$

where $\Phi(j)$ is the set of time series in hierarchical level j, $\hat{y}_{i,t}$ is the forecasted value of entity i at time t, and $y_{i,t}$ is the true value of entity i at time t. Naturally, values closer to zero are better.

The motivation behind the PSEL metric is two-fold. Firstly, none of the commonly used accuracy metrics captures forecast incoherency across aggregation levels. The products sold by The Company are to some extent homogeneous. Thus, it is in the company's interest that the sum of the forecasted sales volume for each product is equal to the total forecasted sales volume. By incorporating the PSEL, the comprehensiveness of the results will be enhanced since it will give insights about the coherency of the forecasts.

Secondly, it is valuable that the forecasted sales volume accumulated over all time steps in the forecast horizon is close to the true accumulated sales volume. Thus, the residuals in the nominator of Equation 3.11 are neither absolute or squared.

To clarify why the details captured by the PSEL metric are of importance, consider a one-step ahead forecast of total sales volume with monthly granularity that supports resource allocation done on a monthly basis, and a 4-step multi-horizon forecast of total sales with weekly granularity that supports week-by-week staff scheduling. It would be desired that the total forecasted sales volume for the weekly forecasts across the entire horizon is equal to the monthly forecasted volume. Such temporal coherency is discussed in [35]. In summary, the PSEL captures details about both spatial and temporal coherency in the forecasts which may be overlooked if evaluating forecasting models simply by e.g MAPE or MASE.

4

The Forecasting Models

The two forecasting models that will be used in this thesis are the Temporal Fusion Transformer (TFT) [19] and the LightGBM [27]. The motivation for choosing the TFT is that studies have shown that the TFT outperforms other common deep learning-based forecasting models [36]. The LightGBM was chosen because it was the go-to model in the M5 accuracy competition, where the top 50 submissions were based on LightGBM, which demonstrated its usefulness in the domain of hierarchical forecasting [4]. Furthermore, a Naïve forecasting model and an exponential smoothing (ES) state-space model [37] will be implemented as benchmarks.

Both the LightGBM and TFT can be trained across multiple time series simultaneously, leveraging the potential benefits of cross-learning. Conversely, the ES models are fit series-by-series, meaning that there are as many ES models as there are time series.

4.1 Benchmarks

In order to assess the actual gain from implementing advanced machine learning models with high complexity, it is common practice to use simpler models as benchmarks and compare their performance. The benchmarks used in this thesis are an exponential smoothing (ES) model based on the state-space framework suggested in [37], as well as the Naïve forecast method.

Fitting exponential smoothing models involves decomposing time series into a trend component, a seasonal component and an error component [37]. Different combinations of these components result in a taxonomy of 24 different ES models. The framework in [37] suggests an automatic model selection approach based on maximizing the likelihood for each ES model, and selecting the model that results in the lowest Akaike's Information Criterion (AIC). The authors state that it is advantageous to use the AIC for model selection because it punishes models with too many parameters, thus handling the trade-off between accuracy and model complexity.

The ES state-space model selection framework was used in combination with the Bottom-Up reconciliation method as a benchmark (therein called ES_bu, herein called ES/BU) in the M5 forecasting competition [4]. ES_bu was the top performing

benchmark out of 50 different benchmarks, and only 415 out of 5507 (7.5 %) of the teams in the competition were able to beat it. Thus, it will serve as a competitive benchmark for the machine learning-based forecasting models in this thesis.

The Naïve forecasting method involves copying the last observed value to predict future values. The h-step multi-horizon Naïve forecast takes the form

$$\hat{y}_{t+\tau} = y_t, \tau \in \{1, ..., h\}. \tag{4.1}$$

The Naïve method autonomously produces coherent forecasts as long as the last observation of each time series is coherent.

4.2 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) is an attention-based deep neural network model designed for multi-horizon forecasting [19]. The TFT was introduced by Google in 2019, implementing several novelties to enhance forecasting accuracy as well as explainability. The TFT combines the power of LSTM-based encoder-decoder models for local temporal processing, and a self-attention mechanism for long-term temporal processing. This allows the TFT to learn both short and long-term temporal dependencies in the data. A schematic view of the architecture of the Temporal Fusion Transformer is shown in Figure 4.1. We can see that the LSTM encoder-decoder described in Section 2.2 is integrated in the TFT architecture, represented by the green and blue rounded boxes. We can also see that and the flow of information in the TFT is similar to the encoder-decoder model.

4.2.1 Direct Forecasting with the TFT

The TFT is a direct forecasting model [18]. Given a pre-specified look back window k and a forecast horizon h, the TFT takes an input sequence consisting of past observations of the target variable $y_{t-k:t} = \{y_{t-k}, ..., y_t\}$, past observations of predictor variables $\boldsymbol{z}_{t-k:t} = \{\boldsymbol{z}_{t-k}...\boldsymbol{z}_t\}$, past and future known variables $\boldsymbol{u}_{t-k:t+h} = \{\boldsymbol{u}_{t-k}, ..., \boldsymbol{u}_{t+h}\}$, as well as static information $\boldsymbol{s}$, and produces an output sequence $\hat{y}_{t+1:t+h} = \{\hat{y}_t, ..., \hat{y}_{t+h}\}$. The forecasts take the form

$$\hat{y}_{t+1:t+h} = f(y_{t-k:t}, \boldsymbol{z}_{t-k:t}, \boldsymbol{u}_{t-k:t+h}, \boldsymbol{s}). \tag{4.2}$$

Note that the forecast expression in Equation 4.2 is similar to that in Equation 2.6, however the functional form of $f(\cdot)$ is different.

The advantage of direct forecasting models is that they can model dependencies between consecutive predicted values $\hat{y}_{i,t}$ and $\hat{y}_{i,t+1}$ since it outputs a sequence of values $\{\hat{y}_{i,t}, ..., \hat{y}_{i,t+h}\}$ [17]. Furthermore, points of significance in time series data are usually identified in relation to adjacent values [19]. The sequence-to-sequence nature of the TFT allows the TFT to capture such points of significance.

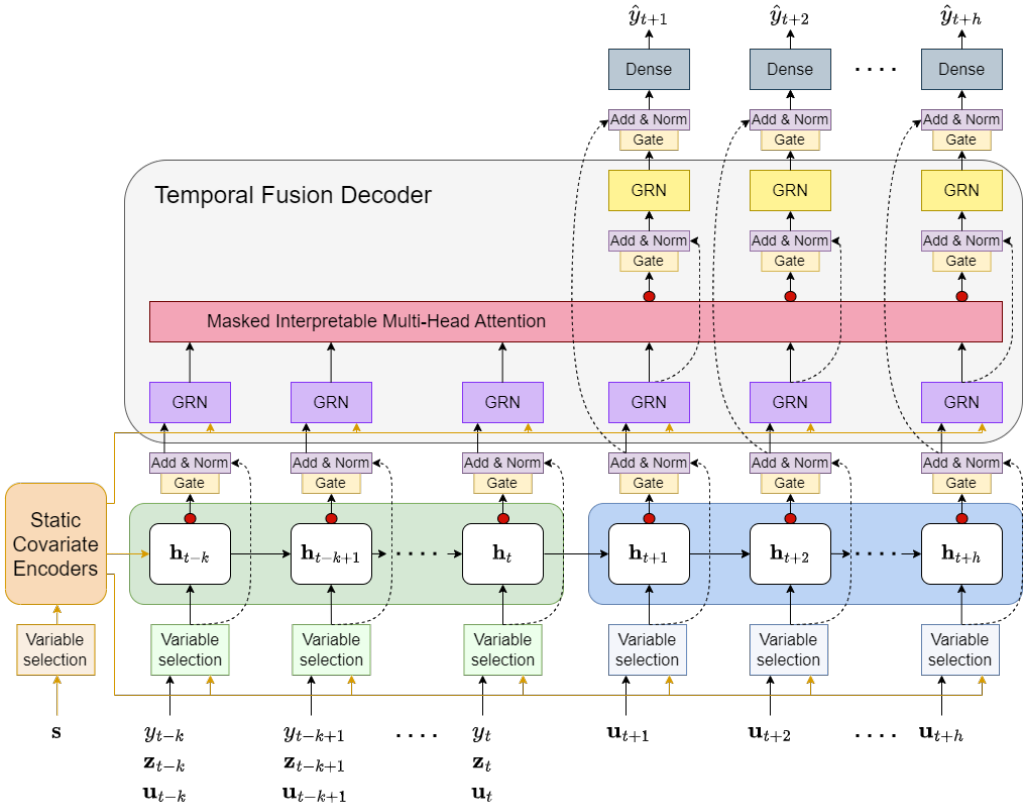


Figure 4.1: High level architecture of the Temporal Fusion Transformer. Arrows represent flow of information, dashed arrows represent residual connections. All blocks that share the same color share weights. Red circles indicate layers where drop-out regularization is used during training.

4.2.2 Model Architecture

Figure 4.1 shows the high level architecture of the Temporal Fusion Transformer. Blocks that share the same color also share the same weights, reducing the number of trainable parameters. The TFT essentially consists of five components, (1) Static Covariate Encoders,(2) Gated Residual Networks, (3) Variable Selection Networks, (4) LSTM Encoder-Decoder, (5) Temporal Fusion Decoder.

Static Covariate Encoders allow the TFT to integrate static metadata by producing context vectors that are used at various steps throughout the network. The static covariate encoders produce four different context vectors, $\boldsymbol{c}_s$ is used in the variable selection of temporal features that are fed into the LSTM encoder-decoder, $\boldsymbol{c}_c$ and $\boldsymbol{c}_h$ are used to initialize the hidden and cell states of the LSTM encoder-decoder. Finally, the context vector $\boldsymbol{c}_e$ is used in the temporal fusion decoder. The flow of these context vectors are displayed as orange arrows in Figure 4.1.

Gated Residual Networks (GRN) are used to give the TFT adaptable depth by allowing the information flow to skip parts of the network that are unnecessary. For example if the input data is noisy, non-linear processing steps can impair the predictive performance [19]. During training, the gating mechanisms give the TFT the

flexibility to skip non-linear processing steps, adapting its complexity for different tasks.

The gating mechanisms are designed as Gated Residual Networks (GRNs), shown in Figure 4.2. The GRN takes as inputs the primary input $\boldsymbol{a}$ (which differs depending on where in the network the GRN is located) as well as a context vector $\boldsymbol{c}$ computed by the static covariate encoder. The mathematical formulation of the GRN is

$$\begin{aligned} \text{GRN}_\omega(\boldsymbol{a}, \boldsymbol{c}) &= \text{LayerNorm}(\boldsymbol{a} + \text{GLU}_\omega(\boldsymbol{\eta}_1)), \\ \boldsymbol{\eta}_1 &= \boldsymbol{W}_{1,\omega}\boldsymbol{\eta}_2 + \boldsymbol{b}_{1,\omega}, \\ \boldsymbol{\eta}_2 &= \text{ELU}(\boldsymbol{W}_{2,\omega}\boldsymbol{a} + \boldsymbol{W}_{3,\omega}\boldsymbol{c} + \boldsymbol{b}_{2,\omega}), \end{aligned} \tag{4.3}$$

where LayerNorm is the layer normalization proposed by [38], GLU and ELU denotes the Gated and Exponential Linear Units, respectively [39], [40]. The vectors $\boldsymbol{\eta}_1$ and $\boldsymbol{\eta}_2$ are the intermediate layers of the GRN. The proposed advantage of using the

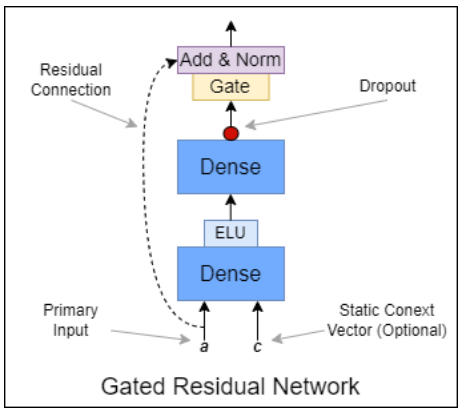


Figure 4.2: High level architecture of the Gated Residual Network.

GRN as a building block in the TFT is that the network can learn whether the nonlinear processing of the $\text{GLU}_\omega(\boldsymbol{\eta}_1)$ term is necessary. If not, it remains close to zero, which means that the primary input $\boldsymbol{a}$ essentially skips the layer entirely.

Variable Selection Networks (VSN) are used to determine the relative importance of all available inputs. In time series forecasting, there are usually several input features whose relation to the target variable is unknown in advance. Thus, some input features may have very low predictive value, and including such features may impair the predictive performance of the model [19]. The VSN allow the TFT to filter out unnecessary inputs. The TFT uses a separate VSN for each of the different input types $\boldsymbol{s}$, past inputs $y_{t-k:t}$, $\boldsymbol{z}_{t-k:t}$, $\boldsymbol{u}_{t-k:t}$, and future known inputs $\boldsymbol{u}_{t+1:t+k}$, distinguished by color in Figure 4.1.

The architecture of the VSN is shown in Figure 4.3. Similar to the GRN, the VSN takes a context vector $\boldsymbol{c}$ as input, which can increase or decrease the variable selection weights $v_\chi^{(j)}$ for each feature. Assume that there are m different time-varying input features fed to the VSN. Each input feature j at time t is transformed into a (d_{model})-dimensional vector $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{model}}$, using linear transformations for continuous variables and entity embeddings for categorical variables. The vector

$\Xi_t = \left[\boldsymbol{\xi}_t^{(1)^T}, ..., \boldsymbol{\xi}_t^{(m)^T}\right]^T$ denotes the collection of encoded features at time t, which is used as input together with the context vector $\boldsymbol{c}$ to determine the variable selection weights $\boldsymbol{v}_\chi$. As shown in Figure 4.3, each transformed input $\boldsymbol{\xi}_t^{(j)}$ is further transformed into $\tilde{\boldsymbol{\xi}}_t^{(j)}$ by a GRN before being multiplied with their corresponding selection weight. Finally, a weighted feature representation

$$\tilde{\boldsymbol{\xi}}_t = \sum_{j=1}^{m} v_{\chi t}^{(j)} \tilde{\boldsymbol{\xi}}_t^{(j)} \tag{4.4}$$

is computed for each time step, which is then passed as input to the LSTM encoder-decoder.

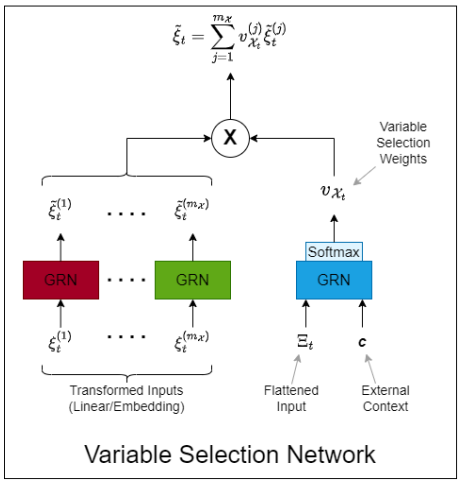


Figure 4.3: High level architecture of the Variable Selection Network.

Besides filtering out noisy input data, the VSN facilitates explainability by comparing the relative sizes of the variable selection weights for different features.

The **LSTM Encoder-Decoder** is a sequence-to-sequence layer that processes short-term temporal relations in the data, allowing the TFT to learn local patterns. As shown in Figure 4.1, a sequence of past inputs $y_{t-k:t}$, $\boldsymbol{z}_{t-k:t}$, $\boldsymbol{u}_{t-k:t}$ are passed to the encoder through the variable selection networks where the input is transformed into weighted feature representations $\tilde{\boldsymbol{\xi}}_{t-k:t}$. The sequence is then processed by the encoder, whose final hidden state is then passed to the decoder. Future known inputs $\boldsymbol{u}_{t+1:t+k}$ pass through another set of VSNs, and the transformed values $\tilde{\boldsymbol{\xi}}_{t+1:t+k}$ are then passed to the decoder, where they are merged with the past transformed inputs $\tilde{\boldsymbol{\xi}}_{t-k:t}$.

The cell state and hidden state of the first LSTM layer in the encoder is initialized using the context vectors $\boldsymbol{c}_c$ and $\boldsymbol{c}_h$ produced by the static covariate encoder, which enables the incorporation of static information for local processing in the encoder-decoder. The LSTM encoder-decoder generates a sequence of temporal features $\boldsymbol{\phi}(t,n) \in \{\boldsymbol{\phi}(t,-k), ..., \boldsymbol{\phi}(t,h)\}$ that is fed to the temporal fusion decoder with an additional skip connection such that

$$\tilde{\boldsymbol{\phi}}(t,n) = \text{LayerNorm}\Big(\tilde{\boldsymbol{\xi}}_{t+n} + \text{GLU}_{\tilde{\phi}}(\phi(t,n))\Big), \tag{4.5}$$

where $n \in [-k, h]$ serves as a positional index.

The **Temporal Fusion Decoder** is responsible for capturing long term temporal dependencies in the data. It uses a layer of GRNs to combine static information with the temporal features from the LSTM encoder-decoder to compute static-enriched temporal features

$$\boldsymbol{\theta}(t, n) = \mathrm{GRN}_{\theta}\Big(\tilde{\boldsymbol{\phi}}(t, n), \boldsymbol{c}_e\Big), \tag{4.6}$$

where $\boldsymbol{c}_e$ is a static context vector produced by the static covariate encoder. The enriched temporal features are then grouped into a single matrix

$$\boldsymbol{\Theta}(t) = [\boldsymbol{\theta}(t, -k), ..., \boldsymbol{\theta}(t, h)]^T \tag{4.7}$$

that is passed to the Interpretable Multi-Head Attention block. The Multi-Head Attention uses a modified version of the *Query, Key, Value* structure used in regular transformers to capture long term dependencies that are difficult for RNN based architectures to learn [19]. The output from this block is a matrix $\boldsymbol{B}(t) = [\boldsymbol{\beta}(t, -k), ..., \boldsymbol{\beta}(t, h)]$ that is passed through a skip connection layer

$$\boldsymbol{\delta}(t, n) = \mathrm{LayerNorm}\Big(\boldsymbol{\theta}(t, n) + \mathrm{GLU}_{\delta}(\boldsymbol{\beta}(t, h))\Big), \tag{4.8}$$

allowing the network to decide the contribution of the Multi-Head Attention processing. The sequence $\{\boldsymbol{\delta}(t, -k), ..., \boldsymbol{\delta}(t, h)\}$ is then passed to a new set of GRNs to produce the final forecasts $\{\hat{y}_t, ..., \hat{y}_{t+h}\}$.

4.3 LightGBM

The LightGBM is an augmented implementation of a gradient boosting decision tree (GBDT), developed by researchers at Microsoft in 2017 [27]. While LightGBM is based on the fundamental principles of gradient boosted decision trees described in Section 2.3, it incorporates two novel techniques designed to enhance its efficiency and scalability. Furthermore, the LightGBM implementation incorporates L1 and L2 regularization by adding penalty terms to the loss function $\mathcal{L}$ that is minimized during training, which helps to prevent overfitting.

The novel ideas introduced in the LightGBM implementation were *Gradient-based One-Sided Sampling* (GOSS) which allows the model to down-sample inputs that are accurately predicted, whilst retaining the prediction accuracy. The second novel idea was *Exclusive Feature Bundling* (EFB) which allowed the model to bundle features together, which further reduces the computational complexity.

4.3.1 Recursive Forecasting with the LightGBM

The LightGBM is a recursive model, which means that it is optimized to generate one-step ahead forecasts. To create an h-step multi-horizon forecast using the LightGBM, the one-step forecasts are iteratively fed back into the model so that subsequent forecasts can use the information from preceding steps.

At time t, the LightGBM generates the one-step ahead forecast

$$\hat{y}_{t+1} = f(\boldsymbol{x}_t), \tag{4.9}$$

where $\boldsymbol{x}_t = (\boldsymbol{z}_t, \boldsymbol{u}_t, \boldsymbol{s})$ is the set of input feature values available at time t. In order to produce h-step multi-horizon forecasts the prediction function $f(\cdot)$ is applied recursively, hence the name recursive forecasting.

At prediction time, the LightGBM reads the data in a tabular format such that a single value of a given input feature j is used as input, rather than a sequence of values for each feature as in the TFT. Thus, the LightGBM is dependent on feature engineering to capture temporal dependencies in the data. Common features are lagged values and statistics computed on a rolling look-back window, such as the mean or median [31].

Using a one-step lagged value and a k steps moving average as an example, the feature vector $\boldsymbol{x}_t$ in Equation 4.9 is defined by

$$\boldsymbol{x}_t = [y_t, \mu_{t-k:t}], \tag{4.10}$$

where $\mu_{t-k:t} = \frac{1}{k}\sum_{\tau=t-k}^{t} y_\tau$. When producing multi-horizon forecasts, each prediction must iteratively be fed back to the model in order to re-compute the input features. Let T be the last forecast origin, then the one-step ahead forecast is defined by

$$\hat{y}_{T+1} = f(\boldsymbol{x}_T) = f(y_T, \mu_{T-k:T}). \tag{4.11}$$

In order to produce the two-step ahead forecast $\hat{y}_{T+2}$, the feature vector $\boldsymbol{x}$ must be re-computed given $\hat{y}_{T+1}$. The new feature vector takes the form

$$\boldsymbol{x}_{T+1} = [\hat{y}_{T+1}, \hat{\mu}_{T-k+1:T+1}], \tag{4.12}$$

where

$$\hat{\mu}_{T-k+1:T+1} = \frac{1}{k}\left(\hat{y}_{T+1} + \sum_{\tau=T-k+1}^{T} y_\tau\right). \tag{4.13}$$

Looking at Equations 4.12 and 4.13, the potential drawback of error accumulation in recursive forecasting methods becomes apparent. If $\hat{y}_{T+1}$ is biased, then the features computed in $\boldsymbol{x}_{T+1}$ will be biased, propagating the bias to each consecutive one-step ahead forecast.

4.3.2 Training Procedure

The fundamental principle of gradient boosting is described in Section 2.3. Thus, the focus of this section is the construction of the weak regression trees, as well as describing the novelties introduced in the LightGBM.

When fitting a regression tree, the training data is split based on the most informative feature in a top-down approach such that the feature split with highest information gain constitutes the root of the tree. Additional splits are then made in

a descending order of information gain. One way of measuring information gain is by measuring the variance after splitting. Given a training data set O, the variance gain of splitting the data at the point d of feature j is defined as

$$V_{j|O}(d) = \frac{1}{n_O}\left(\frac{(\sum_{x_i \in O: x_{ij} \leq d} g_i)^2}{n^j_{l|O}(d)} + \frac{(\sum_{x_i \in O: x_{ij} > d} g_i)^2}{n^j_{r|O}(d)}\right), \tag{4.14}$$

where n_O is the total number of training samples, $n^j_{l|O}(d)$ is the number of samples $[x_i \in O | x_{i,j} \leq d]$, i.e the number of samples x_i whose value of the feature j is smaller than or equal to the splitting point d, and $n^j_{r|O}(d)$ is the number of samples $[x_i \in O | x_{i,j} > d]$.

To find the optimal split point d_j of feature j, the algorithm must first evaluate all possible split points, which is computationally expensive. The novel approach proposed in the implementation of LightGBM uses GOSS to speed up this process [27]. First, training instances are ranked by their corresponding gradients, or pseudo-residuals, in descending order. The top a %, i.e the a % instances with the largest pseudo-residuals form a subset A, and from the remaining instances A^c a subset B with size $|B| = b \times |A^c|$ is randomly sampled. The data instances in the subset $A \cup B$ are then stored in discrete bins to reduce the number of possible split points, a technique called the histogram-based algorithm [27]. A modified version of Equation 4.14 is then applied to the subset $A \cup B$ to compute the estimated variance gain $\tilde{V}_j(d)$.

In simple terms, instances in the training data with small gradients are already well fit, thus they are paid less attention. The instances in subset A are those that contribute the most to the loss are kept. The authors showed that GOSS reduces computation cost greatly without losing much training accuracy [27]. Furthermore, in order to speed up training the LightGBM uses EFB to reduce computational cost by bundling mutually exclusive features such as one-hot encoded categorical variables.

5

Method

This section describes the methods and procedures used during implementation of the forecasting models, as well as a description of the data. The final section of this chapter describes challenges that arose throughout the implementation phase. Full training details are provided in Appendix A.

5.1 The Data

The data set used in this thesis contains time series of historical sales of different products for The Company, that can be structured hierarchically. The most important aggregation levels are the top and bottom, representing the total sales and sales per product, respectively. The intermediate aggregation levels are created by aggregating sales of individual products on shared attributes, implying that the disaggregation is not unique. Therefore, strictly speaking the time series are grouped, not hierarchical. However, the same principles that apply to hierarchical time series apply to grouped time series.

The data set contains 107 observations of historical sales on a weekly basis (approximately two years of data) for a wide range of products. The set of individual time series for each product comprises the bottom level of the hierarchy. The products have shared attributes which are used to aggregate the data and create two intermediate levels in the hierarchy. The top level in the hierarchy is the aggregated sales of all products. As such, the time series are structured in a 4-level hierarchy, where aggregation level 1 corresponds to the total sales volume of The Company, and the bottom level 4 corresponds to the sales volume of each individual product.

The total hierarchy comprises a single time series at aggregation level 1, four time series at aggregation level 2, three time series at aggregation level 3, and 575 time series at aggregation level 4, amounting to a cumulative total of 583 time series.

The bottom level in the data contains a lot of intermittent time series, characterized by having several zero-valued observations. An example of an intermittent time series is shown in Figure 5.1. Intermittent time series are difficult to forecast since they do not display any observable patterns.

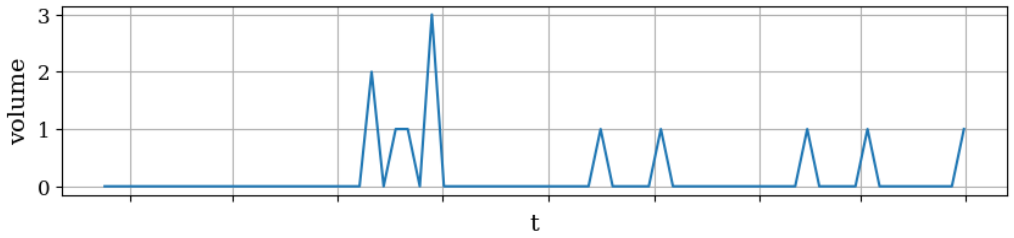


Figure 5.1: A visual representation of an intermittent time series.

5.1.1 Preprocessing

Naturally, time series at different levels in the hierarchy will have different scales due to aggregation. To ensure that all time series were of similar scale, the time series were downscaled using what I called *n-leaves* rescaling. The *n-leaves* rescaling involved dividing each aggregated time series by the number of leaves that connect to it such that

$$y_t^{(p)} \leftarrow \frac{y_t^{(p)}}{|\Omega(p)|} \tag{5.1}$$

where $y_t^{(p)}$ is the value of the time series in node p, and $\Omega(p)$ is the set of leaf nodes of the sub-tree rooted at the node p and $|\Omega(p)|$ is the cardinality of $\Omega(p)$.

5.1.2 Feature Engineering

Both the LightGBM and TFT are able to incorporate data from different sources, including temporal features. Given that the observations in the time series are of weekly granularity, the following temporal features were extracted:

1. Week of the year,
2. Month of the year,
3. Quarter of the year,
4. Year.

The following statistical features were computed from the raw time series data:

1. {1, 2, 3, 4} steps lagged values,
2. {4, 13} steps rolling mean values,
3. {4, 13} steps rolling median values,
4. {4, 13} steps rolling count of zero-valued observations.

Initial tests showed that the statistical features did not contribute to fitting the TFT. Figure 5.2 shows the variable selection weights $\boldsymbol{v}_\chi$ extracted from the Variable Selection Network responsible for time-varying unknown inputs. Thus, to reduce the computational cost of training the TFT, the statistical features were only used for fitting the LightGBM.

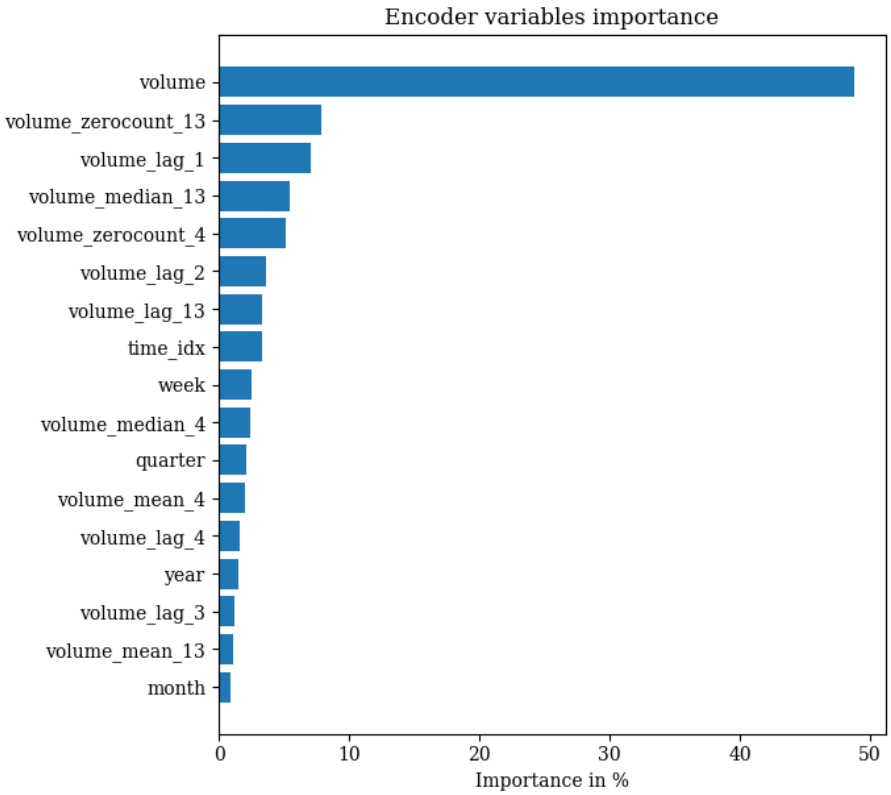


Figure 5.2: Bar plot of feature importance from trained TFT. Last number in each feature indicates the length of the rolling window on which the statistic was computed. This bar plot indicates that statistics computed from the raw data are not of significant importance.

5.2 Cross-Validation

Hyperparameters were tuned using the time-series split cross-validation with three disjoint validation folds and one test fold of size $h = 4$. The last horizon $t \in (t_{max} - h, t_{max}]$ was used as a hold-out test set for comparing final accuracy of the different forecasting models. Thus, the validation folds consisted of observations at times $t \in (t_{max} - i \cdot h, t_{max} - (i-1) \cdot h]$, $i = 2, 3, 4$. All available data preceding the validation fold was used to train the models, indicating that more training data was used for smaller values of i. An illustrative depiction of the data splits is shown in Figure 5.3

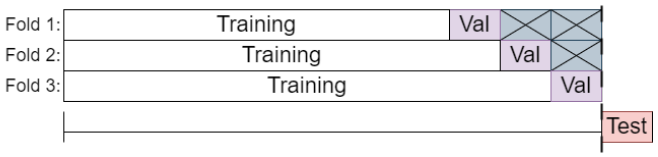


Figure 5.3: Illustrative depiction of the data splits used for cross-validation.

For the sake of brevity, the cross-validation procedure will be explained using only TFT. However, the procedure was identical for the LightGBM. The hyperparameters

of the TFT were tuned once for each validation fold, such that TFT_1 was tuned on fold 1 etc., resulting in the three models TFT_1, TFT_2, and TFT_3 with different sets of hyperparameters. Each model was then trained and validated on all three folds, and the model with the lowest average MAE across all three folds, named TFT_{opt}, was selected for evaluation on the unseen test set. Furthermore, an ensemble model TFT_μ was created by averaging the forecasts of the three suggested models such that

$$f_{\text{TFT}_\mu}(\cdot) = \frac{1}{3}\sum_{i=1}^{3} f_{\text{TFT}_i}(\cdot), \tag{5.2}$$

where $f_{\text{TFT}}(\cdot)$ is the prediction function of the TFT.

Altogether, four models are selected in the cross-validation procedure, TFT_{opt}, TFT_μ, LightGBM_{opt} and LightGBM_μ.

5.3 Hardware / Software

The TFT and LightGBM were implemented in `Python`. For the TFT I used the `PyTorch_Forecasting` library [41], which also provided an easy implementation for the Naïve forecast method. For the LightGBM I used the `lightgbm` library [42]. The ES/BU model was implemented in `R`, using the `forecast` package to produce the base forecasts [43]. For the reconciliation methods, I used the `hierarchicalforecast` package in `Python` [44].

All models were trained on a laptop with an 11th Gen Intel(R) Core(TM) i7 @ 2.80 GHz processor with 4 cores and 16 GB RAM.

5.4 Hyperparameter Tuning

The hyperparameter tuning was done with the `optuna` library, using a Tree-structured Parzen Estimator algorithm for exploring the hyperparameter space [45]. The `PyTorch_Forecasting` has an integrated method `optimize_hyperparameters()` for hyperparameter tuning with optuna.

To obtain stable results when tuning the hyperparameters of the LightGBM, the learning rate was kept at a constant value $\eta = 0.01$. Furthermore, the bagging and feature fractions were set to 1.0 to remove the randomness associated with bagging and feature sampling, and the max_bins parameter was set to 10000 to reduce the randomness when creating discrete bins in the histogram-based algorithm.

Table 5.1 contains the hyperparameters that were tuned for the LightGBM, as well as the corresponding limits and distributions. The λ_L1 and λ_L2 parameters control the L1 and L2 penalty terms of the objective function, num_leaves control the maximum number of leaves allowed in a single weak learner, and the min_child_samples parameter controls the minimum number of data instances required in a node to split the data further. The hyperparameter tuning for LightGBM took approximately 2.5 minutes.

Parameter	Distribution	Limits
λ_L1	Log-Uniform	$[10^{-8}, 10]$
λ_L2	Log-Uniform	$[10^{-8}, 10]$
num_leaves	Integer Uniform	[2, 64]
min_child_samples	Integer Uniform	[5, 100]

Table 5.1: Hyperparameter tuning settings for LightGBM.

Table 5.2 show the hyperparameters and corresponding limits and distributions that were tuned for the TFT. The number of attention heads in the TFT was fixed at 1. The gradient_clip_val controls the maximum absolute value allowed for gradients, i.e gradients are capped at this magnitude. The hidden_size parameter controls the size of the hidden states in the TFT, and the hidden_continuous_size controls the size of the embeddings of the continuous input variables. The dropout parameter controls the dropout fraction in the TFT layers where dropout is applied. The hyperparameter tuning for the TFT was capped at 8 hours, and this constraint was invoked every time.

Parameter	Distribution	Limits
gradient_clip_val	Log-Uniform	[0.01, 1]
hidden_size	Integer Uniform	[4, 32]
hidden_continuous_size	Integer Uniform	[4, 32]
learning_rate	Uniform	[0.001, 0.01]
dropout	Uniform	[0.1, 0.3]

Table 5.2: Hyperparameter tuning settings for TFT.

The encoder length (look back window) of the TFT was set to $k = 52$ and the decoder length (forecast horizon) was set to $h = 4$.

5.5 Forecast Evaluation Metrics

In hierarchical forecasting, it is not desired that forecast errors on the top aggregation level is given equal weight as the errors of a time series in the bottom level. Thus, I will use a weighted MASE (WMASE), similar to the weighted RMSSE that was used in the M5 competition [4]. The WMASE is defined by

$$\text{WMASE} = \sum_{i=1}^{N} w_i \times \text{MASE}_i, \tag{5.3}$$

where each weight w_i is computed by calculating the fraction that entity i contributed to the total sales over the last year of the training data. That means that the most aggregated time series corresponding to total sales will have a weight of 1.0.

Some products were not sold during the period for when the weights were computed, thus given a zero weight. However, this had a negligible effect on the results.

Moreover, I will use the PSEL metric

$$\text{PSEL}_j = \sum_{i \in \Phi(j)} \frac{\sum_{t=1}^{n} \hat{y}_{i,t} - y_{i,t}}{\sum_{t=1}^{n} y_{i,t}}, \tag{5.4}$$

defined in Section 3.3 to measure forecast incoherency and accumulated predicted volume error across the forecast horizon.

5.6 Implementation

The hyperparameter tuning and training of the LightGBM and TFT models was performed using the full hierarchy of the data simultaneously. Having selected the optimal hyperparameters for TFT_{opt}, TFT_{μ}, LightGBM_{opt} and LightGBM_{μ}, each model was trained and validated on fold 3 visualized in Figure 5.3, corresponding to data in the range $t \in [1, t_{max} - h]$. Thereby, the models were trained on the maximum amount of available data before producing the final forecasts.

The base forecasts of each model were then computed for the hold-out test set containing observations from the final $h = 4$ time steps. The base forecasts were then reconciled using the Bottom-Up (BU), Minimum Trace Ordinary Least Squares (MinT OLS), and Minimum Trace Weighted Least Squares (MinT WLS_s) methods. The ES/BU forecasts were produced by fitting an individual exponential smoothing model for each time series in the bottom level in the range $t \in [1, t_{max} - h]$ using the state-space framework, then reconciling the base forecasts using the BU method. The Naïve forecasts are coherent by design since the Naïve method is to predict the last observed value for each time step in the forecast horizon, and therefore does not need to be reconciled.

In different applications of time series forecasting, it is possible that only a specific level in the hierarchy is of interest. To investigate whether there is an advantage in structuring the data as hierarchical time series and train a forecasting model on the entire hierarchy rather than the single level of interest, the cross-validation and hyperparameter search was repeated for each individual level, resulting in a set of single-level optimized models

$$\text{TFT}_{opt,j},\ \text{TFT}_{\mu,j},\ \text{LightGBM}_{opt,j},\ \text{LightGBM}_{\mu,j},\ j \in \{1, ..., 4\},$$

where j is the hierarchical level.

As a final step before evaluating the models, negative forecast were adjusted to 0 and the forecasts were rounded to the nearest integer value, since the sales volume is a non-negative discrete target.

5.7 Challenges

Given the limited availability of data, efficiently managing the trade-off between data utilization during training and testing is a significant challenge, especially with time series data considering that independency between observations can never be assumed. The more data used for validation and testing, the less is available for training the models. Initially, a three-way holdout cross-validation method was considered, but initial findings revealed a high degree of instability concerning the generalization ability of the trained models. To attain more consistent results without compromising the training data, a time series split cross-validation approach was adopted for model selection.

The way that the cross-validation method was designed, the models whose hyperparameters were optimized on the third fold (see Figure 5.3) had access to the data in the first and second fold during hyperparameter tuning. A natural concern was that, as a consequence, the TFT_3 and LightGBM_3 would exhibit superior performance compared to the first and second models. However, no such pattern was observed. Table 5.3 shows the specific models that were selected for each hierarchical level.

Aggregation Level	Selected *opt* Model	
Full Hierarchy	TFT_2	LightGBM_2
Level 1	TFT_2	LightGBM_1
Level 2	TFT_1	LightGBM_3
Level 3	TFT_2	LightGBM_1
Level 4	TFT_2	LightGBM_3

Table 5.3: Summary of which TFT and LightGBM models that were selected as *opt* models for each aggregation level.

The fact that the TFT_3 and LightGBM_3 models did not consistently demonstrate superior performance across all three folds indicates that although this cross-validation method favored models with more data access, the advantage was not decisive.

Another challenge that emerged during the implementation phase was that the Mint WLS_V and Shrink methods proved inapplicable for the data used in this thesis. The WLS_V and Shrink methods involve using the in-sample base forecast errors to estimate the covariance matrix of the out-of-sample base forecast errors. However, in the original paper on minimum trace reconciliation the experiment was conducted using an ARIMA model [11]. Conversely, when using machine learning-based models it is infeasible to estimate out-of-sample forecasting errors with in-sample forecasting errors, since a machine-learning model can learn to fit the training data perfectly.

Thus, the base forecast errors from the validation set were used as a less biased estimator of the out-of-sample errors. However, several products exhibited zero sales throughout the entire validation set, and the TFT and LightGBM models accurately predicted zero. Thus, the base forecast variance over the validation set

was zero in many instances. Since the covariance between any random variable and a constant is zero, the base forecast covariance matrix had several zero-columns and was therefore not invertible. Thus, only the Bottom-Up, OLS, and MinT WLS_s reconciliation methods were used in this thesis.

A final challenge that arose during the implementation was that since sales volume is a discrete target, the final forecast had to be rounded to integer values. However, simply rounding the forecasts defeated the purpose of reconciling the forecast since several bottom level time series were rounded to zero, whereas the aggregated time series had a rounding error of ± 1. Consequently, even after applying reconciliation, the forecasts were not coherent. Thus, a more sophisticated integer optimization step is warranted to keep the coherency intact.

6

Results

The resulting forecast accuracy in terms of WMASE is presented for each model, with and without reconciliation, in Table 6.1. The accuracy for each model is presented at each aggregation level in the time series hierarchy, as well as the average accuracy across all levels. The columns for each aggregation level contains the summed WMASE for all time series in the particular level. The models are ranked in descending order by average WMASE, such that the best performing model is at the top. The rightmost column in Table 6.1 contains the percentage improvement in average WMASE for each model compared to the ES/BU benchmark. The column-wise minimum values are displayed with bold text. The bottom row of Table 6.1 shows the average WMASE across all forecasting models for each aggregation level.

Rank	Model	Aggregation Level				Average	Improvement (%)
		1	2	3	4		
1	TFT_{μ}/MinT WLS_s	**0.231**	0.340	**0.262**	0.540	**0.343**	43.1
2	TFT_{μ}	0.254	**0.312**	0.424	0.544	0.383	36.4
3	TFT_{μ}/MinT OLS	0.238	0.359	0.368	0.630	0.399	33.9
4	$LightGBM_{opt}$	0.421	0.372	0.276	0.542	0.403	33.2
5	$LightGBM_{opt}$/MinT WLS_s	0.313	0.394	0.357	0.552	0.404	33.0
6	$LightGBM_{\mu}$/MinT WLS_s	0.314	0.407	0.386	**0.530**	0.409	32.1
7	$LightGBM_{\mu}$	0.337	0.441	0.324	0.540	0.410	31.9
8	$LightGBM_{\mu}$/MinT OLS	0.246	0.450	0.409	0.544	0.412	31.6
9	$LightGBM_{opt}$/MinT OLS	0.321	0.384	0.357	0.620	0.420	30.3
10	Naïve	0.499	0.453	0.464	0.586	0.501	17.0
11	TFT_{μ}/BU	0.494	0.547	0.474	0.544	0.515	14.6
12	$LightGBM_{opt}$/BU	0.653	0.588	0.603	0.542	0.596	1.1
13	ES/BU	0.587	0.559	0.552	0.713	0.603	-
14	$LightGBM_{\mu}$/BU	0.709	0.632	0.656	0.540	0.634	-5.2
15	TFT_{opt}/BU	0.548	0.731	0.615	0.759	0.664	-10.1
16	TFT_{opt}/MinT WLS_s	0.580	0.628	0.720	0.809	0.684	-13.5
17	TFT_{opt}	0.419	0.414	1.331	0.759	0.731	-21.2
18	TFT_{opt}/MinT OLS	0.682	0.786	0.920	0.818	0.801	-32.9
Average WMASE per Level:		0.436	0.489	0.528	0.617		

Table 6.1: Performance of each model in terms of WMASE, smaller values are better. Results presented are accumulated WMASE on each aggregation level, and on average. Improvement column specifies percentage improvement in average WMASE compared to the ES/BU benchmark. Column-wise minimum values are displayed in **bold**.

The results in Table 6.1 show that the best performing model was the TFT_{μ}/Mint WLS_s with an average WMASE of 0.343 and an improvement of 43.1 % over ES/BU. The second and third place models are also based on TFT_{μ}, indicating its superior performance. Conversely, all models based on TFT_{opt} performed worse than the ES/BU benchmark. All models based on LightGBM expect those using Bottom-Up reconciliation performed better than the benchmark, with similar accuracy scores.

Table 6.2 shows the Percentage Sum of Errors per Level (PSEL) of predicted volume for each model at each hierarchical level, computed over the entire forecast horizon. The rank of the models follows the same order as in Table 6.1, hence there is not necessarily a descending order of superiority in Table 6.2. Incoherent base forecasts are highlighted in gray. The numbers presented in this table the results prior to rounding forecasts to integer values, to clearly demonstrate the purpose of reconciliation.

Rank	Model	Aggregation Level			
		1	2	3	4
1	TFT_{μ}/MinT WLS_s	+ 0.3	+ 0.3	+ 0.3	+ 0.3
2	TFT_{μ}	+ 13.2	- 1.6	+ 28.9	- 39.5
3	TFT_{μ}/MinT OLS	+ 14.7	+ 14.7	+ 14.7	+ 14.7
4	$LightGBM_{opt}$	+ 33.6	+ 1.2	+ 1.5	- 52.1
5	$LightGBM_{opt}$/MinT WLS_s	- 4.0	- 4.0	- 4.0	- 4.0
6	$LightGBM_{\mu}$/MinT WLS_s	- 8.1	- 8.1	- 8.1	- 8.1
7	$LightGBM_{\mu}$	+ 26.8	+ 6.2	- 9.0	- 56.6
8	$LightGBM_{\mu}$/MinT OLS	+ 16.0	+ 16.0	+ 16.0	+ 16.0
9	$LightGBM_{opt}$/MinT OLS	+ 21.7	+ 21.7	+ 21.7	+ 21.7
10	Naïve	- 39.8	- 39.8	- 39.8	- 39.8
11	TFT_{μ}/BU	- 39.5	- 39.5	- 39.5	- 39.5
12	$LightGBM_{opt}$/BU	- 52.1	- 52.1	- 52.1	- 52.1
13	ES/BU	+ 46.8	+ 46.8	+ 46.8	+ 46.8
14	$LightGBM_{\mu}$/BU	- 56.6	- 56.6	- 56.6	- 56.6
15	TFT_{opt}/BU	+ 7.5	+ 7.5	+ 7.5	+ 7.5
16	TFT_{opt}/MinT WLS_s	+ 45.5	+ 45.5	+ 45.5	+ 45.5
17	TFT_{opt}	+ 33.4	+ 19.9	+ 104.7	+ 7.5
18	TFT_{opt}/MinT OLS	+ 54.4	+ 54.4	+ 54.4	+ 54.4

Table 6.2: PSEL for each model over the forecast horizon. Models are ordered by rank in terms of average WMASE presented in Table 6.1. Gray rows indicate base forecasts that are not coherent. Values closer to zero are better.

The results in Table 6.2 present a dimension of the forecasts that is not captured by the WMASE, namely the accumulated volume error across the forecast horizon and across aggregation levels. For example, looking at the $LightGBM_{opt}$ (rank 4) and $LightGBM_{opt}$/MinT WLS_s (rank 5) in Table 6.1 the difference in average WMASE is minimal, indicating that the models are equal in performance. However, looking at the same two models in Table 6.2, we see that the PSEL of $LightGBM_{opt}$

at aggregation level 1 is +33.6 % and -52.1 % at level 4, whereas the PSEL of LightGBM$_{opt}$/MinT WLS$_s$ is -4 % at all levels. Similar results are observed for all models after applying the MinT WLS$_s$ reconciliation, except for the TFT$_{opt}$.

6.1 Base-Model Forecasts

This section contains a sample of the forecasts produced with TFT$_{opt}$, LightGBM$_{opt}$ and the benchmark model ES/BU. For reasons of confidentiality, the volume on the y-axis has been removed for all aggregated time series. However, the volume is displayed for the level 4 bottom-level time series to demonstrate the randomness of the sales patterns for individual products, as well as showing the scale difference between products. In all figures, the forecast origin is marked with a black vertical dashed line at time $t + 0$, and a window of time from $t - 12$ to $t + 4$ is displayed for each time series. The target time series is dashed between the final observation in the training data and the first observation in the test data to clarify that the forecasting models did not have access to that information.

Figure 6.1 illustrates the time series for aggregation level 1, which is the total sales volume of all products. The figure shows that the LightGBM$_{opt}$ and TFT$_{opt}$ forecasts are similar in performance and that both models had lower WMASE than ES/BU.

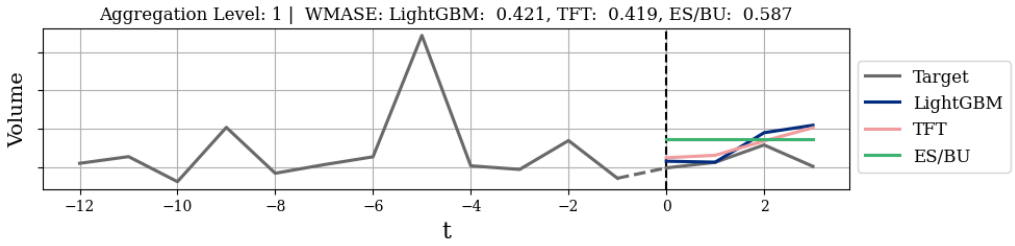


Figure 6.1: Forecasts from TFT$_{opt}$, LightGBM$_{opt}$ and ES/BU on the time series at aggregation level 1. The dashed black vertical line marks the forecast origin. The dashed gray line is the connection between the training data and test data.

Figure 6.2 shows one of the time series at aggregation level 2. The LightGBM$_{opt}$ is slightly better than the ES/BU, whereas the TFT$_{opt}$ has the highest error.

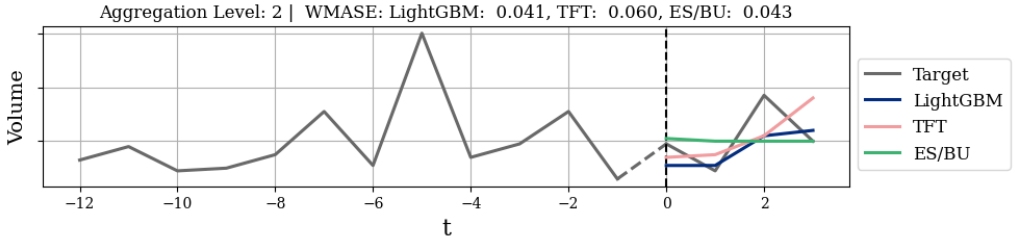


Figure 6.2: Forecasts from TFT$_{opt}$, LightGBM$_{opt}$ and ES/BU on one of the time series at aggregation level 2. The dashed black vertical line marks the forecast origin. The dashed gray line is the connection between the training data and test data.

Figure 6.3 depicts a time series from the third aggregation level. The graph shows that the TFT_{opt} forecast has a large positive bias, with a WMASE score of 1.245 which is significantly larger than LightGBM at 0.210 and ES/BU at 0.410.

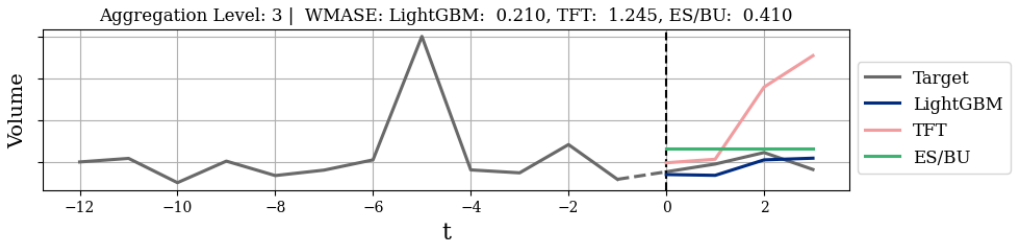


Figure 6.3: Forecasts from TFT_{opt}, LightGBM_{opt} and ES/BU on one of the time series at aggregation level 3. The dashed black vertical line marks the forecast origin. The dashed gray line is the connection between the training data and test data. TFT_{opt} demonstrates an unreliable forecast.

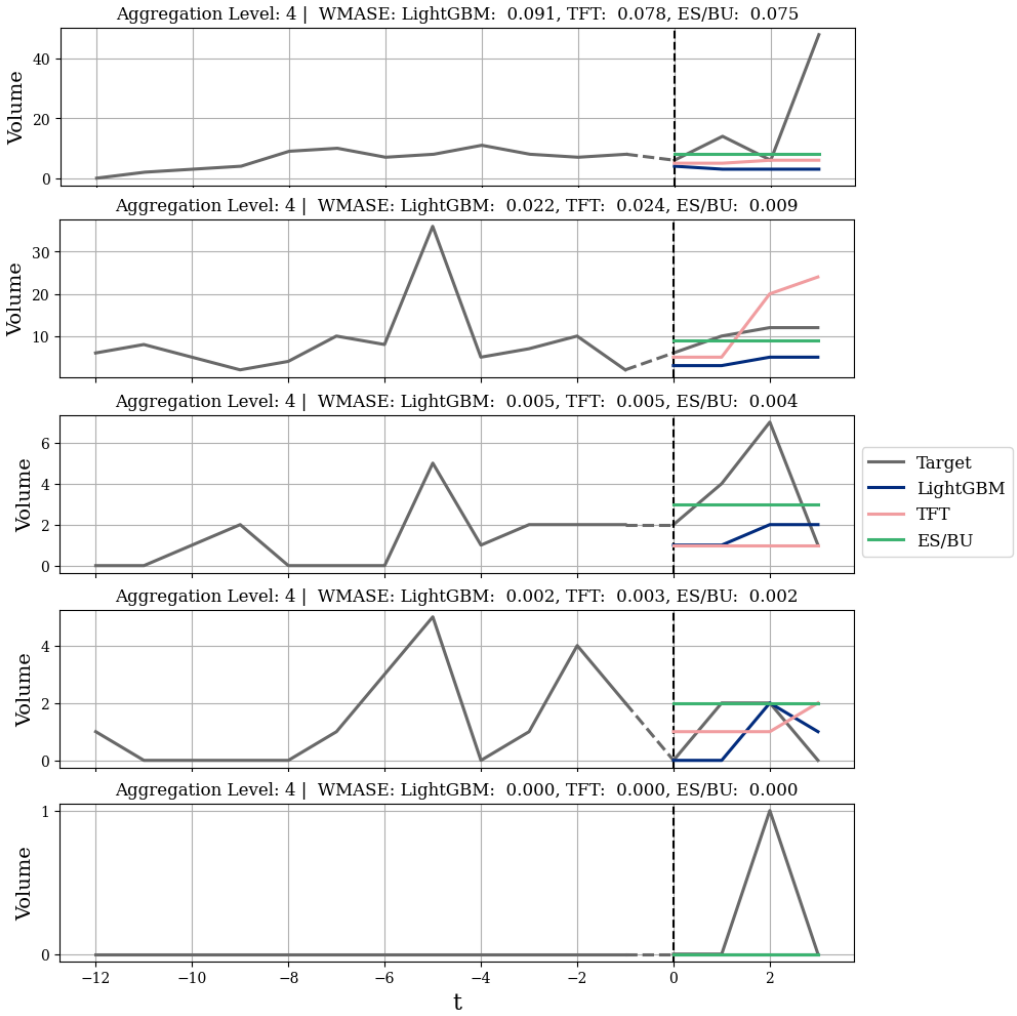


Figure 6.4: Forecasts from TFT_{opt}, LightGBM_{opt} and ES/BU on five bottom-level time series. The dashed black vertical line marks the forecast origin. The dashed gray line is the connection between the training data and test data.

Figure 6.4 contains a set of five different time series at aggregation level 4, which are sales of individual products. The figure illustrates that the patterns of the bottom-level time series are random and that all models struggle to capture meaningful temporal dependencies.

6.2 Averaged and Reconciled Forecasts

Table 6.1 shows that the performance of the TFT model was improved when using the averaged forecasts TFT_{μ} compared to the single forecast TFT_{opt}, and that the performance of TFT_{μ} was further improved by applying the Minimum trace WLS_s reconciliation. Figure 6.5 illustrates the same time series as Figure 6.3, where the TFT_{opt} demonstrated inaccurate performance, with the TFT_{μ} and TFT_{μ}/Mint WLS_s forecasts to visualize the possible effects of averaging and reconciling the base forecasts.

The first graph in Figure 6.5 shows that using the averaged TFT_{μ} model reduced the forecast error from 1.245 to 0.374. The second graph shows that applying the MinT WLS_s reconciliation to the base TFT_{opt} forecasts had a similar effect, reducing the forecast error to 0.647. The final graph in Figure 6.5 demonstrates the combined effect of using the averaged TFT_{μ} model and applying MinT WLS_s reconciliation, reducing the forecast error from 1.245 to 0.208.

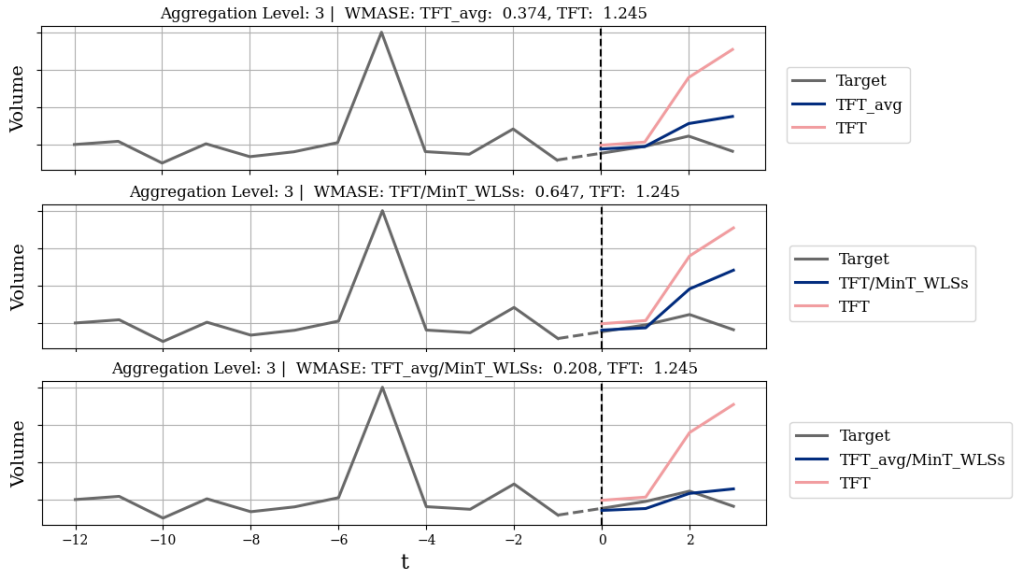


Figure 6.5: Visualized effects of using the ensemble model and reconciliation for the TFT forecast.

Figure 6.6 show the same time series and adaptions to the LightGBM as is shown for the TFT in Figure 6.5. Using the averaged LightGBM_{μ} model resulted in an increased error from 0.210 to 0.254, applying the MinT WLS_s increased the error from 0.210 to 0.270, and the combined model LightGBM_{μ}/MinT WLS_s had the highest forecast error at 0.290.

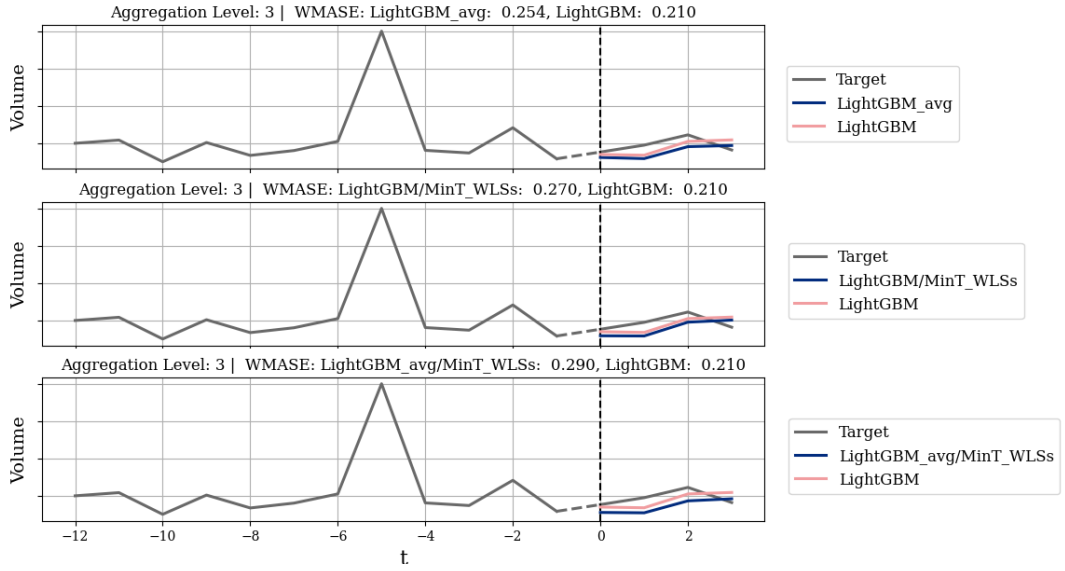


Figure 6.6: Visualized effects of using the ensemble model and reconciliation for the LightGBM forecast.

6.3 Treating the Time Series as Hierarchical

In Table 6.3, the WMASE for TFT_{opt} and TFT_{μ} models trained on the entire hierarchy are compared to TFT_{opt} and TFT_{μ} models trained level by level. The results show that the TFT_{μ} outperformed both level-wise models on all aggregation levels except level 3. However, the TFT_{μ}/MinT WLS_s outperformed both level-wise models on all aggregation levels. The level-wise TFT's exhibit significantly poor performance on aggregation level 1, where the amount of available data is the smallest.

Model	Aggregation Level				Average
	1	2	3	4	
TFT_{μ} /MinT WLS_s	**0.231**	0.340	**0.262**	**0.540**	**0.343**
TFT_{μ}	0.254	**0.312**	0.424	0.544	0.383
TFT_{opt} (Level-by-level)	0.870	0.351	0.358	0.636	0.554
TFT_{opt}/MinT WLS_s	0.580	0.628	0.720	0.809	0.684
TFT_{opt}	0.419	0.414	1.331	0.759	0.731
TFT_{μ} (Level-by-level)	1.516	0.345	0.401	0.675	0.734

Table 6.3: Comparison of WMASE for TFT models trained on the entire hierarchy and models trained level-by-level. Column-wise best values are marked with bold text.

Table 6.4 shows the WMASE for the LightGBM models trained on the entire hierarchy compared to those optimized level by level. The results show that both level-wise LightGBM models outperformed the full-hierarchy models on aggregation

levels 1 and 2, indicating that there is no gain in treating the data as hierarchical time series if the goal is to forecast the top levels of the hierarchy. However, the level-wise LightGBM models exhibit the worst performance of all models on aggregation level 3 and 4.

Model	Aggregation Level				Average
	1	2	3	4	
LightGBM$_{opt}$	0.421	0.372	**0.276**	0.542	**0.403**
LightGBM$_{opt}$ (Level-by-level)	**0.205**	0.339	0.502	0.566	0.403
LightGBM$_{opt}$/MinT WLS$_s$	0.313	0.394	0.357	0.552	0.404
LightGBM$_{\mu}$/MinT WLS$_s$	0.314	0.407	0.386	**0.530**	0.409
LightGBM$_{\mu}$	0.337	0.441	0.324	0.540	0.410
LightGBM$_{\mu}$ (Level-by-level)	0.220	**0.319**	0.460	0.703	0.426

Table 6.4: Comparison of WMASE for LightGBM models trained on the entire hierarchy and models trained level-by-level. Column-wise best values are marked with bold text.

Furthermore, Table 6.4 show that when combining the *opt* models for each level, the average WMASE across all levels is similar to the average WMASE of the LightGBM$_{opt}$ trained on the entire hierarchy.

7

Discussion

The purpose of this thesis was to explore viable methods for demand forecasting, facing the challenge of limited data availability. To fulfill this purpose, one objective was to compare the predictive ability of the machine learning-based forecasting models TFT and LightGBM against an exponential smoothing state-space model. The results showed in Table 6.1 indicate that both the TFT and the LightGBM have significant potential in forecasting demand, despite limited data availability.

All except two of the LightGBM models implemented were able to achieve a significantly higher forecasting accuracy than the ES/BU benchmark model, suggesting that it is a reliable forecasting model. Conversely, the results for the TFT were not as convincing. Although the TFT_{μ} demonstrated superior performance to all other models, LightGBM included, the TFT_{opt} demonstrated inferior performance to all other models, raising concerns regarding its stability on small data sets.

Looking at Table 6.2, we see that the models TFT_{μ}, LightGBM_{opt} and LightGBM_{μ} all overestimated the total volume over the forecast horizon when forecasting the level 1 time series, but underestimated the total volume when aggregating the forecasts for all level 4 time series. A probable explanation for the latter observation is the intermittent characteristic that describes many of the time series in the fourth level.

For example, looking at the bottom time series in Figure 6.4, we see a time series of a product that has not been sold for at least the last 12 weeks, but has one unit sold in the test set. Such sudden spikes are extremely difficult to forecast. The models can learn during training that more often than not, it is better to predict zero to achieve higher accuracy on such intermittent time series.

The observed pattern of overestimating the total volume at the highest aggregation level and underestimating the total volume at the bottom level illuminates the importance of coherency in hierarchical time series forecasting.

A second objective in this thesis was to investigate the effects of hierarchical reconciliation. The results in Table 6.1 show that there is no unambiguous answer to this question. Expanding on the observation that the TFT_{μ}, LightGBM_{opt} and LightGBM_{μ} underestimated the total volume at the bottom level, we see that the

Bottom-Up reconciled forecasts resulted in worse performance than the base forecasts in all three cases. Similarly, the MinT OLS resulted in lower accuracy than the base forecasts in all instances, but the average accuracy was not as severely affected. However, the results indicate that the MinT WLS_s reconciliation method has the potential to increase accuracy.

Although applying reconciliation methods did not show significant signs of improving the base forecasts in terms of WMASE, the significance of reconciliation is demonstrated in Table 6.2. The effect of applying MinT WLS_s reconciliation improved the PSEL for the bottom and top level for the TFT_{μ}, LightGBM_{opt} and LightGBM_{μ} without significantly compromising the PSEL on level 2 and 3. For the TFT_{opt} model, the MinT WLS_s improved the average forecast accuracy, but the PSEL was higher for the bottom and top level.

7.1 Limitations of the Results

One inherent limitation of the analysis conducted in this thesis is that all results were derived from a test set comprising observations from only $h = 4$ time steps. However, when calculating the WMASE across the entire hierarchy, there are 583 observations per time step which enhances the stability of the results, as it increases the likelihood of identifying weaknesses in models with poor generalization ability. For example, the TFT_{opt} displayed better forecasting accuracy on time series in level 1 and 2 compared to the ES/BU benchmark, but at the third level its accuracy was significantly worse. An evident sign that the TFT_{opt} model had poor generalization ability is displayed in Figure 6.3. Thus, fitting the models across all time series simultaneously can be beneficial not only for training, but also evaluation.

The lack of test data is still a problem, since the specific data points included in the test set may not be representative of the actual data distribution, resulting in a biased performance evaluation. As mentioned in Section 2.4, biased evaluations are accepted if the goal is to compare the relative accuracy between models, but not if the goal is to evaluate the generalization ability of a particular forecasting model. Thus, the WMASE scores shown in Table 6.1 are useful for comparing the forecasting models, but they might not be reliable representations of the generalization ability of the models. Consequently, I will exercise caution when addressing the results of this thesis and their generalizability.

Furthermore, the WMASE of the Naïve forecasting model indicates that the data in the test set was less volatile than data in the training set. When using the Naïve forecasting method, the MASE is essentially the ratio of the h-step multi-horizon naïve forecast and the one-step in-sample naïve forecast

$$\text{MASE} = \frac{\frac{1}{h}\sum_{t=n+1}^{n+h}|y_t - \hat{y}_t|}{\frac{1}{n-1}\sum_{t=2}^{n}|y_t - y_{t-1}|},$$

where $\hat{y}_t = y_n \ \forall t \in [n+1, n+h]$. A reported average WMASE of 0.501 implies that the Naïve method performed better on the test set than the training set, despite

comparing an h-step multi-horizon forecast with a one-step forecast. This suggests that the data in the test set was less turbulent, or volatile, than in the training set. Consequently, it is likely that the reported WMASE for all other models are optimistically biased.

7.2 ML-based vs. Traditional Forecasting Models

Addressing the viability of ML-based forecasting models compared to traditional ones in the scope of small-scale demand forecasting involves a three-way trade-off between accuracy, stability, and complexity.

Achieving maximum accuracy is of central importance as it directly contributes to reducing stock keeping or minimizing missed opportunities, thereby mitigating costs for The Company. The results presented in Table 6.1 indicate that the ensemble model TFT_{μ} produces the most accurate forecasts. However, the TFT_{opt} model produces the least accurate forecasts. Conversely, the LightGBM_{μ} and LightGBM_{opt} demonstrated similar performance.

Stable forecasts are important for reliable and consistent decision-making. Looking at the TFT results in Table 6.3, the standard deviation of the average WMASE is $\sigma_{TFT} = 0.159$, compared to the LightGBM results in Table 6.4 where the standard deviation is $\sigma_{LightGBM} = 0.008$. This observation suggests that the LightGBM is a more stable forecast model than the TFT, as it is less sensitive to implementation specifics. Furthermore, the level-by-level optimized TFT models exhibited poor performance on aggregation level 1 where data is most limited, whereas the LightGBM exhibited superior performance on level 1, indicating that the LightGBM is more stable than the TFT on very small amounts of data.

Moreover, several aspects pertaining to complexity must be considered. The most prominent aspect being the complexity related to model selection. Stable and data efficient model selection methods are especially important when the amount of available data is limited, due to the inherent trade-off of data utilization in training versus testing. The three-way holdout cross-validation method that was initially used proved infeasible due to unstable forecasting results. To improve stability, the simple solution would have been to increase the size of the validation and test set. However, given that the data set consists of observations from 107 time steps, using a validation and test set consisting of 4 time steps each already allocates approximately 7.5 % of the data, that becomes unavailable for training. Increasing the size of the validation and test set decreases the amount of data available for training, which can introduce a pessimistic bias in the accuracy evaluation of the models.

The model selection part of this thesis was an iterative process that ultimately warranted setting some hyperparameters to fixed values to attain more stable results. The bagging fraction and feature fraction in the LightGBM had to be set to 1.0, effectively disabling bagging and feature selection, to reduce randomness in the model selection and evaluation process. Although bagging and feature selection

are proven methods for reducing overfitting, initial experiments showed that the accuracy of the LightGBM was too sensitive to the randomness attributed to bagging and feature selection, most likely due to limited data availability. Furthermore, the learning rate had to be kept at a low value for both the TFT and the LightGBM for more stable convergence.

Conversely, the ES/BU forecast model based on the state-space framework is an "off-the-shelf" solution. It does not require any scaling or feature engineering, and it does not require any hyperparameter tuning, since model selection is integrated in the framework. Moreover, the model selection in the state-space framework involves selecting the model with the minimum Akaike's Information Criterion (AIC) score, punishing complex models that are more prone to overfitting.

A second aspect of complexity is the complexity pertaining to the forecasting models themselves. The TFT and LightGBM are more complex than the ES state-space model. The TFT is remarkably complex, it involves several layers of non-linear processing, and if the hyperparameter search space is not restricted, the number of trainable parameters can quickly grow out of proportion with regards to the amount of available data. The LightGBM is a simpler model, and the fundamental principle of gradient boosting is intuitively easier to understand than the deep-learning algorithm underlying the TFT. Furthermore, the LightGBM and ES/BU had significantly lower execution times than the TFT.

7.3 Hierarchical Time Series and Reconciliation

Two aspects of hierarchical forecasting were investigated in this thesis. Firstly, the effects of applying hierarchical reconciliation methods were examined. The second aspect relates to the investigation of whether the TFT and LightGBM can extract additional information by structuring the time series hierarchically, and ultimately produce more accurate forecasts.

Out of the three reconciliation methods used, the Bottom-Up method demonstrated the worst performance. A common pattern among the well-performing forecasting algorithms was that they underestimated the total sales volume when aggregated across all bottom level time series. Thus, this pessimistic bias was propagated through the hierarchy, as showed in Table 6.2, resulting in poor accuracy. It is a known problem that the BU approach suffers from information loss, using only the information from the bottom level time series. Furthermore, the bottom level time series often have the lowest signal-to-noise ratio, making accurate forecasting difficult. This observation aligns with the results obtained in this thesis. Looking at the average WMASE per level displayed in Table 6.1, it was highest at aggregation level 4, and lowest at aggregation level 1. The BU method did however improve the average WMASE for the TFT_{opt} model, but that slight improvement can be attributed to correcting the poor accuracy of the TFT_{opt} on the third aggregation level, as the accuracy of the TFT_{opt}/BU model was lower on levels 1 and 2.

Conversely, The MinT WLS_s reconciliation method demonstrated high potential. Whereas applying the MinT WLS_s did not have a significant effect on the average WMASE for the LightGBM, it did positively affect the PSEL. Furthermore, it is in The Company's interest that levels 1 and 4 are coherent. The level 1 forecast is important for The Company for deciding the capacity utilization in their factories, and the level 4 forecasts are important for resource allocation. Correcting the incoherency without negatively affecting the average WMASE is therefore a significant improvement of the quality of the forecasts.

It was expected that the MinT WLS_s would give better results than the OLS approach. The time series hierarchy used in this thesis was flat, with 1, 4, 3, 575 time series in levels 1 through 4. The MinT OLS method weighs the errors of all forecasts equally, meaning that the cumulative weight of the bottom level time series is significantly higher than the upper level time series. Given that the forecasting models on average exhibited worse performance on the bottom level time series, it would be better to increase the weight for the upper level time series which is exactly what the The MinT WLS_s does. The MinT WLS_s puts a higher weight on correcting the errors of the top level time series during reconciliation, to some extent anchoring the forecasts of the bottom level time series to the upper ones.

The most notable affect of applying MinT WLS_s is showed in Figure 6.5. The base forecast of the TFT_{opt} is way too optimistic, potentially due to overfitting. Interestingly, that specific forecast was the only observed instance where the TFT_{opt} model exhibited such poor performance. Thus, the MinT WLS_s reconciliation was able to correct this overestimation. The TFT ensemble forecast produced with the TFT_{μ} model has a similar effect on the TFT_{opt} forecast, and the combined effect of using the ensemble forecast and MinT WLS_s reconciliation demonstrated promising results, reducing the WMASE from 1.245 to 0.208.

An interesting thought that arises from this observation is the potential in using hierarchical reconciliation for error correction in time series forecasting. The MinT WLS_s did not significantly affect the forecast accuracy of the $LightGBM_{opt}$ or $LightGBM_{\mu}$, but no severe misprediction was observed for either of those models. In contrast, the average WMASE of the TFT_{μ} forecast was improved by 10.4 %, with the most significant improvement observed at level 3 at 38.2 %, where there was great room for improvement in the TFT_{μ} forecast.

The hypothesis that laid ground for investigating the potential gain of structuring time series hierarchically even though only a single hierarchical level is of interest, was that it can serve as a form of data augmentation, as well as enabling the use of reconciliation. The results for the TFT showed in Table 6.3 suggests that there could be gain in structuring the time series hierarchically, as the TFT_{μ} and TFT_{μ}/MinT WLS_s outperformed the level-wise optimized TFT models on 3/4 respectively 4/4 aggregation levels. The LightGBM exhibited similar results for aggregation level 3 and 4. Thus, these results collectively suggests that the forecast accuracy on the bottom level time series is enhanced by structuring the time series hierarchically, with and without reconciliation.

8

Conclusions

This thesis was an exploratory study, investigating ways to construct accurate forecasting algorithms for small-scale demand forecasting. Concerning the comparison of the TFT and LightGBM forecasting models to the ES state-space model, the findings of this thesis indicate that both models are capable of outperforming the ES/BU benchmark. The main concern when stating a conclusion regarding this comparison is the stability of the models.

Although the best performing model was based on the TFT, so was the worst performing model. Conversely, the LightGBM exhibited more reliable results. Moreover, the gradient boosting framework is more intuitive than deep learning, and the execution time of the LightGBM was significantly faster.

Conclusively, whereas both the TFT and LightGBM demonstrate great potential, I would recommend that the LightGBM be considered a more viable choice for The Company for forecasting the demand of their products. The amount of data available seems insufficient for reliably training the TFT.

Regarding structuring the time series as hierarchical, and using reconciliation methods, the results of this thesis suggest that the Minimum Trace (MinT) WLS_s method is a viable choice for reconciling the forecasts. Correcting the incoherency between aggregation levels 1 and 4 without compromising the overall forecasting accuracy is highly valuable for The Company.

Furthermore, the results in this thesis show evidence that the forecasting accuracy for the bottom level time series is enhanced when structuring the time series hierarchically and training the models across the entire hierarchy, rather than level by level. For the TFT, this was true for all aggregation levels.

A peripheral yet important aspect illuminated in this thesis is that the WMASE metric does not capture incoherency or accumulated volume error, two dimensions that are of importance in demand forecasting. The Percentage Sum of Errors per Level (PSEL) metric was suggested to account for those dimensions of the results. However, the PSEL metric would not have made sense if the products had been heterogeneous, since it would not make sense to accumulate the sales volume of products that are completely different.

8.1 Future work

Two interesting findings from the analysis performed in this thesis that warrant further investigation are (1) the possibility to use reconciliation as a mechanism for correcting severe mispredictions, and (2) creating a hierarchical/grouped structure of time series as a form of data augmentation.

Another interesting subject for future research that would benefit the domain of hierarchical forecasting would be to establish an accuracy metric that can punish incoherence.

Bibliography

[1] S. Makridakis, R. Hogarth, and A. Gaba, "Forecasting and uncertainty in the economic and business world," *International Journal of Forecasting*, vol. 25, pp. 794–812, Oct. 2009. DOI: 10.1016/j.ijforecast.2009.05.012.

[2] M. Anderer and F. Li, "Hierarchical forecasting with a top-down alignment of independent-level forecasts," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1405–1414, Oct. 2022. DOI: 10.1016/j.ijforecast.2021.12.015.

[3] M. Abolghasemi, R. J. Hyndman, G. Tarr, and C. Bergmeir, *Machine learning applications in time series hierarchical forecasting*, 2019. DOI: 10.48550/ARXIV.1912.00370.

[4] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1346–1364, 2022. DOI: 10.1016/j.ijforecast.2021.11.013.

[5] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, English, 3rd. Australia: OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/.

[6] B. Paria, R. Sen, A. Ahmed, and A. Das, *Hierarchically regularized deep forecasting*, 2021. DOI: 10.48550/ARXIV.2106.07630.

[7] C. Fry and M. Brundage, "The m4 forecasting competition – a practitioner's view," *International Journal of Forecasting*, vol. 36, no. 1, pp. 156–160, 2020. DOI: 10.1016/j.ijforecast.2019.02.013.

[8] T. JANUSCHOWSKI, J. GASTHAUS, W. YUYANG, S. S. RANGAPURAM, and L. CALLOT, "Deep learning for forecasting: Current trends and challenges.," no. 51, pp. 42–47, 2018, ISSN: 15559068.

[9] T. Januschowski, J. Gasthaus, Y. Wang, *et al.*, "Criteria for classifying forecasting methods," *International Journal of Forecasting*, vol. 36, no. 1, pp. 167–177, 2020. DOI: 10.1016/j.ijforecast.2019.05.008.

[10] A.-A. Semenoglou, E. Spiliotis, S. Makridakis, and V. Assimakopoulos, "Investigating the accuracy of cross-learning time series forecasting methods,"

International Journal of Forecasting, vol. 37, no. 3, pp. 1072–1084, 2021. DOI: 10.1016/j.ijforecast.2020.11.009.

[11] S. L. Wickramasuriya, G. Athanasopoulos, and R. J. Hyndman, "Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization," *Journal of the American Statistical Association*, vol. 114, no. 526, pp. 804–819, 2019. DOI: 10.1080/01621459.2018.1448825.

[12] G. Athanasopoulos, R. Hyndman, R. Ahmed, and H. L. Shang, "Optimal combination forecasts for hierarchical," *Computational Statistics & Data Analysis*, vol. 55, pp. 2579–2589, Sep. 2011. DOI: 10.1016/j.csda.2011.03.006.

[13] S. B. Taieb, J. Yu, M. Barreto, and R. Rajagopal, "Regularization in hierarchical time series forecasting with application to electricity smart meter data," *AAAI*, vol. 31, no. 1, Feb. 2017. DOI: 10.1609/aaai.v31i1.11167.

[14] E. Paton, *H&m, a fashion giant, has a problem: $4.3 billion in unsold clothes*, Mar. 2018. [Online]. Available: https://www.nytimes.com/2018/03/27/business/hm-clothes-stock-sales.html.

[15] T. Hogg and B. Huberman, "Avoiding moral hazards in organizational forecasting.," *Proceedings. Third International Symposium on Electronic Commerce, Electronic Commerce, 2002. Proceedings. Third International Symposium on, Electronic commerce*, pp. 22–29, 2002, ISSN: 0-7695-1861-3. DOI: 10.1109/ISEC.2002.1166907.

[16] U. Hook, *"blame the algorithm" is the new "don't blame me. i just work here."* Medium, blog, Accessed Jan 31, 2023, Jan. 2019. [Online]. Available: https://medium.com/@uwehook/blame-the-algorithm-is-the-new-dont-blame-me-i-just-work-here-fb9bc6107931.

[17] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, "Machine learning strategies for time series forecasting," in 2013, vol. 138. DOI: 10.1007/978-3-642-36318-4_3.

[18] B. Lim and S. Zohren, "Time-series forecasting with deep learning: A survey," *Phil. Trans. R. Soc. A*, vol. 379, no. 2194, 2021. DOI: 10.1098/rsta.2020.0209.

[19] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, *Temporal fusion transformers for interpretable multi-horizon time series forecasting*, 2019. DOI: 10.48550/ARXIV.1912.09363.

[20] B. Mehlig, *Machine Learning with Neural Networks*. Cambridge University Press, Oct. 2021. DOI: 10.1017/9781108860604.

[21] C. Fan, Y. Zhang, Y. Pan, *et al.*, "Multi-horizon time series forecasting with temporal attention learning," in *Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery & data mining*, 2019, pp. 2527–2535.

[22] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[23] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[24] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka, *A multi-horizon quantile recurrent forecaster*, 2018. arXiv: 1711.11053 [stat.ML].

[25] Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Aït-Bachir, and V. Strijov, "Position-based content attention for time series forecasting with sequence-to-sequence rnns," in *Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V 24*, Springer, 2017, pp. 533–544.

[26] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[27] G. Ke, Q. Meng, T. Finley, *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17, Curran Associates Inc., 2017, pp. 3149–3157.

[28] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002. DOI: 10.1016/S0167-9473(01)00065-2.

[29] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, pp. 123–140, 1996. DOI: 10.1007/BF00058655.

[30] S. Raschka, *Model evaluation, model selection, and algorithm selection in machine learning*, 2020. arXiv: 1811.12808 [cs.LG].

[31] A. D. Lainder and R. D. Wolfinger, "Forecasting with gradient boosted trees: Augmentation, tuning, and cross-validation strategies: Winning solution to the m5 uncertainty competition," *International Journal of Forecasting*, vol. 38, no. 4, pp. 1426–1433, 2022. DOI: 10.1016/j.ijforecast.2021.12.003.

[32] J. M. B. Christoph Bergmeir, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, pp. 192–213, 2012, ISSN: 0020-0255. DOI: 10.1016/j.ins.2011.12.028.

[33] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006. DOI: 10.1016/j.ijforecast.2006.03.001.

[34] S. L. Wickramasuriya, B. A. Turlach, and R. J. Hyndman, "Optimal non-negative forecast reconciliation," *Statistics & Computing*, vol. 30, no. 5, pp. 1167–1182, Apr. 2020. DOI: 10.1007/s11222-020-09930-0.

[35] G. Athanasopoulos, R. J. Hyndman, N. Kourentzes, and F. Petropoulos, "Forecasting with temporal hierarchies," *European Journal of Operational Research*, vol. 262, no. 1, pp. 60–74, 2017. DOI: 10.1016/j.ejor.2017.02.046.

[36] S. Elsayed, D. Thyssens, A. Rashed, H. S. Jomaa, and L. Schmidt-Thieme, *Do we really need deep learning models for time series forecasting?* 2021. arXiv: 2101.02118.

[37] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting*, vol. 18, no. 3, pp. 439–454, 2002. DOI: 10.1016/S0169-2070(01)00110-8.

[38] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[39] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, *Fast and accurate deep network learning by exponential linear units (elus)*, 2016. arXiv: 1511.07289 [cs.LG].

[40] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, *Language modeling with gated convolutional networks*, 2017. arXiv: 1612.08083 [cs.CL].

[41] J. Beitner, *Pytorch-Forecasting*, https://github.com/jdb78/pytorch-forecasting, Accessed: May 31, 2023, 2023.

[42] Microsoft, *LightGBM*, https://github.com/microsoft/LightGBM, Accessed: May 31, 2023, 2023.

[43] R. J. Hyndman, *Forecast*, https://github.com/robjhyndman/forecast, Accessed: May 31, 2023, 2023.

[44] Nixtla, *Hierarchicalforecast*, https://github.com/nixtla/hierarchicalforecast, Accessed: May 31, 2023, 2023.

[45] Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi, "Multiobjective tree-structured parzen estimator for computationally expensive optimization problems," in *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, Association for Computing Machinery, 2020, pp. 533–541. DOI: 10.1145/3377930.3389817. [Online]. Available: doi.org/10.1145/3377930.3389817.

A

Training Details

The TFT was trained using the ADAM optimizer. The weight initialization used was the default implementation in the `PyTorch_Forecasting` library, using a mix of Xavier uniform initialization, zero initialization, and Kaiming normal initialization. The training was done on batches of size 32, with max encoder length (look back window) of 52 and max prediction length (forecast horizon) of 4. In the training phase, a validation loss patience of 5 epochs was used with a minimum decrease of $\delta = 10^{-3}$ as an early stopping condition. A maximum of 50 epochs was set, but never invoked. Training the TFT model took on average approximately 15 minutes. Table A.1 contains a detailed description of the size of each layer in the TFT

No.	Name	Type	Params
0	loss	MAE	0
1	logging_metrics	ModuleList	0
2	input_embeddings	MultiEmbedding	877
3	prescalers	ModuleDict	128
4	static_variable_selection	VariableSelectionNetwork	3.1 K
5	encoder_variable_selection	VariableSelectionNetwork	3.4 K
6	decoder_variable_selection	VariableSelectionNetwork	1.9 K
7	static_context_variable_selection	GatedResidualNetwork	1.4 K
8	static_context_initial_hidden_lstm	GatedResidualNetwork	1.4 K
9	static_context_initial_cell_lstm	GatedResidualNetwork	1.4 K
10	static_context_enrichment	GatedResidualNetwork	1.4 K
11	lstm_encoder	LSTM	2.7 K
12	lstm_decoder	LSTM	2.7 K
13	post_lstm_gate_encoder	GatedLinearUnit	684
14	post_lstm_add_norm_encoder	AddNorm	36
15	static_enrichment	GatedResidualNetwork	1.7 K
16	multihead_attn	InterpretableMultiHeadAttention	1.4 K
17	post_attn_gate_norm	GateAddNorm	720
18	pos_wise_ff	GatedResidualNetwork	1.4 K
19	pre_output_gate_norm	GateAddNorm	720
20	output_layer	Linear	19
21	Sum	-	27.1 K

Table A.1: Number of trainable parameters at each layer in the TFT

The LightGBM models were trained using a maximum of 1000 iterations, with a val-

idation loss patience of 100 iterations used as an early stopping. The early stopping condition was invoked almost every time, indicating that convergence was reached before 1000 iterations. Training the LightGBM models took on average approximately 20 seconds. Since the LightGBM is based on gradient boosted decision trees, its size is not easily described. Each training iteration involves fitting a new weak learner, meaning that if 1000 iterations are used, the LightGBM consists of 1000 weak learners. However, the size of each weak learner can vary, and is decided by the *num_leaves* and *min_child_samples* parameters.

Both the TFT and LightGBM were optimized using the Mean Absolute Error (MAE) as loss function.

The ES/BU model took 25 seconds to fit. The ES/BU model was initially fit to the M5 data set to ensure that I arrived at the same results as was obtained in the competition.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY