Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Eivind Strøm

Evaluation of Multi-step Forecasting Models

An Empirical Deep Learning Study

Master's thesis in Computer Science
Supervisor: Odd Erik Gundersen
August 2021

NTNU
Norwegian University of
Science and Technology

Eivind Strøm

Evaluation of Multi-step Forecasting Models

An Empirical Deep Learning Study

Master's thesis in Computer Science
Supervisor: Odd Erik Gundersen
August 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Preface

This thesis was produced as part of achieving the degree Master of Science at the Norwegian University of Science and Technology, NTNU, in Trondheim, Norway. The field of specialization is Artificial Intelligence at Department of Computer Science. The thesis is independent work by Eivind Strøm and is motivated by professional and academic interests in the fields of deep learning and time series forecasting.

I want to sincerely thank my supervisor, Adjunct Associate Professor Odd Erik Gundersen for deeply valuable discussions and guidance during the semester. Lastly, I would like to thank my friends and family for their continuous support in this period.

Eivind Strøm

Trondheim, 02 August, 2021

Sammendrag

Denne masteroppgaven omhandler evaluering av metoder for flerperiodeprediksjon av tidsserier ved å gjennomføre en empirisk studie med dype læringsmodeller. I dag blir modeller for flerperiodeprediksjon evaluert ved at en bruker en eller flere evalueringsmetrikker og aggregerer resultatene for å få ett enkelt tall på hvor god en modell er. Denne metoden kan føre til at informasjon som er viktig for modellutviklere og industri faller bort i aggregeringsprosedyren. For å gi bedre informasjon til utviklere som skal evaluere modeller foreslår vi fire nye evalueringsmetrikker: 1) en metrikk som rangerer modeller basert på antall serier modellen er best og dårligst på i et datasett. 2) en variansvektet metrikk som hensyntar forskjeller i varians over sesongperioder. 3) en delta-horisont metrikk som måler hvor mye hver modell endrer på sin prognose over prediksjonsperioden. 4) en dekomponert metrikk som relaterer evalueringen av prediksjoner til tidsseriekonseptene trend, sesong, støy og tid. For å vise hvordan de foreslåtte metrikkene kan anvendes implementerer vi fire dype læringsmodeller og gjennomfører eksperimenter på fem datasett. Resultatene viser hvordan den nåværende metoden for å evaluere prediksjoner via aggregering skjuler viktig informasjon, og vi viser viktigheten av å hensynta sesongvariasjoner samt evaluering per tidsserie. Vi viser også hvordan de foreslåtte metrikkene kan brukes i flere sammenhenger, og diskuterer anvendbarheten av metrikkene i lys av de empiriske resultatene.

Abstract

This thesis addresses the evaluation of multi-step point forecasting models by conducting an empirical deep learning study. Currently, deep learning models for multi-step forecasting are evaluated on datasets by selecting one or several error metrics and aggregating errors across the time series and the forecast horizon. This approach hides insights that would otherwise be useful for practitioners and industry when evaluating and selecting the best forecasting models. We propose four novel metrics to provide additional insights when evaluating models: 1) a win-loss ranking metric that shows how models perform across time series in the dataset. 2) a variance weighted metric that accounts for differences in variance across the seasonal period. 3) a delta horizon metric measuring how much models update their estimates over the forecast horizon. 4) decomposed errors that relate the forecasting error to trend, seasonality, and noise. To show the applicability of the proposed metrics, we implement four recent deep learning architectures and conduct experiments on five benchmark datasets. Our results show how the current approach of aggregating metrics neglects valuable information and we show the importance of considering seasonality and errors on individual time series. Lastly, we highlight several use cases for the proposed metrics and discuss the applicability in light of the empirical results.

Keywords: Multi-step forecasting, Performance metrics, Empirical study, Deep learning, AI, DeepAR, TFT

Table of Contents

List of Tables

List of Figures

Acronyms

AIC Akaike Information Criterion.

AR AutoRegressive.

ARIMA AutoRegressive Integrated Moving Average.

ASE Absolute Scaled Error.

GRN Gated Residual Network.

LSTM Long Short Term Memory.

MA Moving Average.

MAE Mean Absolute Error.

MASE Mean Absolute Scaled Error.

MLE Maximum Likelihood Estimation.

MSE Mean Squared Error.

RMSE Root Mean Squared Error.

RMSSE Root Mean Squared Scaled Error.

RNN Recurrent Neural Network.

SARIMA Seasonal AutoRegressive Integrated Moving Average.

Seq2Seq Sequence-to-Sequence.

sMAPE Symmetric Mean Absolute Percentage Error.

SSE Squared Scaled Error.

SSM State Space Model.

STL Seasonal-Trend decomposition based on Loess.

TCN Temporal Convolutional Network.

TDI Temporal Distortion Index.

TDM Temporal Distortion Mix.

TFT Temporal Fusion Transformer.

VW Variance Weighted.

1 | Introduction

Time series forecasting has been a prominent research field since the early 1980s, when the *Journal of Forecasting* and *International Journal of Forecasting* were founded. Between 1982 and 2005, over 940 papers were published, a summary of which is given by De Gooijer and Hyndman [1]. The modeling and forecasting of time series have been a key part of academic research due to its many important real-world applications, including: forecasting wind and solar power generation [2, 3], traffic [4], demand forecasting[5], trend analysis of climate change [6], data-driven medicine applications [7] and the forecasting of financial indices [8]. In most applications, it is important to produce forecasts for several time points in the future to allow decision making based on predicted trends. This is known as multi-step forecasting and has historically, despite its many applications, been less studied compared to one-step forecasts [9]. Nonetheless, multi-step forecasting remains a critical part of several real-world applications and has begun to see an increase in attention, especially in the deep learning community [10].

A major contributor to advances in the field of time series forecasting has been large scale empirical studies, designed to empirically evaluate methods and comparing newly proposed models to the state-of-the-art [11]. However, despite having a long history, the problem of objectively evaluating the results of such studies remains an issue in the field. For example, the results of the M3 competition have been revisited and discussed on several occasions [12, 13, 14, 15]. Fildes *et al.* [16] established early on that the ranking of the performance of various methods will vary depending on the performance metric used in the evaluation. Thus, research on developing robust and widely applicable metrics has been a prominent subject in the field [17]. Yet, Makridakis *et al.* [15] most recently raised concerns for the need of objective and unbiased ways to compare and test the performance of forecasting methods. Hence, the evaluation and testing of model performance is still a problem that poses the need for further research.

Despite academicians not agreeing on a single best performance metric (if such a metric even exists, see e.g., [18]), several empirical studies have been conducted and evaluated by different performance metrics [19]. We find that most empirical studies on multi-step forecasting proceed by evaluating models on several time series or datasets and aggregate the errors across the time series and the forecast horizon using some performance metric. This approach of aggregating the results allows researchers to draw general recommendations and conclusions. However, general recommendations are of limited use during the actual evaluation to be performed by a practitioner in an industry setting. Important questions such as: "Does the model perform well on all datasets? Will a combination of models perform better than a single model?" and "How does the model perform in terms of the seasonal period? Does the model frequently update its estimates across the forecast horizon or are the forecasts relatively stable?" Such questions would be useful to answer in the evaluation and

selection of models for industry applications, and we hypothesize that the aggregation of metrics through averaging hides these insights that would otherwise be useful to the practitioner.

Our main research question, therefore, is: *Does evaluating forecasting models by using aggregated metrics neglect insights on model performance, and can new metrics provide novel insights to the practitioner that develops and evaluates models for industry applications?*

The purpose of this thesis is to improve the insights gained when evaluating multi-step point forecasting models. First, we propose four novel performance metrics to provide industry practitioners with additional insights when evaluating models: 1) a win-loss ranking metric that shows how models perform across time series in the dataset. 2) a variance weighted metric that accounts for differences in variance across the seasonal period. 3) a delta horizon metric that measures how much a model updates its forecasts during the forecast horizon. 4) decomposed errors that relate the forecasting error to trend, seasonality, and noise. Additionally, we employ a recently proposed metric for measuring error in terms of temporal alignment.

Second, we conduct an empirical deep learning study to show the applicability of the proposed metrics in different scenarios. For this purpose, we implement five forecasting models that have been prominent in the recent literature on multi-step forecasting with deep learning. We train the models on five well-researched benchmark datasets that exhibit distinct characteristics and are representative for real-world industry applications. The empirical study extends the current literature on evaluation of deep learning models. It is also, to our knowledge, one of few works that implement, train, and evaluate each model within the same framework.

Furthermore, we show why the use of aggregated metrics can hide aspects of the experimental results that should be important for practitioners and industry. We do this through an extensive evaluation of the experimental results, highlighting several use cases for the proposed metrics. Our results indicate that no single model is best on any dataset, and a combination of models is likely to perform better. We find significant differences in how stable forecasts are over the forecast horizon as measured by the delta horizon metric. Lastly, the results indicate that improved accuracy over the forecast horizon largely stems from improved prediction of the time series trend component.

The remainder of the thesis is structured as follows: Preliminary background for understanding and developing the methodology is presented in chapter 2. Chapter 3 presents a review of related work and our contributions to the literature. Chapter 4 presents the proposed evaluation metrics and the models that are applied in our experiments. The experimental setting is presented in chapter 5 and obtained results are evaluated and discussed in chapter 6. Lastly, chapter 7 concludes the thesis and suggests directions for future research.

2 | Background

This chapter provides background and preliminary theory necessary for developing the methodology of chapter 4. The first two sections introduce fundamental concepts of time series and forecasting, in addition to notation that will be used throughout this thesis. Section 2.3 describes commonly employed performance metrics that we will build upon to propose new metrics. Lastly, section 2.4 and section 2.5 introduce the underpinnings of state space models and deep learning architectures for forecasting, which we use in our empirical experiments.

2.1 Fundamental Components of Time Series

We begin by defining a univariate time series Y of length n as an ordered sequence of observations $Y = (y_1, y_2, \ldots, y_n)$, where $y_t \in \mathbb{R}$ is the observation at time t for $t \in \{0, 1, \ldots n\}$. A useful way of interpreting a univariate time series is to decompose it into three components by performing a *time series decomposition*. The additive decomposition model is written as

$$y_t = T_t + S_t + R_t, \tag{2.1}$$

and the multiplicative model as

$$y_t = T_t \times S_t \times R_t, \tag{2.2}$$

where y_t is the time series data point, T_t is the trend-cycle component of the series, S_t is the seasonal component and R_t is the remaining component, all observed at time t [20]. There are several ways to decompose a time series, including classical decomposition, X-11, SEATS and STL. We will focus on STL, which is short for Seasonal-Trend decomposition based on Loess, and use it to relate error metrics to trend, season, and noise in chapter 4. Decomposing a time series allows us to determine the characteristics of a time series and can be useful when interpreting its underlying drivers. Furthermore, the concept of trend, season and remainder components are fundamental to state space models, which we introduce later in this chapter.

Figure 2.1 shows the STL decomposition of an electricity consumption time series. We can see that the trend-cycle component, T_t, captures the general momentum and direction of the time series while ignoring seasonality and random fluctuations. In contrast, the seasonal component captures any seasonality within a defined seasonal period m, and the remainder captures leftover movement, commonly referred to as noise. STL is a robust and versatile filtering procedure for time series decomposition that allows the seasonal component to vary over time [21]. In general, when performing a decomposition, we have to specify a seasonal periodicity m for which the decomposition is to be performed, e.g., $m = 24$ for hourly measured traffic where each season is defined as a day. For further details on how the

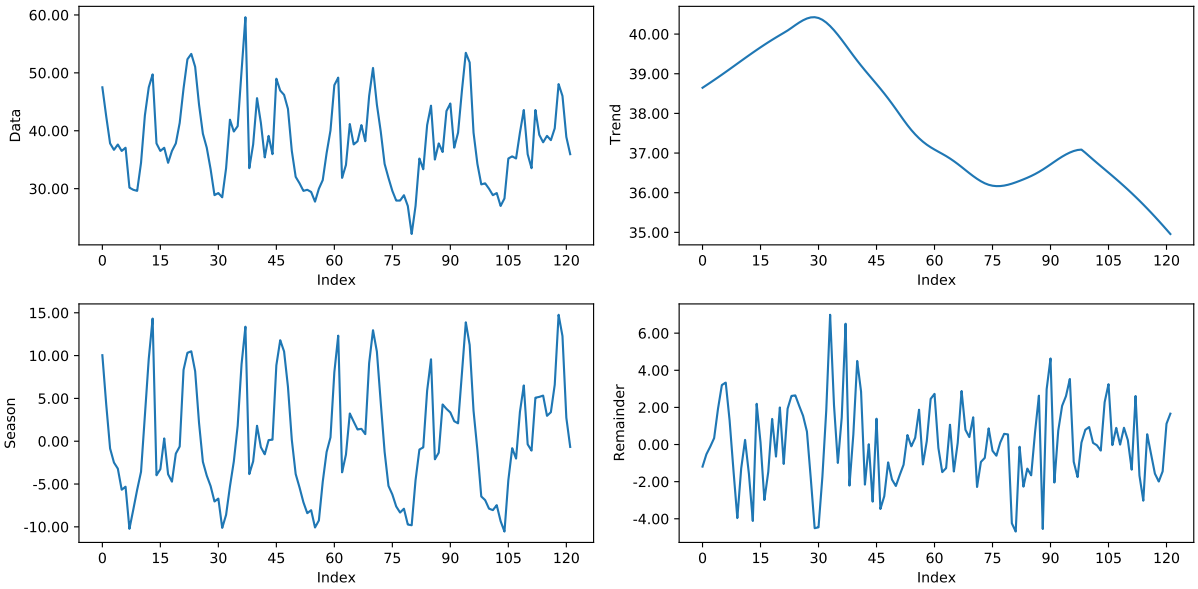


Figure 2.1: An example showcasing the STL decomposition of an electricity consumption time series

STL decomposition is computed, we refer to [20, 21]. In the next section, we define notation and methods for multi-step time series forecasting.

2.2 Time Series Forecasting

The general goal of time series forecasting is to predict the target $y_{i,t}$ at time t given an entity i and prior observations $y_{i,t-k}$, where k is the lookback window. The entity i, which also can be referred to as a group, represents a logical grouping of temporal information which we refer to as a single time series. This could be a sensor producing data points at regular intervals or tracking the electricity consumed by a client or household. When we collect several entities of the same type, i.e., several sensors measuring traffic, we refer to the set of entities as a time series dataset. Formalizing the above, we can express a one-step-ahead forecast as

$$\hat{y}_{i,t+1} = f(y_{i,t-k:t}, \mathbf{x}_{i,t-k:t}, \mathbf{s}_i), \tag{2.3}$$

where $\hat{y}_{i,t+1}$ is the forecasted value, $y_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$ are prior observations of the target, $\mathbf{x}_{i,t-k:t} = \{\mathbf{x}_{i,t-k}, \ldots, \mathbf{x}_{i,t}\}$ are any exogenous inputs, s_i is static metadata, e.g, sensor location, and $f(\cdot)$ is the prediction function learned by the model [10]. Although one-step-ahead forecasts are useful, several applications require predictions for multiple time steps into the future. Notable examples are the future prediction of power consumption for power trading, wind and solar power generation for grid balancing, and prediction of highway congestion for traffic rerouting and optimization. We will refer to the process of forecasting several steps into the future as multi-step forecasting,[1] which can be written as

$$\hat{y}_{t+\tau} = f(y_{t-k:t}, \mathbf{x}_{t-k:t}, \mathbf{u}_{t-k:t+\tau}\mathbf{s}, \tau), \tag{2.4}$$

[1]Note that some literature refer to multi-step forecasting as *multi-horizon forecasting*. We prefer to use multi-step as the notion of horizon pertains to how far into the future one would like to forecast. Thus, multi-horizon can easily be confused with predicting over several different time horizons.

where $\tau \in \{1, \dots, T\}$ is the discrete forecast horizon, $\mathbf{u}_{t-k:t} = \{\mathbf{x}_{t-k}, \dots, \mathbf{x}_t, \dots, \mathbf{x}_{t+T}\}$ are known covariates such as date-time information and x_t are historically observed exogenous inputs [10]. Note that we have omitted the entity notation i in Equation 2.4 for brevity.

There are two main approaches for obtaining multi-step forecasts: iterative methods and direct methods. Iterative approaches typically involve the application of a one-step-ahead model and recursively feeding the model its own predictions to generate τ-step-ahead predictions. Therefore, any one-step-ahead model can effectively be used for multi-step forecasts. However, the error produced for each time step accumulates and can potentially lead to poor performance on longer prediction horizons [9]. Direct approaches attempt to deal with this issue by directly forecasting all time steps, which is the approach taken by sequence-to-sequence deep learning architectures [22].

A common way to benchmark time series forecasts is to compare them with the *naive method* for forecasting [20]. The naive method assumes that all future forecasts are equal to the previously observed value. When the time series is seasonal, we adjust the method by using the value observed at the same period in the previous season. Lastly, the naive method can also be applied for multi-step forecasting, in which case the previously observed value is used as the forecast for the future T steps. The naive, seasonal naive, and multi-step naive methods are given by

$$\begin{aligned} \text{Naive}: &\quad \hat{y}_t = y_{t-1}, \\ \text{Seasonal naive}: &\quad \hat{y}_t = y_{t-m}, \\ \text{Multi-step naive}: &\quad \hat{y}_{t+T} = y_{t-1}, \end{aligned} \tag{2.5}$$

where m is the seasonal period and T is the maximum forecast horizon. Naive forecasts are useful for benchmarking models, and is the baseline commonly employed for computing *scaled errors* as a performance metric. In the next section, we elaborate on scaled errors and other performance metrics commonly applied in the literature, in addition to their individual merits and issues.

2.3 Performance Metrics

In this section, we provide an overview of the range of error metrics that have been proposed to measure the performance of deterministic point forecasts. Furthermore, we discuss how the same metrics are extended and applied to multi-step forecasts by aggregating over the forecast horizon. Note that the literature interchangeably refers to performance metrics as accuracy measures, performance measures and evaluation metrics, to name a few. We will prefer the use of *performance metric* when referring to any metric that assesses the performance of point forecasts in any given way.

Assessing the accuracy of forecasting models is a challenging task due to the diversity of available time series, forecasting models, and different time series characteristics, as we have previously seen. Several metrics for forecasting performance have been proposed in the past. De Gooijer and Hyndman [1] provide a list of the most commonly used metrics before 2006, with further elaboration and review provided by Hyndman and Koehler [17]. The most common metrics for evaluating forecast performance are based on measuring the error between the predicted values, $\hat{y}_t$, and the ground truth values, y_t. We will refer to such metrics more specifically as *error metrics*.

Given a time series with n observations, let y_t denote the ground truth value at time t and $\hat{y}_t$ denote the forecasted value for y_t. Then, the forecast error at time t, e_t, is defined as $e_t = y_t - \hat{y}_t$. Provided a distribution of errors, the goal of selecting a metric is to provide an informative and clear summary of the error distribution while being aware that the selection of an appropriate metric will be highly dependent on the context in which it is used [23]. Table 2.1 provides an overview of the most commonly applied error metrics from five main classes. In the table, we introduce relative errors where e_t^* denotes a benchmark forecast error obtained by $e_t^* = y_t - \hat{y}_t^*$, where $\hat{y}_t^*$ is the forecast of some benchmark method for comparison. Like Hyndman and Koehler [17], we will use the notation mean(e_t) to denote the sample mean of errors $\{e_t\}$ over the period of interest, where the period could comprise of observations from the training, validation or test set depending on the context. Next, we briefly present the advantages and disadvantages associated with each class of error metrics.

Table 2.1: Overview of performance metrics based on forecast errors

Metric		Class	Equation
MSE	Mean Squared Error	Scale-dependent	$\text{mean}(e_t^2)$
RMSE	Root Mean Squared Error	Scale-dependent	$\sqrt{\text{MSE}}$
MAE	Mean Absolute Error	Scale-dependent	$\text{mean}(\lvert e_t \rvert)$
MAPE	Mean Absolute Percentage Error	Percentage based	$\text{mean}(\frac{\lvert e_t \rvert}{\lvert y_t \rvert})$
sMAPE	Symmetric Mean Absolute Percentage Error	Percentage based	$\text{mean}(\frac{2\times\lvert e_t \rvert}{\lvert y_t \rvert+\lvert \hat{y}_t \rvert})$
MRAE	Mean Relative Absolute Error	Relative errors	$\text{mean}(\lvert \frac{e_t}{e_t^*} \rvert)$
RelMAE	Relative Mean Absolute Error	Relative measures	$\frac{MAE}{MAE^*}$
MASE	Mean Absolute Scaled Error	Scaled errors	$\text{mean}(\frac{\lvert e_t \rvert}{\text{mean}(\lvert y_t - y_{t-1} \rvert)})$

Note: Variants using median and geometric mean instead of the mean exist for most of the measures.

Scale-dependent metrics are dependent on the scale of the data and should only be applied when comparing forecasts on the same dataset. While being easy to compute and understand, they tend to be sensitive to extreme outliers and consequently can produce biased results [23, 24]. Common scale-dependent measures include mean squared error (MSE), root mean squared error (RMSE) and mean absolute error (MAE). Note that RMSE is widely preferred over MSE as a performance metric, because the root operation returns errors on the same scale as the data. The use of RMSE in the first M-competition[2] was widely criticized due to the aforementioned problems [26, 27].

Percentage-based error metrics was proposed to be scale-independent, and thus can be used to compare forecast methods across datasets of differing scales. Mean absolute percentage error (MAPE) was used in the first M-competition, however, has been criticized for producing anomalies when observed values are close to or equal to zero, and has a bias favoring forecasts that are below the observed values [28]. Symmetric mean absolute percentage error (sMAPE) was proposed to combat the bias of MAPE, however, has also been criticized

[2]The M-competitions are a series of open forecasting competitions organized by Spyros Makridakis, the latest of which was M5 held in 2020 [25].

as an asymmetric metric and still has problems when y_t and $\hat{y}_t$ are close to zero [17]. Nonetheless, sMAPE is more resilient to outliers compared to metrics without error bounds.

Relative errors metrics are based on relative errors and deal with scale-dependence by dividing forecast errors by a benchmark error, e_t^*, often defined to be the result of the naive method where $\hat{y}_t = y_{t-1}$ and were first recommended by Armstrong and Collopy [23]. Although intuitively appealing, relative error metrics have issues when e_t^* is small or equal to 0.

Relative metrics are similar to relative errors, and are produced by the ratio of an error metric from the evaluated forecast method, e.g., MAE, and the error metric from a benchmark method, MAE_b. Similar relative metrics can be defined using MSE, MAPE, etc.[3] Although relative metrics are appealing due to their interpretability, they suffer from the same issues as the underlying base metric used in addition to requiring several forecasts on the same series to compute [28].

Scaled error metrics were introduced to address all of the aforementioned issues. Hyndman and Koehler [17] proposed the mean absolute scaled error (MASE) with the idea of scaling errors based on in-sample MAE from a benchmark method such as the naive method. A variant of MASE, root mean squared scaled error (RMSSE) was recently used to evaluate the M5 competition for multi-step hierarchical forecasting [25].

The error metrics are extended to the multi-step forecasting case by simply pooling the errors computed over the entire forecast horizon and computing the aggregate measure across all time steps to obtain a single value. This method of pooling and aggregating is commonly applied when evaluating a large number of models in empirical studies and forecasting competitions, see e.g., [30, 15, 31]. Although other performance metrics exist, they are typically custom made for a specific context or situation in which more creative evaluations of forecasts are possible. These are typically industry settings, where notable examples are the evaluation of solar power forecasting [32], solar irradiance forecasting [33] and activity recognition [34]. Although these possibilities are interesting, we aim to propose metrics that are applicable across a wide range of applications and not only for very specific time series. In the following two sections, we present background on state space models and deep learning architectures for multi-step forecasting.

2.4 State Space Models

State space models (SSMs) comprise a class of forecasting models that directly incorporate structural assumptions into the model. The distinguishing characteristic of SSMs is that observations are considered a composition of components that are modeled separately, such as trend, season, and noise. State space models are flexible and able to handle a wide range of problems when the structure of the time series is well understood [35]. Prominent examples of SSMs include the autoregressive integrated moving average (ARIMA) model [36] and exponential smoothing [37]. In this thesis, we apply seasonal ARIMA models (SARIMA) as a baseline for comparison with deep learning models.

An important underpinning of the standard ARIMA model is the autoregressive component, or AR model. In essence, an AR model forecasts future values by a linear combination of its previous values. In other words, it is a multiple regression using lagged values of the

[3] Special cases include Theil's U statistic, when the benchmark method is the naive forecast and the relative metric is RMSE or RMSPE [29].

target as regressors. Thus, the AR model can be written as

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t, \tag{2.6}$$

where ϵ_t is white noise and $\phi_1, \ldots, \phi_n$ are the parameters of the model [20]. We denote this as an AR(p) model, where p is referred to as the order of the model and represents how many lagged values of the target variable that is used to predict its future value.

Similar to the AR(p) model, we can define a moving average model, or MA(q) model for short. In contrast to an autoregressive model, the moving average model resembles a multiple regression using the current and previous forecast errors rather than the observed data. The MA(q) model is written as

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_p \epsilon_{t-p}, \tag{2.7}$$

where ϵ_t is white noise and $\theta_1, \ldots, \theta_p$ are the parameters to be estimated [35]. Hyndman and Athanasopoulos [20] notes that, applying the MA(q) model, "y_t can be thought of as a weighted moving average of the past few forecast errors." The AR and MA model will be revisited when we formalize the ARIMA model in chapter 4.

An important precaution for applying AR and MA models is that we require the data to be *stationary*. A stationary time series entails that statistical properties of the series, such as the mean and variance, do not change over time [20]. In order to identify whether a series is stationary or not, one typically applies a KPSS test [38]. Furthermore, non-stationary time series can be made stationary through the process of *differencing*. Differencing comprises several transformations that can be applied to reduce variance and de-trend or de-seasonalize a non-stationary time series. The KPSS test and combinations of these transformations are typically incorporated in statistical software for estimating the ARIMA model, such as the `auto.arima` function developed in the R language by Hyndman and Khandakar [39].

2.5 Deep Learning Architectures for Forecasting

An alternative approach to state space modeling is deep learning. Deep neural networks have become increasingly popular for the task of multi-step forecasting during recent years, due to their capability of extracting higher order features and identifying complex patterns without the need for direct human intervention [40]. Furthermore, they can learn from several time series without the need of fitting a separate model to every series, which is the case for SSMs. This, however, comes with an increased need for training data and often reduced interpretability.

Deep neural networks learn to predict by passing observations through a series of non-linear transformations, i.e., layers, to learn more abstract and useful feature representations [41]. In the context of time series, we can view the task of forecasting as encoding historical information as a latent variable, z_t, which is ultimately decoded to produce the forecast:

$$\begin{aligned} \hat{y}_{t+\tau} &= f(y_{t-k:t}, \mathbf{x}_{t-k:t}, \mathbf{u}_{t-k:t+\tau}\mathbf{s}, \tau) = g_{dec}(z_t), \\ z_t &= g_{enc}(y_{t-k:t}, \mathbf{x}_{t-k:t}, \mathbf{u}_{t-k:t+\tau}\mathbf{s}, \tau), \end{aligned} \tag{2.8}$$

where $\hat{y}_{t+\tau}$ is the τ-step ahead forecast, and $g_{enc}(\cdot)$, $g_{dec}(\cdot)$ are encoder and decoder functions, respectively [10]. Encoders and decoders are the basic building blocks that make up most of deep learning architectures, including architectures designed for temporal data.

Recurrent neural networks [42] (RNNs) is a canonical network type for processing sequential data, which several more advanced architectures are built upon [43]. As time series can be viewed as sequential data with a defined order, RNNs are also suitable for temporal modeling. The basic component of an RNN is the RNN cells, which store temporal data in the form of a hidden state:

$$\mathbf{z}_t = \nu(\mathbf{z}_{t-1}, y_t, \mathbf{x}_t, \mathbf{s}), \tag{2.9}$$

where $\mathbf{z}_t$ is the hidden state (i.e., memory) of the RNN, and $\nu(\cdot)$ is the learned memory update function [10]. In the simplest case of *Elman* RNNs [44], the output sequence is computed by iterating the following equations:

$$\begin{aligned} \mathbf{y}_{t+1} &= \sigma(\mathbf{W}_y \mathbf{z}_t + \mathbf{b}_y), \\ \mathbf{z}_t &= \sigma(\mathbf{W}_{z_1} \mathbf{z}_{t-1} + \mathbf{W}_{z_2} \mathbf{y}_t + \mathbf{W}_{z_3} \mathbf{x}_t + \mathbf{W}_{z_4} \mathbf{s} + \mathbf{b}_y), \end{aligned} \tag{2.10}$$

where σ is the sigmoid activation function, $\mathbf{x}_t$ are historically observed exogenous inputs and $\mathbf{s}$ is static metadata [10]. RNNs are commonly implemented as the encoder and decoder functions of Equation 2.8 in generic sequence-to-sequence networks [45, 22]. We will revisit sequence-to-sequence networks and other architectures in chapter 4 when introducing the models selected for experiments.

3 | Literature Review

The previous chapter presented preliminary theory on time series forecasting, performance metrics and the general structure of forecasting models. In this chapter, we review the literature on time series forecasting and performance metrics that is relevant to our work.

Most relevant to this thesis is the literature focusing on the evaluation of forecasting methods, i.e., research on performance metrics. Criteria for obtaining applicable and robust performance metrics have been defined and discussed in the literature: performance metrics should be reliable, valid, robust to outliers, scale-independent, and provide an informative summarization of the distribution of errors [23, 46]. Yet, performance metrics for forecasting have been highly debated throughout the existence of the field of time series forecasting. It has been well established that no single performance metric will be superior for all situations, and the identification of the "best" point forecast is highly dependent on the performance metric selected [47, 48, 16]. Nonetheless, several empirical studies have been conducted in attempts to compare and identify the most accurate models despite these observations [23, 19]. The most notable review on performance metrics was conducted by Hyndman and Koehler [17], identifying the problems of current metrics and proposing scaled errors and MASE as possible solutions.

Since Hyndman and Koehler [17], scaled errors have used in several studies and forecasting competitions [49, 15, 25] and have been identified as one of the better performance metrics in terms of statistical properties [50]. Other metrics have also been proposed, such as the unscaled mean bounded relative absolute error (UMBRAE) by Chen *et al.* [28], who analyze the proposed metric and its favorable statistical properties. However, we note that as the complexity of the metric increases (as is the case with UMBRAE), we find fewer studies applying the proposed metric.[1] Moreover, as we previously mentioned, the best point forecast will depend on the performance metric selected because different point forecasts will minimize the expected errors for various metrics [18]. Therefore, in this thesis, instead of focusing on proposing or selecting the single best error metric, we propose metrics that provide additional insights to the forecast practitioner and industry.

The emphasis on conducting larger empirical studies to evaluate methods against established baselines and state-of-the-art have been fundamental to the development of new forecasting methods [11]. As such, the evaluation of forecasting models through empirical studies and forecasting competitions have been a considerable feature of the Journal of Forecasting and the International Journal of Forecasting since their inception [19]. The first large scale empirical studies were conducted in the form of forecasting competitions, the first competition held by Makridakis *et al.* [51] in 1979 on a total of 1001 time series. The competition was widely criticized for employing inappropriate error metrics [27]. Since then, several empirical studies comparing forecasting models, different methods of preprocessing

[1] We have yet to find a paper employing UMBRAE as the selected performance metric.

and analysis of direct versus indirect forecasting strategies have been conducted [14, 52, 9, 25]. However, we have yet to identify studies that propose new metrics that allow industry and practitioners to further differentiate between models in the evaluation process, which is what we do.

Ben Taieb *et al.* [30] reviewed and compared direct and indirect forecasting strategies for multi-step time series based on data from the NN5 competition.[2] An and Anh [52] conducted a similar study, using neural networks exclusively. Both studies employed metrics such as MSE, MAPE and sMAPE to evaluate forecasting strategies, using time series from a single dataset. Although some results were presented with respect to the forecast horizon, the evaluation was mainly based on the aggregated metrics to draw general conclusions and recommendations when selecting a multi-step strategy. Parmezan *et al.* [31] performed a large scale empirical study, evaluating 11 methods on 95 different datasets. The results were evaluated in terms of a *multi-criteria* performance metric consisting of MSE and Theil's U coefficient, and prediction on change in direction (POCID) which, as claimed by the authors, allowed them to compare the investigated algorithms objectively [31]. The authors conclude with general recommendations and observations based on the aggregate of all results. Although general recommendations can be helpful when developing models for a new problem, they will not aid in the process of evaluation in a real-world industry setting where considerations other than average error are important. Our thesis provides suggestions for such tools in the form of metrics that can be applied in a variety of use cases.

Historically, neural networks have performed poorly in empirical studies, and their use for forecasting applications have been questioned on several occasions [53, 54, 14]. Furthermore, forecasting competitions received few to no submissions with methods based on neural networks. Most notably, the M3 competition featured only one neural network, which nonetheless performed poorly [12]. As a follow up, the NN3 competition was arranged using the same dataset to encourage additional submissions of neural network based methods. None, however, outperformed the original M3 competitors [14]. Makridakis *et al.* [15] later compared modern machine learning algorithms with statistical methods on the original M3 dataset, again finding statistical methods to be superior. It was concluded that, with the longest time series being only 126 observations, neural networks and machine learning are simply not fit for short univariate time series. Despite these findings, deep learning for forecasting applications has gained popularity during recent years [10].

The deep learning revolution has been driven by noteworthy achievements in the fields of image processing [55], reinforcement learning [56], and natural language processing [57]. Despite having shown poor performance on short univariate time series, recent advancements have proven deep learning to be applicable for time series forecasting where complex data representations are necessary. Several architectures have been proposed for time series forecasting depending on the type of application. Sequence-to-sequence RNNs and long short term memory networks (LSTMs) have become popular due to their ability to learn temporal patterns and long range memory [58, 59, 60, 61]. Temporal convolutional networks have been proposed as an alternative to RNNs, implementing long range memory through dilated convolutions and been shown to outperform the canonical Seq2Seq models on several tasks [62, 63, 64]. Attention based approaches [65, 66, 67, 68] and tensor factorization [69, 59, 70] have been proposed for tackling multivariate time series with a large numbers of input series. Lastly, a recent trend is the incorporation of uncertainty estimates that have

[2] A paper describing the results from the NN5 competition arranged in 2008 was never published.

been of particular interest for sales and demand forecasting, and implemented in the form of quantile regression networks [60, 68] and hybrid models [71, 72, 40]. In this thesis, we select four deep learning models based on different underlying architectures and extend the current literature on evaluating deep learning models in the context of multi-step point forecasting.

To summarize, the contributions of this thesis are as follows. First, we extend the current literature on performance metrics for multi-step time series forecasting, which is an area of research where there have been few recent contributions. Research on performance metrics has historically focused on proposing statistically robust metrics that can be aggregated to one number, however, we have yet to find papers that propose metrics for deriving additional insights in the evaluation process, which is what we do. Furthermore, we explicitly focus on multi-step forecasting, an aspect which is often neglected when discussing performance metrics.

Second, we propose four novel metrics and show their applicability through a large scale empirical study. Most empirical studies focus on providing practitioners with general recommendations for which models and techniques are suitable in different situations. In contrast, we focus on the use of new performance metrics and how they can be applied in the evaluation process. To the best of our knowledge, this is a novel contribution to the literature.

Lastly, we extend the current literature on evaluating deep learning models in the setting of point forecasting. We are the first study to compare the Seq2Seq, DeepAR, TCN and TFT models for multi-step point forecasting, and do so on five benchmark datasets. The results provide interesting findings, indicating that as models get better they tend to do so across all dimensions.

4 | Methodology

In this chapter, we present the methodology developed in order to answer the research question. Section 4.1 presents the proposed performance metrics that will provide additional insights during the evaluation of multi-step forecasting models. Section 4.2 presents the five multi-step forecasting models that have been selected for experiments: one state space model baseline and four deep learning architectures. These will provide a basis for obtaining experimental results that can be analyzed with the newly proposed metrics.

4.1 Metrics for Evaluating Multi-step Forecasts

In order to establish a baseline for evaluation, we follow the general approach in the literature and compute the aggregate of an appropriate error metric to obtain a single score for each model and dataset. As recommended by Hyndman and Koehler [17], we use scaled errors that are independent of the scale of the data, have a well defined mean and variance, and are symmetric by penalizing positive, negative, large and small forecasting errors equally. Although Hyndman and Koehler [17] prefers the mean absolute scaled error (MASE) over root mean squared scaled error (RMSSE), it is often unclear which metric will be most preferable in different situations [18]. For example, absolute errors are optimized for the median which results in lower error scores for methods that forecast closer to the median value [73]. If we consider a dataset that contains sporadic ranges of $y_t = 0$, which are common in solar power generation data and sales data, then MASE will be favorable for methods that forecast smaller values. In contrast, squared errors are optimized for the mean and will prefer forecasts with larger values compared to MASE [74]. For this reason, a variant of RMSSE was selected for use in the recent M5 competition [25]. In our case, we employ a variety of datasets where the use of either metric could be warranted depending on the context. Therefore, we employ both the MASE and RMSSE metrics in our evaluation and showcase the implications of the different metrics when evaluating the results.

To obtain the MASE and RMSSE metric, we first define the scaled errors prior to aggregating, i.e., prior to performing the mean and root mean across all observations. This is done to allow aggregation and grouping of the scaled errors across different dimensions. For example, we can compute the MASE per time series or per step in the forecast horizon. Additionally, we use the definition of scaled errors when deriving new metrics.

We begin by defining the absolute scaled error, ASE_t^i, of time series i at time t as

$$ASE_t^i = |q_t^i| \quad \text{where} \quad q_t^i = \frac{e_t^i}{\text{MAE}\big(Naive(i,T,m)\big)}. \tag{4.1}$$

In Equation 4.1, q_t^i denotes the scaled error at time t for time series i, and $Naive(i,T,m)$ denotes forecasts obtained on the training data of time series i using the seasonal naive

multi-step method with a T-step forecast horizon and seasonal periodicity m. Hence, the denominator consists of the mean absolute error (MAE) computed from the seasonal multi-step naive method on the training data. It is important to note that we scale the errors from each time series i *separately* because they can be of different scales. This is easily overlooked when pooling several hundreds of series as one single dataset, which is common practice when applying deep learning models. Moreover, note that we use seasonal naive multi-step forecasts as the baseline, in contrast to single-step forecasts. Naive single-step forecasts are more commonly applied for this purpose, likely due to their simpler computation, see e.g., [25]. We elect to use multi-step forecasts as this keeps the interpretability of scaled errors clearer. The naive multi-step forecasts accurately represents the benchmark that would be possible to obtain using the naive method, in contrast to naive single-step.[1] After obtaining absolute scaled errors, we compute the aggregated MASE metric as

$$MASE = \text{mean}\big(\text{ASE}_t^i\big), \tag{4.2}$$

where the mean operation is computed over all time series i and time steps t.

Similarly, we define the squared scaled errors, SSE_i^t, of time series i at time t as

$$SSE_t^i = (q_t^i)^2 \quad \text{where} \quad q_t^i = \frac{e_t^i}{\text{RMSE}\big(Naive(i,T,m)\big)}, \tag{4.3}$$

q_t^i denotes the scaled error at time t and $Naive(i,T,m)$ denotes seasonal naive multi-step forecasts on the training set as above. Note that for squared scaled errors, the denominator is computed using the corresponding RMSE measure, as recommended by [17]. Again, the aggregated RMSSE metric is obtained by

$$RMSSE = \sqrt{\text{mean}\big(\text{SSE}_t^i\big)}, \tag{4.4}$$

where the root mean operation is computed over all time series i and time steps t.

MASE and RMSSE will be our baseline evaluation metrics and represent how empirical studies typically summarize and evaluate multi-step forecasts. To provide further insights that are neglected by only evaluating results in terms of aggregated metrics, we propose four novel metrics and implement a fifth temporal alignment metric.

4.1.1 Win-loss Ranking

The aggregated MASE and RMSSE metrics tell us how well a model performs on an entire dataset overall, disregarding the information about which time series they perform well on. To identify how well models perform with respect to each time series, we propose a win-loss ranking metric to rank models according to their win-loss counts.

Assume that we have a time series dataset $D = \{1, \dots d\}$ where $i \in D$ denotes a single series in the dataset. Furthermore, assume that we evaluate a set of models $H = \{1, \dots h\}$ on dataset D, where $g \in H$ denotes a single model in the set of models. Letting the aggregated error obtained by model g on time series i be denoted by $e_{g,i}$, we define the set of time series in

[1] Using the naive single-step forecast would imply that, forecasting across T-time steps, the model would iteratively have access to future information that a forecasting model would not.

which model g obtains the lowest error (best score) as W_g. The win-rank of model g is then given by

$$R_W = |W_g|, \tag{4.5}$$

where R_W represents the number of times model m obtains the best score, i.e., wins, on dataset D. Similarly, we define the set of time series in which model g obtains the highest error as L_g, whereby the loss-rank can be written as

$$R_L = |L_g|, \tag{4.6}$$

where R_L is the number of times model g obtains the worst score on dataset D.

The win-loss ranks of models can provide further insights when evaluating models on datasets with a large number of time series. For example, consider the case where the aggregated error metric decisively indicates that one model is better than its peers. It is possible that it is better on average on all series, or conversely, it could be very good at some series and very bad at others. It might turn out that it is optimal to use different models on different parts of the dataset, a fact that is, not necessarily neglected, but hidden when just using aggregated metrics. We will see concrete examples of this when we evaluate our results in chapter 6.

4.1.2 Variance Weighted Errors

Seasonal time series that are driven by real-world phenomena, such as electricity consumption and traffic congestion, tend to be highly variable over the seasonal period. For example, traffic tends to be easy to predict during baseline hours and nighttime. The real uncertainty arises during rush hour around 08:00 AM and 16:00 PM when traffic congestion spikes. Electricity consumption is highly variable during daytime depending on external factors such as weather and consumer patterns. Furthermore, different parts of a time series might not be that relevant to forecast. When considering the use of traffic forecasts for rerouting and optimization traffic, rerouting might be irrelevant during nighttime, making the use of forecasting nighttime congestion less important. It is not unreasonable to assume that practitioners are in most need of accurate forecasts when uncertainty is at its greatest, i.e., during rush hour traffic and daytime electricity consumption. The standard error metrics do not account for these differences in uncertainty over the seasonal period when evaluating forecast performance. Therefore, we propose to include a proxy for uncertainty directly in the error metric.

For the purpose of including uncertainty in the error metric, we propose *variance weighted errors*. By this approach, we weight the errors obtained at a particular time during the seasonal period by the variance of the target values at that same particular time. To illustrate this, Figure 4.1 shows a plot of 30 consecutive days of measured traffic occupancy from a single time series. It should be evident that the variance around rush hour is significantly higher than during the middle of the day and during nighttime. Thus, we want to scale the errors of forecasts to closer match this distinct feature of the dataset. I.e., by increasing the importance of errors made under high uncertainty and decreasing the importance of errors when traffic is more stable.

To formalize this concept, assume we are given a dataset with a logical seasonal periodicity m and date-time covariates $c \in \{1, \ldots, m\}$.[2] The covariates represent a partitioning of the

[2]The Oxford Dictionary defines covariates as "independent variables that can influence the outcome of a given statistical trial, but which is not of direct interest."

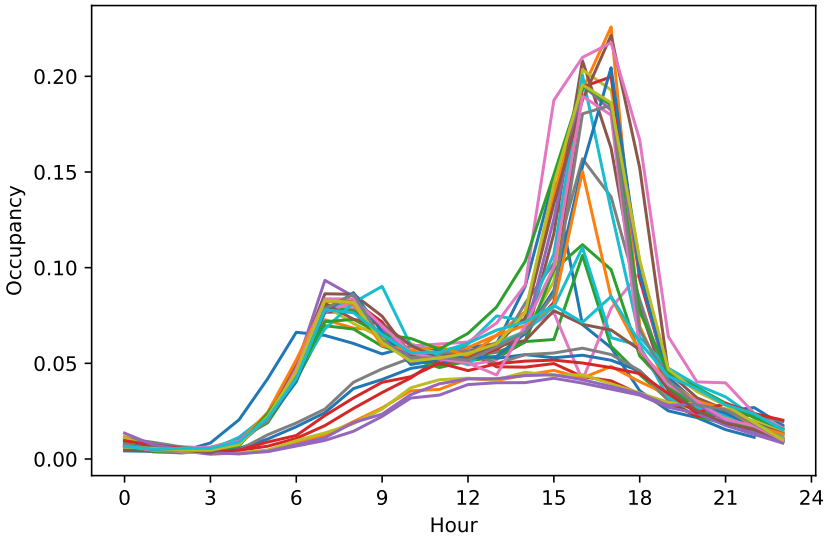


Figure 4.1: An example of daily traffic occupancy plotted for 30 consecutive days

seasonal period. For example, given traffic measured at every hour, the seasonal period is $m = 24$ and each seasonal period constitutes a day with hours $c \in \{1, 2, \ldots, 24\}$. The covariate in this case is the hour of the day at which a forecast was made. A covariate could also be the month of year, or day of the week. The important point is that the covariates c represent a logical partitioning of the seasonal period where one expects different behavior at different points during the seasonal period. We define the estimated variance, $\hat{V}_c$, of the target values y_t^c at covariate c as $\hat{V}_c = \text{Variance}(y_t^c)$, where $t \in \{c, c \cdot 1, c \cdot 2, \ldots, c \cdot \frac{n}{m}\}$. That is, $\hat{V}_c$ is the estimated variance of the targets occurring during covariate c. To obtain the variance weight, w_c, we normalize the estimated variance $\hat{V}_c$ by the sum of variances and scale by the seasonal periodicity:

$$w_c = \frac{\hat{V}_c}{\sum_{c=1}^{m} \hat{V}_c} \times m. \tag{4.7}$$

Therefore, by Equation 4.7, $\sum_{c=1}^{m} w_c = m$ and each weight will act as a scaling factor according to how much the target varies during a particular covariate. In the special case where the variance is equal for all covariates, we have $w_c = 1$ for $c \in \{1, \ldots, m\}$ and the errors will not be changed by the weights. Note that we have omitted the superscript i notation representing the time series to simplify the notation.

Continuing, we define the *variance weighted* MASE (MASE_{VW}) which we derive from the absolute scaled errors as defined in Equation 4.1. Given weight w_c^i obtained on time series i at covariate c, we define the MASE_{VW} as

$$MASE_{VW} = \text{mean}\big(ASE_c^i \cdot w_c^i\big), \tag{4.8}$$

where ASE_c^i is the absolute scaled error obtained on time series i grouped by covariate c, and the mean operation is computed across all time series and covariates. Note that the reason for scaling prior to averaging is that the MASE_{VW} can be computed per time series or per covariate depending upon which variables the mean is performed over.

Similarly, we define the variance weighted RMSSE (RMSSE_{VW}) as

$$RMSSE_{VW} = \sqrt{\text{mean}\big(SSE_c^i \cdot w_c^i\big)}, \tag{4.9}$$

where SSE_c^i and w_c^i are the absolute scaled error and variance weight obtained on time series i grouped by covariate c, respectively. The root mean operation is again computed across all time series and covariates.

4.1.3 Delta Horizon Metric

The two previous metrics address how models perform on individual time series in a dataset, and how forecasts across the seasonal period can mask the performance of models during critical time points. To gain further insight into how models behave with respect to the multi-step forecast horizon, we propose the *delta horizon* metric, or Δ_H. The Δ_H metric measures how much a multi-step forecasting model updates the forecasted value for y_t at time t as the time point approaches. For example, given a 24-step forecast at time t, the 24^{th} step is the forecasted value $\hat{y}_{t+24}$ for y_{t+24}. At $t+1$, the model produces a new forecast $\hat{y}_{t+24}$ for y_{t+24}, which is now a 23-step ahead forecast. In other words, the model updates its initial forecast for y_{t+24} as t approaches $t+24$. We might find that model X adjusts its forecasted value for y_t significantly over the forecast horizon, while model Y produces stable forecasts over the horizon, relative to X. This is what we aim to capture with the Δ_H metric.

To formalize the above, let $\hat{y}_t^\tau$ denote the τ-step ahead forecast for the observed value y_t and $\tau \in \{1, \dots T\}$ where T is the forecast horizon. Then, we define the delta horizon metric as

$$\Delta_H = \text{mean}\big(|\hat{y}_t^\tau - \hat{y}_t^{\tau-1}|\big), \quad \text{for} \quad 2 \leq \tau \leq T, \tag{4.10}$$

where the mean operation is performed over both τ and t. Hence, the delta horizon metric measures the mean absolute difference between each update of $\hat{y}_t$ over the forecast horizon τ.

4.1.4 Decomposed Error Metrics

The last metrics we propose are *decomposed errors*, which relate errors to the time series trend-cycle component, seasonality component and remainder component (noise). By this approach, we can in more detail compare performance across models or datasets and analyze how the models perform with respect to the components of the time series. The purpose of this is to provide further insights to industry and practitioners when interpreting their results and provide insights that can allow practitioners to build better models.

We begin by denoting the observed ground truth values y_t and the τ-step ahead forecast for the observed value y_t as $\hat{y}_t^\tau$ for $\tau \in \{1, \dots T\}$, where T denotes the maximum forecast horizon. Then, we define the *forecasted path* with forecast horizon τ as $p_\tau = \{\hat{y}_t^\tau, \hat{y}_{t+1}^\tau, \dots, \hat{y}_{t+n}^\tau\}$ where t and n are the starting point and length of the test series, respectively. Similarly, we define the path of the ground truth test series as $p_G = \{y_t, y_{t+1}, \dots, y_{t+n}\}$. Thus, p_1, p_T and p_G represents the path of all 1-step-ahead forecasts, T-step-ahead forecasts and ground truth test series, respectively. Note that we have omitted denoting the time series by index i to simplify notation.

With the above, we define decomposed errors as the error between each time series component resulting from the STL-decomposition of p_τ and p_G. Thus, we let $\mathcal{T}_{t,p_\tau}$, S_{t,p_τ}, R_{t,p_τ} denote the trend-cycle, seasonal and remainder components of p_τ respectively, and $\mathcal{T}_{t,p_G}$, S_{t,p_G}, R_{t,p_G} denote the components of p_G. Then, we define the decomposed trend, season,

and remainder errors as

$$\begin{aligned} \text{Trend error:} \quad & e_{t,\mathcal{T}} = \mathcal{T}_{t,p_G} - \mathcal{T}_{t,p_\tau}, \\ \text{Seasonal error:} \quad & e_{t,S} = S_{t,p_G} - S_{t,p_\tau}, \\ \text{Remainder error:} \quad & e_{t,R} = R_{t,p_G} - R_{t,p_\tau}. \end{aligned} \tag{4.11}$$

In order to summarize the errors, we use the same approach as for scaled errors and calculate the RMSSE version of each decomposed metric as follows:

$$\begin{aligned} RMSSE_{\mathcal{T}} &= \sqrt{(q_{i,t,\mathcal{T}})^2} \quad \text{where} \quad q_{i,t,\mathcal{T}} = \frac{e_{i,t,\mathcal{T}}}{\text{RMSE}\big(Naive(i,T,m)\big)}, \\ RMSSE_S &= \sqrt{(q_{i,t,S})^2} \quad \text{where} \quad q_{i,t,S} = \frac{e_{i,t,S}}{\text{RMSE}\big(Naive(i,T,m)\big)}, \\ RMSSE_R &= \sqrt{(q_{i,t,R})^2} \quad \text{where} \quad q_{i,t,R} = \frac{e_{i,t,R}}{\text{RMSE}\big(Naive(i,T,m)\big)}, \end{aligned} \tag{4.12}$$

where $\text{RMSE}\big(Naive(i,T,m)\big)$ denotes the RMSE of the seasonal naive multi-step forecasts on the training set of time series i with seasonal period m, as when computed for scaled errors. Note that we include the i notation to show that each metric is scaled according to each individual time series. Decomposed errors can also be computed and aggregated by MASE instead of RMSSE, although we elect to compute only the RMSSE version to keep the number of metrics for the evaluation tractable.

The decomposed metrics indicate how accurately the model captures the trend, season, and remainder components by each τ-step-ahead path, allowing practitioners and industry to see whether the error is mainly due to erroneous prediction of trend, season or remainder (noise). We compute the decomposed errors of p_1 and p_T to compare how the models capture the time series components at the first and last prediction horizons. Henceforth, we will refer to the three metrics $RMSSE_{\mathcal{T}}$, $RMSSE_S$, $RMSSE_R$ as simply the "decomposed errors" for path p_1 and p_T.

4.1.5 Temporal Distortion Metrics

The metrics we have discussed so far are all derived from forecast errors, i.e., $e_t = y_t - \hat{y}_t$. The reason being that there are few alternatives. One alternative that has been recently proposed in the context of evaluating solar irradiance forecasts is the *temporal distortion index* (TDI) [33]. The TDI does not measure the error of a forecast by how much it deviates in value from the target at each time point. Rather, it measures the degree to which the forecast is aligned with the ground truth data in time. For example, by producing one-step-ahead naive forecasts, one obtains the exact same pattern as the underlying time series, only shifted one observation back in time. Thus, the naive method produces the correct shape, but the incorrect alignment in time.[3] The TDI could serve as an alternative to error metrics, or, at the very least, provide additional insights in terms of how temporally aligned the produced forecasts are.

[3]The TDI metric is closely related to dynamic time warping (DTW) and time series classification, although we do not visit these concepts in this thesis.

Following Vallance *et al.* [33], the idea behind the TDI is to align a test time series $T = (T_1, T_2, ...T_n)$ with a reference time series $R = (R_1, R_2, ...R_n)$ by warping the time axis of T. The local distance that associates the temporal indices of the two series is defined as

$$\forall (i, j)[1 : N]^2, \quad d(T_i, R_j) = ||T_i - R_j||, \tag{4.13}$$

and a path between indices of the test series and reference series as

$$w = ((i_1, j_1), \ldots, (i_l, j_l), \ldots, (i_k, j_k)) \quad \text{with} \quad k \in \mathbb{N}. \tag{4.14}$$

Dynamic programming is used to find the optimal bath between indices in T and R by minimizing the total cost of possible paths. The total cost function is defined as the sum of local distances (4.13) between the two series, calculated on the sequence of path points (i_l, j_l) for $l \in [1, 2, ..., k]$. Then, the TDI is defined as the area between the optimal path and the identity path, normalized by the area under the identity path corresponding to a percentage of the maximum temporal distortion [75, 33]:

$$TDI = \frac{1}{N} \sum_{l=1}^{k-1} |(i_{l+1} - i_i)(i_{l+1} + i_i - j_{l+1} - j_l)|. \tag{4.15}$$

In addition to the TDI, Vallance *et al.* [33] defines a *temporal distortion mix* (TDM), which is a more interpretable version of the TDI. The TDM is expressed as a signed percentage, where −100% and 100% represents a systematically in advance and late forecast respectively, and provides an indication of how forecasts are aligned in time. For details on calculating the TDM metric we refer to [33].

To employ the TDI and TDM in our evaluation, we compute the aggregate average to indicate how models perform temporally. Because the TDI metric requires computation over the entire forecast horizon, the temporal metrics cannot be grouped by date-time covariates or forecast horizon. In the next section, we present the five forecasting models that we evaluate in our experiments.

4.2 Forecasting Models

In order to test the applicability of the proposed metrics, we run experiments with five forecasting models: seasonal-ARIMA (SARIMA) [36], a generic sequence-to-sequence network (Seq2Seq) [22], a generic temporal convolutional network (TCN) [64], the DeepAR model [72], and the Temporal Fusion Transformer (TFT) [68]. The SARIMA model is used as the baseline SSM in this study and is commonly applied in the literature to benchmark new forecasting models. The deep learning models have been selected for the sake of using models based on different architectures. The differences in architecture should provide more diverse forecasts that allow us to evaluate the proposed metrics for a range of different use cases.

Furthermore, to increase the difference between models, DeepAR and TFT will be implemented with access to date-time covariates and static metadata (i.e., the time series ID of entity i). In contrast, Seq2Seq and TCN will solely be trained on the time series data with no additional features. The rationale being that we increase the probability of models learning different feature representations. This, in turn, should provide more diversity in forecasted values, and hence different use cases for the proposed metrics.

4.2.1 SARIMA

The SARIMA model is a generalization of the Box-Jenkins ARIMA model, which can accommodate data with both seasonal and non-seasonal features [36]. The standard ARIMA model is a combination of the autoregressive (AR) and moving average (MA) models described in section 2.4, in addition to an *integration* component that ensures the fitted data is stationary. We specify a model by writing ARIMA(p,i,q), where p, i and q represent the AR order, degree of integration and MA order, respectively. The model can be written as

$$y'_t = c + \epsilon_t + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_p \epsilon_{t-p}, \tag{4.16}$$

where ϵ_t is white noise, y'_t is the first differenced series, $\phi_1,\ldots,\phi_n$ are the AR components, and $\theta_1,\ldots,\theta_p$ are the MA components [20]. As with the AR and MA models, stationarity is required when estimating the model. Observe that the ARIMA model is just a combination of the underlying AR(p) and MA(q) models, such that an ARIMA$(p,0,0)$ model is the equivalent of an AR(p) model, and ARIMA$(0,0,q)$ the equivalent of MA(q). The ARIMA model is estimated by maximum likelihood estimation (MLE), maximizing the probability of obtaining the data that was observed. For further details on estimation, we refer to [20].

Three of the datasets we employ in our experiments are highly seasonal, therefore, an SARIMA model is needed to account for seasonality. The SARIMA model extends ARIMA by including differences at lags equal to the seasonal period, m, to remove seasonal effects from the series. The seasonal component of SARIMA comprises of four additional terms, and we write it as SARIMA$(p,i,q)(P,D,Q)_m$, where P is the number of seasonal AR terms, D is the number of seasonal differences, Q is the number of seasonal MA terms and m is the seasonal period. The seasonal terms are similar to non-seasonal terms but differ by being lagged m times to correspond to the seasonal period. The terms are added to the model simply by multiplying them with the equivalent non-seasonal terms. By first defining the *backshift operator*, B, as $By_t = y_{t-1}$ [36], we can write an SARIMA$(1,1,1)(1,1,1)_4$ model as

$$(1-\phi_1 B)(1-\Phi_1 B^4)(1-B)(1-B^4)y_t = (1+\theta_1 B)(1+\Theta_1 B^4)\epsilon_t, \tag{4.17}$$

where $(1-\phi_1 B)$ and $(1+\theta_1 B)$ are the non-seasonal AR(1) and MA(1) components, $(1-\Phi_1 B^4)$ and $(1+\Theta_1 B^4)$ are the seasonal AR(1) and MA(1) components, and $(1-B)$, $(1-B^4)$, are the non-seasonal and seasonal differences, respectively.

To select appropriate model orders, we use the stepwise algorithm by Hyndman and Khandakar [39], designed to automate model selection when forecasting a large number of univariate time series. The selection of the optimal model order is based on minimizing the Akaike information criterion (AIC) [76]. Further details on the application of the SARIMA model in our experiments is provided in chapter 5 on the experimental setup. Next, we present the deep learning models.

4.2.2 Sequence-to-Sequence Network

The first deep learning model we implement in our study is based on a general sequence-to-sequence architecture, also referred to as an encoder-decoder architecture. It was popularized by Sutskever *et al.* [22] and has historically produced strong results on natural language processing tasks [77]. As time series can be viewed as sequences of inputs and targets, several sequence-to-sequence architectures have been proposed for forecasting [72, 40, 70]. We

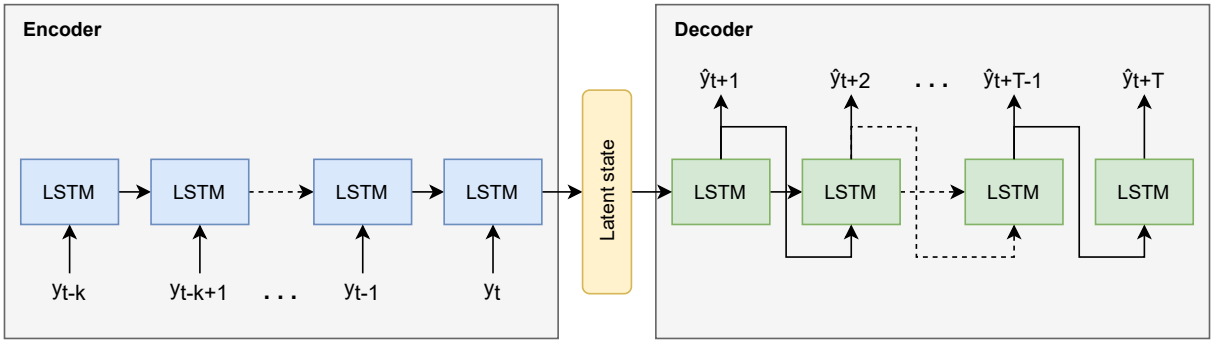


Figure 4.2: Overview of a generic sequence-to-sequence architecture with separate LSTM networks in the encoder and decoder

implement a basic version of the sequence-to-sequence architecture as a baseline for our deep learning models, which we will henceforth refer to as the Seq2Seq model.

In accordance with [22], we implement the Seq2Seq model by combining an encoder and a decoder in the form of two *long short term memory networks* [78] (LSTMs), as shown in Figure 4.2. LSTMs were proposed to mitigate the problems of exploding and vanishing gradients in RNNs, as we discussed in section 2.5. Furthermore, LSTMs are better at exploiting long range dependencies [45], making them suitable for time series modeling. This is achieved by adding a cell state, $\mathbf{c}_t$, which acts as the long-term memory of the network and is modulated by three *gates*:

$$\begin{aligned}
\text{Input gate}: &\quad \mathbf{i}_t = \sigma(\mathbf{W}_{i_1}\mathbf{z}_{t-1} + \mathbf{W}_{i_2}y_t + \mathbf{b}_i),\\
\text{Output gate}: &\quad \mathbf{o}_t = \sigma(\mathbf{W}_{o_1}\mathbf{z}_{t-1} + \mathbf{W}_{o_2}y_t + \mathbf{b}_o),\\
\text{Forget gate}: &\quad \mathbf{f}_t = \sigma(\mathbf{W}_{f_1}\mathbf{z}_{t-1} + \mathbf{W}_{f_2}y_t + \mathbf{b}_f),\\
\text{Hidden state}: &\quad \mathbf{z}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t),\\
\text{Cell state}: &\quad \mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh(\mathbf{W}_{c_1}\mathbf{z}_{t-1} + \mathbf{W}_{c_2}y_t + \mathbf{b}_c),
\end{aligned} \tag{4.18}$$

where $\mathbf{z}_t$ is the LSTM hidden state, $\sigma(\cdot)$ is the sigmoid activation function, $\tanh(\cdot)$ is the tanh activation function and $\odot$ is the Hadamard product [45]. Note that the Seq2Seq model will be trained solely on the time series data, with no additional static metadata or date-time covariates as features.

4.2.3 Temporal Convolutional Network

Temporal convolutional networks are based on convolutional layers and will provide a novel architecture type for our experiments. Furthermore, TCNs have recently been shown to outperform canonical RNN and LSTM architectures on selected sequence-to-sequence tasks [64] and will be interesting to compare with the Seq2Seq model. Several versions of the temporal convolutional architecture have been proposed for different use cases [79, 62, 63]. We follow the architecture proposed by Bai *et al.* [64], implementing a generic TCN that combines simplicity, autoregressive prediction, and long memory.

The TCN is based on two principles: 1) an input sequence of any length can be mapped to an output sequence of any length. 2) *causal convolutions* that ensure information from

the future cannot be accessed in the past. To accomplish the former point, the model is implemented as a 1D fully-convolutional network [80]. For the latter, we enforce the restriction that output at time t is solely convolved with items at time t and earlier. To increase the memory of the causal convolutions, the TCN implements *dilated convolutions* [79]. Given a sequence $\mathbf{x} \in \mathbb{R}^n$ and a filter $f : \{0, \dots, k-1\} \to \mathbb{R}$, the dilated convolution F on the item s in the sequence is defined as

$$F(s) = (\mathbf{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot (x)_{s-d \cdot i}, \tag{4.19}$$

where $*$ is the convolution operator, d is the dilation factor, k is the filter size and $s - d \cdot i$ defines the direction of the past [64]. The TCN is easily adapted to the task of forecasting by interpreting the sequence $\mathbf{x}$ as a time series. The dilation factor d introduces fixed steps between every two adjacent filters and is increased exponentially for each convolutional layer in the network, increasing the receptive field back in time.

Bai *et al.* [64] implements *residual blocks* [81] in place of convolutional layers in order to stabilize the training of very deep networks. Each block comprises two dilated convolutional layers, weight normalization [82], the ReLU activation function [83] and dropout for regularization [84]. For further details on TCNs, we refer to their original paper [64].

Similar to the Seq2Seq model, we do not add any static metadata or date-time covariates to the TCN model. In the following two sections, we present the DeepAR and TFT models which will incorporate static and date-time covariates as features.

4.2.4 DeepAR

DeepAR is a probabilistic model based on an autoregressive RNN architecture, developed by Amazon Research for probabilistic sales and demand forecasts [72]. DeepAR outperformed several models for demand forecasting, including Croston and ETS [37, 39], and outperformed the matrix factorization model [69] at point forecast accuracy on the electricity and traffic datasets, highlighting its applicability for point forecasting. The model has since been established as the baseline when proposing new architectures and is therefore selected for our experiments.

Denoting the value of time series i at time t by $y_{i,t}$, the input lookback window as k and the prediction horizon, i.e., number of steps to forecast into the future, as T. The goal of DeepAR is to model the conditional distribution $P(\mathbf{y}_{i,t:t+T} | \mathbf{y}_{i,t-k:t}, \mathbf{x}_{i,t-k:t+T})$, where $\mathbf{y}_{i,t:t+T} = \{y_{i,t}, \dots, y_{i,t+T}\}$ is the series of τ-step ahead predictions, $\mathbf{y}_{i,t-k:t} = \{y_{i,t-k}, \dots, y_{i,t}\}$ are prior observations of the target, and $\mathbf{x}_{i,t-k:t+T}$ are known covariates. Note that t represents the time at which a T-step ahead prediction is made given prior data and covariates. Following [72], the DeepAR model assumes that the model distribution, Q_Θ, consists of a product of likelihood factors as follows:

$$Q_\Theta\big(\mathbf{y}_{i,t:t+T} | \mathbf{y}_{i,t-k:t-1}, \mathbf{x}_{i,t-k:t+T}\big) = \prod_{t=t}^{t+T} Q_\Theta(y_{i,t} | \mathbf{y}_{i,t-k:t+1}, \mathbf{x}_{i,t-k:t+T}) = \prod_{t=t}^{t+T} \ell\big(y_{i,t} | \theta(\mathbf{h}_{i,t}, \Theta)\big), \tag{4.20}$$

and are parameterized by the output $\mathbf{h_{i,t}}$ of an autoregressive RNN:

$$h_{i,t} = \nu\big(\mathbf{h}_{i,t-1}, y_{i,t-1}, \mathbf{x}_{i,t}, \Theta\big), \tag{4.21}$$

where $\nu(\cdot)$ is the learning function implemented as a multi-layer LSTM network, and the likelihood factors $\ell\big(z_{i,t}|\theta(\mathbf{h}_{i,t}\big)$ are fixed distributions with parameters given by the network output $\mathbf{h}_{i,t}$ [72]. The model is autoregressive and recurrent, as both the previously observed input, $y_{i,t-1}$, and the previous network output, $\mathbf{h}_{i,t-1}$, is used as input when obtaining the next output. The parameters Θ of the model, consisting of the RNN $\nu(\cdot)$ function and the parameters of θ (the likelihood model), are learned by maximizing log likelihood.

In accordance with [72], we use the presented architecture as both encoder and decoder, i.e., in a sequence-to-sequence type architecture. Furthermore, we use Gaussian likelihood, $\ell(y|\mu,\sigma)$, as the distribution model, where the Gaussian likelihood is parameterized by the mean and standard deviation, i.e., μ,σ, respectively. When predicting, we feed the model the input series $\mathbf{y}_{i,t-k:t}$ and sample $\hat{y}_{i,t} \sim \ell(\cdot|\theta_{i,t})$, before refeeding $\hat{y}_{i,t}$ to the model for the next prediction step and repeating the process to the end of the prediction horizon T, generating one sample path. Repeating the sampling process generates several paths representing a joint predicted distribution for the $\tau \in \{1,\dots,T\}$ predictions steps. The point forecast is then simply the mean of the predicted distributions. Quantiles can also be computed from the predicted joint distribution to obtain prediction intervals, although we solely evaluate point forecasts in this thesis. For more in-depth details on the DeepAR model, including optimization and scale handling, we refer to the original paper by Salinas *et al.* [72].

4.2.5 Temporal Fusion Transformer

The last model we implement is the TFT developed by Google Research [68], achieving state-of-the-art performance on multi-step forecasting compared to several recently proposed models [69, 60, 72, 40, 85]. The model is built on an attention based RNN architecture and combines several components, including gating mechanisms, variable selection networks, covariate encoders, temporal attention and quantile forecasts, an overview of which is shown in Figure 4.3. The TFT will represent the state-of-the-art model in our experiments that combines elements from several archetypal network architectures, most notably the transformer architecture [86]. Keep in mind that the TFT model description is quite complex, hence, for a complete description of the architecture we refer to the original paper by Lim *et al.* [68].

The TFT model, like the DeepAR model, produces prediction intervals, however, it does so by quantile regression rather than sampling from estimated distributions. Quantile forecasts are given as

$$\hat{y}_i(q,t,\tau) = f_q\big(y_{i,t-k:t}, \mathbf{z}_{i,t-k,t}, \mathbf{x}_{i,t-k,t+\tau}, \mathbf{s}_i, \tau\big), \tag{4.22}$$

where $\hat{y}_i(q,t,\tau)$ is the predicted q^{th} quantile of the τ-step ahead forecast at time t and f_q is the prediction model [68]. Note that the TFT model separates time dependent known and unknown covariates, treating them differently in the architecture. Known covariates, i.e., time of day or day of week at time t, are denoted as $x_{i,t}$ and unknown covariates such as additional external regressors are denoted as $z_{i,t}$.

First, the network architecture implements a gating mechanism to apply non-linear processing on targets and covariates only where this is beneficial. The rationale being that non-linear processing for e.g., noisy datasets is unnecessary and can result in overfitting. The

gated residual network (GRN) constitutes one building block of the TFT [68]:

$$\begin{aligned} \mathrm{GRN}_\omega(\mathbf{a},\mathbf{c}) &= \mathrm{LayerNorm}\big(\mathbf{a} + GLU_\omega(\eta_1)\big), \\ \eta_1 &= \mathbf{W}_{1,\omega}\eta_2 + \mathbf{b}_{1,\omega}, \\ \eta_2 &= \mathrm{ELU}\big(\mathbf{W}_{2,\omega}\mathbf{a} + \mathbf{W}_{3,\omega}\mathbf{c} + \mathbf{b}_{2,\omega}\big), \end{aligned} \tag{4.23}$$

where ELU is the exponential linear function [87], η_1, η_2 are intermediate layers, LayerNorm denotes layer normalization [88], GLU denotes *gated linear units* as in [62] and the ω index denotes weight sharing in the network.

Second, TFT uses separate GRN encoders to process static covariates differently, producing context vectors $\mathbf{c}_s, \mathbf{c}_e, \mathbf{c}_c, \mathbf{c}_h$. Context vectors are inserted at specific locations in the temporal fusion decoder in order to enhance the temporal features with static information [68].

Third, TFT implements *variable selection networks* which are used to process static and time dependent covariates. The variable selection networks improve the interpretability of the model by providing weights to covariates, indicating which covariates are most influential for prediction. Furthermore, the variable selection network reduces the weight of features identified to be noisy and that might negatively impact performance. In short, variable weights are generated by feeding past inputs, Ξ_t, and a context vector c_s through a GRN for variable selection:

$$\mathbf{v}_{\chi_t} = \mathrm{Softmax}\big(\mathrm{GRN}_{\mathbf{v}_\chi}(\Xi_t, \mathbf{c}_s)\big), \tag{4.24}$$

where Softmax is the softmax activation function [43] and $\mathbf{v}_{\chi_t}$ is the vector of variable weights [68]. Post variable selection, outputs are fed to a sequence-to-sequence encoder-decoder architecture with LSTM layers, where observed features are fed to the encoder and known future features are fed to the decoder. The output is passed to the *temporal fusion decoder*. The core feature of the TFT is the temporal fusion decoder, which incorporates the static covariates, a masked interpretable multi-head attention layer adapted from [86] and position-wise feed forward networks before producing outputs, as seen in Figure 4.3.

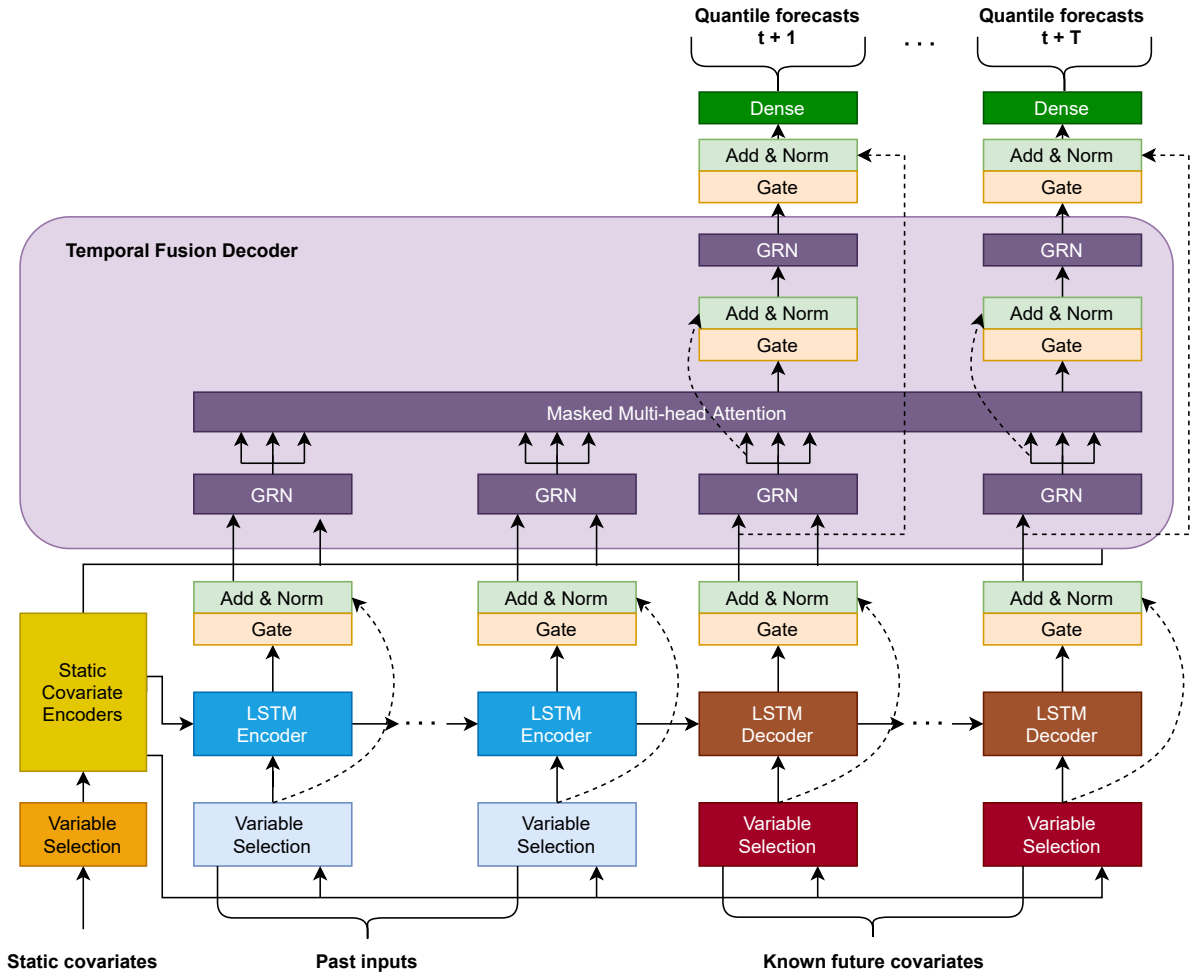


Figure 4.3: Overview of the TFT architecture, recreated from [68]

5 | Experiments

In this chapter, we present details on the experiments we conduct in order to answer the research question. Section 5.1 provides an overview of the datasets employed. Section 5.2 presents the training procedure and hyperparameter tuning for each implemented model. To obtain realistic and accurate results, we closely follow the experimental setups defined in each model's respective papers. Lastly, section 5.3 provides implementation details regarding software, tools used, and hardware setup.

5.1 Datasets and Processing

In order to show inherent differences between performance metrics and models, we select five datasets for experimentation: electricity, traffic, volatility, solar and wind. Each dataset has distinct characteristics and represents important real-world industry applications where forecasting models are rigorously used. Limiting the selection to five datasets allows us to evaluate our results more thoroughly without solely basing the evaluation on aggregated metrics, which is what prior empirical studies tend to do. Furthermore, the datasets have been routinely applied as benchmarks and standards for evaluation in previous research. Therefore, all our results should be comparable to previous research and of relevance for future research within the field. In the following, we describe each dataset and the dataset processing in detail, an overview of which is provided in Table 5.1.

5.1.1 Datasets

Electricity: The Electricity Load Diagrams dataset is collected from the UCI Machine Learning Repository [89] and is commonly used as a benchmark for forecasting models [72, 40, 85, 70, 68]. The dataset consists of 370 time series describing the power consumption of distinct clients from 2011 to 2014. Furthermore, it exhibits high daily seasonality and each time series varies significantly in magnitude, i.e., $y_t \in [0, 1 \times 10^5]$. The data is originally provided in 15-minute intervals and is aggregated to hourly frequency to be consistent with previous research. The aggregated dataset consists of approx. 2.2 million data points.

Traffic: The PEMS-SF dataset is collected from the UCI Machine Learning Repository [89] and is typically used as a benchmark alongside electricity, again see e.g., [72, 40, 85, 70, 68]. The dataset comprises 963 time series describing the occupancy rate along 440 SF Bay Area highways. Each time series represents a sensor measuring lane occupancy $y_t \in [0, 1]$. The dataset features high daily seasonality in addition to peak hour traffic spikes. The data is provided in 10-minute intervals and is, like electricity, aggregated to hourly frequency for a total of approx. 4 million observations.

Table 5.1: Overview of datasets, dataset metadata and static parameters employed for experiments

	Electricity	Traffic	Volatility	Solar	Wind
Number of series	369	963	31	137	28
Total observations	2198k	3997k	151k	7200k	306k
Frequency	Hourly	Hourly	Daily	10 min.	Daily
Seasonal period	24	24	5	144	30
Target domain	$\mathbb{R}_{0\geq}$	$[0, 1]$	$\mathbb{R}$	$\mathbb{R}_{0\geq}$	$[0, 1]$
Training samples	500k	500k	100k	2000k	50k
Input length	168	168	252	240	210
Output length	24	24	5	24	30

Volatility: The volatility dataset is collected from the OMI realized library [90] comprising of daily realized volatility computed from the intraday data of 31 stock indices where each index is treated as a time series. In contrast to the other datasets, observations are only provided for business days (i.e., Monday to Friday excluding holidays). Furthermore, each series has different starting points, starting from between 2000-01-03 and 2012-10-15, and ending at 2021-06-02, for a total of 151k observations. The volatility dataset is noisy with no definite seasonality and contains fewer observations than the previous datasets. It was used in the study by Lim *et al.* [68] to contrast with the strongly seasonal electricity and traffic datasets and show the interpretability advantages of the TFT model. As there is some evidence for a weekly seasonal effect in volatility indices [91], we use seasonal period $m = 5$.

Solar: The solar power dataset is provided by NREL[1] and represents the solar power production records of 137 photovoltaic power stations in Alabama State during the year of 2006. The dataset has been previously used for state-of-the-art forecasting research by, e.g., Lai *et al.* [67] and Li *et al.* [85]. The dataset exhibits daily seasonality and intermittent periods of zero power production during nighttime, adding an interesting dynamic. The phenomenon of sporadic periods of $y_t = 0$ can also be found in sales forecasting which was the task selected for the recent M5 competition [25]. The dataset is provided in 5-minute intervals and data is resampled to 10-minute intervals in accordance with Lai *et al.* [67] for a total of 7.2 million data points.

Wind: The wind power dataset is collected from Kaggle.[2] We use the dataset consisting of the aggregated wind power output of 29 countries between 1986 and 2015, measured as the percentage of the respective countries maximum output, i.e., $y_t \in [0, 1]$. We resample the hourly frequency to daily averages to be consistent with Li *et al.* [85]. Note that we remove the data from Cyprus (CY) as it only comprises $y_t = 0$. This leaves a total of 28 time series, one per country. The wind dataset is extremely noisy with very slight yearly and monthly seasonality. Furthermore, in contrast to the other datasets, wind power is completely independent of known time inputs and will serve as an interesting comparison to the other datasets. The resulting dataset comprises a total of 306k data points. We use a monthly seasonal period, $m = 30$, although a stronger yearly seasonal period can be observed. Yearly seasonality is simply too long to fit an adequate number of periods for the SARIMA model, and we find monthly seasonality to provide more reasonable results for this reason.

[1] https://www.nrel.gov/grid/solar-power-data.html
[2] https://www.kaggle.com/sohier/30-years-of-european-wind-generation

5.1.2 Processing

All datasets are preprocessed by adding date-time information to each time series, i.e., date-time covariates. The date-time covariates will serve several purposes: 1) they are used as known inputs by the DeepAR and TFT model. 2) grouping by different date-time covariates such as hour of day and day of week will be an important evaluation tool, especially for seasonal datasets that are highly dependent on time.

To ensure replication of the results on electricity, traffic and volatility by Lim *et al.* [68], we use their source code made available at the Googe Research GitHub repository[3] to obtain and process the aforementioned datasets. The solar dataset is obtained at the GitHub repository[4] of Lai *et al.* [92]. Lastly, the wind dataset is obtained directly from the Kaggle source. All datasets are processed by adding the following covariates:

1. *Group id*: Each time series in the dataset is represented by a group id.
2. *Time index*: Each series has a time index representing the order of observations. This is either hours, days or 10-minute intervals from start depending on the dataset frequency.
3. *Known inputs*: The electricity, traffic and solar datasets are additionally added hour and day of week variables. To volatility and wind, we add day of week, day of month, week of year and month.

Table 5.1 provides an overview of the datasets, including the number of series, observations in total, frequency, seasonality and target domain. Additionally, it provides dataset specific parameters which we present in the following section on training procedure.

5.2 Training and Forecasting Procedure

This section describes the experimental training and forecasting procedure in detail. Each dataset is split along the time axis into three parts: A training set, validation set, and test set used for model learning, hyperparameter tuning and forecast evaluation, respectively. To keep consistent with previous work, we use the following splits for each dataset.

Following previous work on electricity, we use the period between 2014-01-01 to 2014-08-07 for training, 2014-08-08 to 2014-08-31 for validation and the immediate following week for testing [69, 72, 68]. Similarly, for traffic we use the data prior to 2008-06-15 for training and validation, where the last 10% is used as validation. The week immediately following validation is used for testing [69, 72, 68]. For volatility, data before 2016 is used for training, between 2016 and 2017 for validation and 2017 to 2019-06-28 for testing [68]. Solar is split with training data prior to November 2006, validation data during November and testing data during December in the same year. Lastly, for wind we use data prior to 2014 as training, between 2014 and 2015 as validation and the year of 2015 for testing in accordance with [85].

[3] https://github.com/google-research/google-research/tree/master/tft
[4] https://github.com/laiguokun/multivariate-time-series-data

5.2.1 Fitting the SARIMA Models

We are unable to find adequate details on how previous research has implemented and fitted ARIMA or SARIMA models, despite the large base of papers that compare their models to an ARIMA baseline [59, 40, 85, 68, 93]. Furthermore, we have yet to find a paper releasing their code implementation of the ARIMA baseline (or any other models used as baselines for that matter), as most papers only provide implementations for *their* proposed model. At most, it is revealed that they use the `auto.arima` function from the R library[39]. Therefore, we design a custom framework for fitting the SARIMA to the datasets.

We find the most reasonable approach to be fitting a single SARIMA model for each individual time series in each dataset. SARIMA is a univariate model. Although one can add external regressors, we elect to only fit the model on the dependent variable for consistency with the deep learning models as these are also univariate. The fact that we fit a single model to each series means that the SARIMA models have no opportunity to learn from other time series in the dataset, unlike the deep learning models.

For fitting the models, we use a rolling window approach, fitting the model to a moving window consisting of the most recent data as described in Hyndman and Athanasopoulos [20, Chapter 3]. This is done for three reasons: One, it means the models will only be fitted on the most relevant data. Two, the model will react to any changes in the data distribution as the rolling window moves in time. Three, it significantly reduces the required computational time for selecting and fitting the most appropriate model. Like previous research, we use `auto.arima`, which combines unit root tests and minimization of the AIC and MLE to obtain a parameterized SARIMA model. Note that `auto.arima` will fit a seasonal model provided that we supply a time series data object with specified seasonal period, m. We limit the search to a model of order ARIMA$(5,2,5)(5,2,5)$ and use stepwise search with approximation [39]. The search is performed on the most recent moving window prior to the test period. When forecasting, the moving window is moved forward and the model is updated and refit before forecasting T-steps ahead.[5] When fitting the seasonal component of the model, we use seasonal period m as defined in Table 5.1.

The rolling window approach requires specifying a constant lookback window, k. The ideal solution would be to select the lookback window based on a time series cross-validation approach [20], and therefore select k for each time series individually. However, this would significantly increase the computational resources required considering the size of the datasets. To keep the problem tractable, we performed preliminary experiments and found the following static lookback windows to produce the best results: $k = 240$ for electricity, $k = 720$ for traffic, $k = 756$ for volatility, $k = 240$ for solar, and $k = 300$ for wind. The search was performed in terms of including multiples of the seasonal period m in the lookback window, to ensure that the models are fitted on an adequate number of seasonal periods.

To summarize our approach, for each time series in the dataset, we use `auto.arima` to find the optimal model order and update the model iteratively on the moving window as we produce T-step ahead forecasts for each time step. This approach keeps the problem tractable (considering we have a total of 1528 time series to fit), allowing us to fit a large number of models without having to manually intervene on each fit.

[5] Note that refitting the model during the prediction period is what any practitioner would do in practice. Although this looks like target leakage, it is just a version of time series cross validation performed on the test set and is detrimental to the state-space modeling approach, see [20, Chapter 5.10].

5.2.2 Training the Deep Learning Models

We follow the general approach of previous research when training the deep learning models. Training windows are created and sampled following Salinas *et al.* [72]. Denoting a dataset by $\{\mathbf{y}_{i,1:N}\}_{i=1}^{M}$ where M is the number of different time series and N is number of available observations per time series, training instances are sampled with a fixed lookback window of length k and forecast horizon of length T. Sampled windows then consists of input observations $\mathbf{y}_{i,t-k:t} = \{y_{i,t-k}, \ldots, y_{i,t}\}$ used to forecast the values $\hat{\mathbf{y}}_{i,t:t+T} = \{\hat{y}_{i,t}, \ldots, \hat{y}_{i,t+T}\}$ for different starting times t. The total training data set consists of H sliding windows $\{\mathbf{y}_{i,t-k:t+T}\}_{t\in H, i\in M}$. We ensure that the same selection of samples is used and loaded when training each deep learning model. Hence, each model will see the same training and validation data. In accordance with [72, 68], we use 500k samples for training on electricity and traffic and 100k samples for volatility. In accordance with [85], we use 50k samples on wind. Lastly, there is no predefined scheme for the solar dataset. Hence, we elect to use 2000k samples as the dataset is considerably larger and find this to work well in preliminary experimentation.

Covariates for each model are selected for the DeepAR and TFT model as in previous research [72, 68]. For solar and wind where previous configurations are unavailable, we select similar covariates to that of electricity and traffic while omitting covariates that are logically independent from the target. I.e., wind power output is clearly independent on the day of the week but might be dependent on the month due to yearly seasonality. Hence, for solar we use group id as static covariate and hour of day as known input. For wind we use group id as static covariate and month as known input. The specification for which covariates are employed per dataset is presented in Table 5.2. Lastly, all models are trained with early stopping on the validation loss, with a patience of 5 epochs in accordance with [68].

Table 5.2: Overview of the covariates added to the DeepAR and TFT model for each dataset

Covariates	Electricity	Traffic	Volatility	Solar	Wind
Time series ID	✓	✓	✓	✓	✓
Time from start	✓	✓	✓	-	-
Hour of day	✓	✓	-	✓	-
Day of week	✓	✓	✓	-	-
Day of month	-	-	✓	-	-
Month of year	-	-	✓	-	✓
Week of year	-	-	✓	-	-

5.2.3 Hyperparameter search

We use the same hyperparameters from previous research where this is available. Lim *et al.* [68] provides hyperparameters for the TFT model on traffic, electricity and solar. For DeepAR we use the hyperparameters for electricity and traffic as stated in the original paper [72]. For the other datasets and models where previous research is not available, we use the Optuna hyperparameter optimization framework [94], which is a state-of-the-art optimization library specifically designed for machine learning and neural networks. Table 5.3 shows the selected search space for each hyperparameter and model. Note that we use the same search space for hyperparameters that are common across models for consistency. The search was conducted running 250 trials for each search, using a batch size of 64, a maximum of 25

epochs and pruning callbacks for poor trials [94]. To keep running time tractable, we subset the training and validation sets according to the following fractions, respectively: Electricity, 0.15 and 0.33. Traffic, 0.15 and 0.33. Solar, 0.10 and 0.20. Volatility, 1.00 and 1.00. Wind 1.00 and 1.00. The final hyperparameters for each model and dataset are attached in appendix A.1.

Table 5.3: Overview of search ranges for Optuna hyperparameter optimization

	DeepAR	Seq2Seq	TCN	TFT
Learning rate	$[1 \times 10^{-6}, 1 \times 10^{-2}]$	$[1 \times 10^{-6}, 1 \times 10^{-2}]$	$[1 \times 10^{-6}, 1 \times 10^{-2}]$	$[1 \times 10^{-6}, 1 \times 10^{-2}]$
Num. layers	[2, 6]	[1, 3]	[2, 6]	-
Hydden size	[16, 256]	[16, 256]	[16, 256]	[16, 256]
Dropout	[0.1, 0.9]	[0.1, 0.9]	[0.1, 0.9]	[0.1, 0.9]
Max gradient norm.	$[1 \times 10^{-2}, 1 \times 10^{2}]$	$[1 \times 10^{-2}, 1 \times 10^{2}]$	$[1 \times 10^{-2}, 1 \times 10^{2}]$	$[1 \times 10^{-2}, 1 \times 10^{2}]$
Kernel size	-	-	[4, 16]	-
Attention head size	-	-	-	{1, 4}

5.3 Implementation

This section provides details regarding the tools and libraries employed to implement the models and run experiments. The source code for the implemented models, hyperparameter tuning, proposed performance metrics, running each experiment, and the evaluation framework is attached as a single code base for replication of all reported metrics, experiments, and results.

PyTorch [95] was used to implement the Seq2Seq model and TCN model, the TCN model based on the original implementation by [64]. The DeepAR model and TFT model have been configured and implemented using the PyTorch Forecasting library for time series forecasting with deep learning models, developed by Beitner [96]. For improving the transparency when training models and making it possible to run several different models and experiments within the same framework, we used the PyTorch Lightning library for training and monitoring PyTorch models [97].

The SARIMA model was implemented in the R language [98], utilizing the `auto.arima` function [39]. Although implementations of `auto.arima` exist in Python, we found the R implementation to be significantly faster and is the implementation employed in the literature.

Lastly, all models have been trained on the NTNU IDUN computing cluster [99]. The deep learning models have been tuned and trained utilizing a single Nvidia Tesla V100 GPU. The hyperparameter tuning process required up to approximately 5 days of run time per model, and the training of models up to approximately 12 hours per model. The SARIMA models in R were fitted utilizing a single CPU, requiring run-times of up to 5 days.

6 | Results and Discussion

In this chapter we present and evaluate the experimental results obtained as described in chapter 5. First, as the baseline, we evaluate our results in terms of aggregated error metrics which is the commonly employed approach in the literature. Furthermore, we show how aggregated metrics hide valuable information by the proposed win-loss ranking metric. Second, we analyze the results in terms of the proposed variance weighted error metric and show how differences in the seasonal period can impact the choice of model during evaluation. Third, we evaluate the results in relation to the forecast horizon and discuss the applicability of the proposed delta horizon metric. Lastly, we use the decomposed error metrics and TDI to gain additional insights that relate the errors to the concepts of trend, season, noise, and time.

6.1 Baseline Evaluation

We begin by evaluating results in terms of aggregated errors, in other words, averaging errors across the prediction horizon and across the series in the dataset. This results in a single value per metric for each model and dataset. This approach represents how the performance of models are evaluated and compared in the literature. Table 6.1 presents aggregated errors in terms of the MASE and RMSSE metrics.

Table 6.1: Results in terms of MASE and RMSSE aggregated across the prediction horizon and each time series

	Electricity		Traffic		Volatility		Solar		Wind	
	MASE	RMSSE	MASE	RMSSE	MASE	RMSSE	MASE	RMSSE	MASE	RMSSE
SARIMA	0.289	0.358	0.329	0.463	0.682	0.660	0.361	0.459	0.93	0.878
DeepAR	0.322	0.380	0.227	0.362	0.704	0.678	0.274	0.332	0.817	0.773
Seq2Seq	0.274	0.330	0.177	0.303	0.669	0.647	0.259	0.327	0.826	0.767
TCN	0.252	0.312	0.167	0.298	0.678	0.652	0.318	0.354	0.827	0.769
TFT	0.260	0.330	0.130	0.293	0.689	0.664	0.251	0.345	0.796	0.761

Note: The best and worst scores are marked green and red respectively.

In general, we find that TFT, Seq2Seq, and TCN are the best performing models and consistently perform well on all datasets, obtaining relatively similar scores (within 0.03 points excluding traffic and solar MASE). In contrast, the worst performing model tends to be SARIMA, although the model performs reasonably well on volatility. Furthermore, we find that DeepAR is considerably worse on electricity and volatility but is the second best on solar

(RMSSE) and wind (MASE). The TFT model is considered state-of-the-art and is the best performing model on three out of five datasets in terms of MASE. Surprisingly, we find that less complex models such as Seq2Seq and TCN outperforms the TFT model on datasets where it should have an inherent advantage. Recall from chapter 4 that both the TFT and DeepAR models use known covariates such as hour of day, day of week and the ID of the time series to improve forecasts. This should add considerable value in terms of performance on the electricity dataset, where time of day is crucial to predicting power consumption spikes.[1] Yet, Seq2Seq and TCN yields better scores without covariates or information about which time series they are predicting on. Furthermore, we would expect the covariates to be of little value on the wind dataset, as wind output should be relatively independent on day and month (although a general yearly seasonality is present). Still, the TFT model obtains the best score on wind.

Another interesting finding is the considerable difference between MASE and RMSSE, most notably on the traffic and solar datasets. As was discussed in section 4.1, MASE favors forecasts towards the median in contrast to RMSSE which favors forecasts towards the mean [73, 74]. The solar dataset is skewed towards 0 as the sun shines only during daytime, leaving the nightly periods (i.e., roughly half of the dataset), with 0 in power output. Thus, MASE will assign lower values to models that produce forecasts closer to 0. On solar, we observe that TFT performs best in terms of MASE and Seq2Seq in terms of RMSSE. This highlights the importance of selecting an appropriate error metric. For solar it seems reasonable to use RMSSE, as forecasting the mean (i.e., higher values and actual power production in this case), should be favored rather than correctly forecasting that there will be little to no sun during the night. We observe a similar effect on traffic, where TFT is 0.037 points better than TCN by MASE, and only 0.005 points better by RMSSE.

As should be apparent, the evaluation using aggregated metrics provides an indication of which models perform better *on average*. However, aggregated metrics yield little insight on *where* the models are outperforming. Table 6.2 presents results using the win-loss ranking metric, showing the number of times a model is the best (or worst) performing model per time series in the dataset. In other words, errors are not averaged across all time series, but rather evaluated per series while counting wins and losses for each model. First, we observe that the TFT model is the most frequent winner on electricity, although the TCN model scored best in terms of aggregated errors. This difference could stem from a difference in losses, where TCN is worst on only 3 series, in contrast to TFT being worst on 17 (22) series when measured by MASE (RMSSE). Another possibility is that the TCN model performs well on average on the majority of series without tending towards the extreme predictions, as it has fewer wins, but also fewer losses, compared to TFT. This could be a desirable property, because typically it is preferable to avoid major forecasting errors and continually perform reasonably, rather than providing both extremely accurate and inaccurate forecasts. Another interesting finding is the win-loss ranks for the SARIMA model. On electricity, SARIMA is frequently both the best and the worst model (80 wins and 97 losses in terms of MASE). This fact is entirely neglected by the aggregated metrics, in which SARIMA seems to be a bad model. However, as the win-loss metric reveals, it is in fact the best model on 22% of the series in the dataset. The metric also reveals SARIMA to be the worst model on nearly all traffic series. We suspect this to be due to SARIMA only picking up the general seasonal pattern of the series, unable to properly fit rush hour traffic spikes.

[1] TFT was also fitted with an additional input variable on the volatility dataset to be consistent with the experiments of Lim *et al.* [68].

Table 6.2: Results in terms of the win-loss ranking metric calculated based on MASE and RMSSE for each time series

	Electricity		Traffic		Volatility		Solar		Wind	
	Wins	Losses	Wins	Losses	Wins	Losses	Wins	Losses	Wins	Losses
SARIMA										
- MASE	80	97	2	853	1	1	0	94	0	28
- RMSSE	67	109	3	850	1	3	0	127	0	28
DeepAR										
- MASE	21	209	27	101	4	19	49	1	1	0
- RMSSE	25	192	49	103	2	20	79	1	6	0
S2S										
- MASE	13	43	5	3	17	0	12	0	1	0
- RMSSE	24	43	107	1	22	0	39	0	8	0
TCN										
- MASE	109	3	22	5	3	5	0	42	3	0
- RMSSE	110	3	131	5	4	1	0	4	3	0
TFT										
- MASE	146	17	907	1	6	6	76	0	23	0
- RMSSE	143	22	673	4	2	7	19	5	11	0

Note: The best and worst scores are marked green and red respectively.

Lastly, we see the same discrepancy between evaluations using MASE and RMSSE on the solar dataset. Here, TFT is the best model on 76 series when measured by MASE and only 19 when measured by RMSSE. This was also reflected in the aggregated metrics. We elect to continue the evaluation in the following sections using RMSSE to not complicate the evaluation and the number of metrics. Furthermore, it seems reasonable to prefer forecasts towards the mean in cases such as the solar dataset, as was argued in [25].

6.2 Weighting Errors by Variance

We have hypothesized that aggregating metrics across the prediction horizon and time series can potentially make models that are good at predicting what is already known appear to be better than models that are good at predicting uncertain events. This is especially important for time series with characteristics rooted in real-world phenomena, such as power consumption, traffic flows and solar power generation, which also are important industrial applications. To account for such differences in variance over the seasonal period, we analyze the results in terms of the proposed variance weighted errors.

Firstly, we focus on the first row of Figure 6.1. It shows plots of the RMSSE metric aggregated per hour for the electricity, traffic and solar datasets, respectively.[2] Unsurprisingly, we observe that there are significant differences in mean error depending on what time of the day is forecasted. Power consumption is difficult to predict during the day and evening, traffic is difficult to predict at 07:00 AM and 16:00PM during rush hour, and solar power generation is zero during night. Interestingly, Figure 6.1 shows that, on average, models are very consistent and follow the same error pattern. A model with lower error score tends to

[2]Plots for variance weighted RMSSE on volatility and wind are attached in appendix A.2.

be better at forecasting across all hours. In other words, we do not find that one model is very good at predicting peak time traffic while another model is better for baseline traffic. This, however, could be due to the law of large numbers and averaging across all series. As we found in the previous section, although one model could be best in terms of aggregated errors, another model can still have significantly higher win rank with lower average error. Thus, we do not know whether there are significant differences on each time series prior to aggregation. We do, however, find one significant difference on the solar dataset. That is, DeepAR is the worst model at predicting nighttime power production which we already know is 0, and thus irrelevant. At the same time, DeepAR is the best model when predicting solar generation during peak hour. In other words, the model is punished for making bad predictions for something we already know, outweighing the improvements it makes during critical hours. A simple solution would be to just disregard the periods when the sun is down, however, this would not be as easily applicable to datasets such as electricity or traffic.

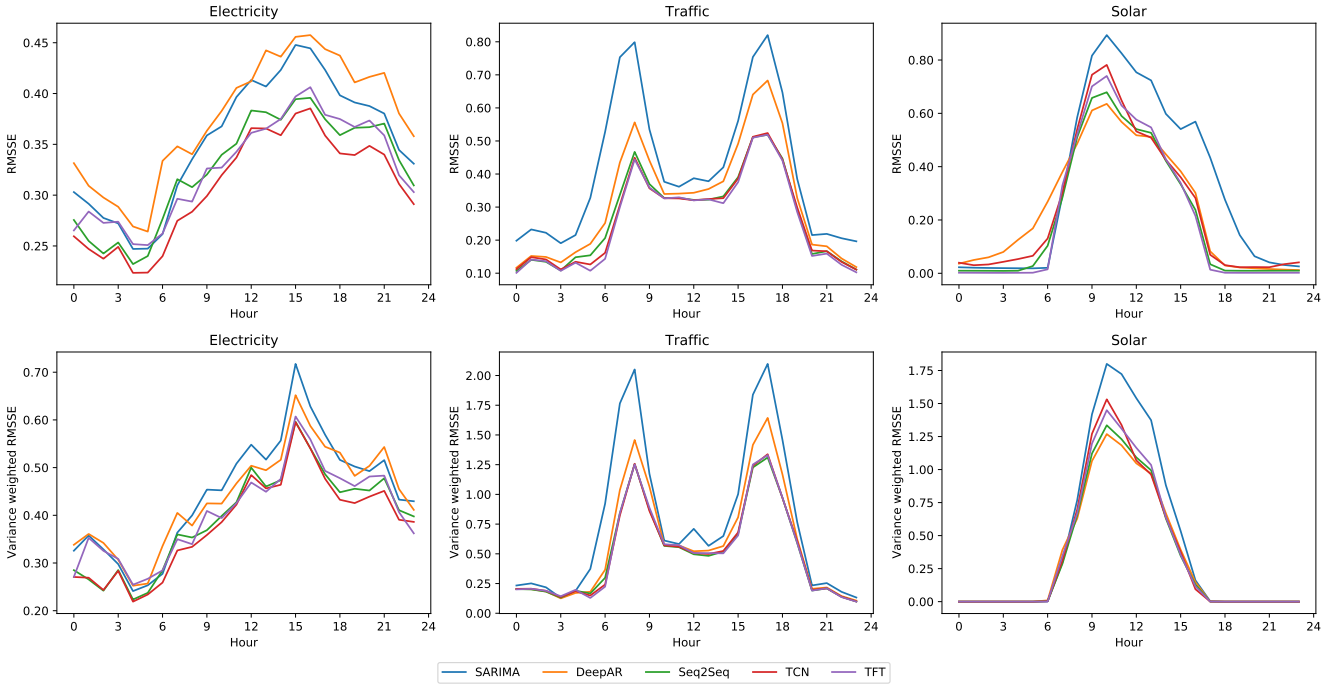


Figure 6.1: Results in terms of RMSSE (first row) and variance weighted RMSSE (second row) computed per hour of day for the electricity, traffic and solar datasets

To account for the differences in uncertainty we proposed variance weighted erros, weighting errors in relation to how much variance there is at the predicted hour. The second row of Figure 6.1 plots RMSSE_{VM} for each prediction hour. Most notably, we see that errors on the solar dataset between 18:00 PM and 06:00 AM have been scaled to 0 as this period has no variance in power output. Furthermore, the error during peak output has increased significantly in magnitude, as more weight is put on the period where power output is highly variable. The same result is visible, although not as pronounced, for electricity and traffic. Differences between models where the target is relatively predictable (i.e., low target variance) have been reduced, whereas more weight is put on periods where the target is unpredictable, i.e., during peak traffic and electricity hours.

Table 6.3 shows RMSSE_{VM} measure for all datasets in addition to RMSSE as previously presented in Table 6.1 for comparison. Note that electricity, traffic and solar is weighted by

the variance at each hour, while volatility and wind is weighted by the variance at each day of week due to having daily frequency. As expected, we observe significant increases in error magnitude for electricity, traffic and solar (up to 28.8%, 119% and 77.4% respectively). The datasets are highly seasonal and partly driven by known phenomena. In contrast wind and volatility shows only a marginal increase in magnitude (up to 1.11% and 0.30% respectively), as their variance is close to constant and independent of the seasonal period.[3] As a consequence, we find that the best models change from TFT to Seq2Seq on traffic and from Seq2Seq to DeepAR on solar, which agrees with our discussion of DeepAR in Figure 6.1.

Table 6.3: Results in terms of aggregated RMSSE and RMSSE_{VM} for each model and dataset

	Electricity		Traffic		Volatility		Solar		Wind	
	RMSSE	RMSSE_{VW}	RMSSE	RMSSE_{VW}	RMSSE	RMSSE_{VW}	RMSSE	RMSSE_{VW}	RMSSE	RMSSE_{VW}
SARIMA	0.358	0.461	0.463	0.990	0.739	0.745	0.459	0.779	0.878	0.880
DeepAR	0.380	0.450	0.362	0.748	0.759	0.767	0.332	0.560	0.773	0.775
Seq2Seq	0.330	0.406	0.303	0.638	0.724	0.732	0.327	0.576	0.767	0.769
TCN	0.312	0.394	0.298	0.642	0.730	0.737	0.354	0.618	0.769	0.770
TFT	0.330	0.415	0.293	0.642	0.744	0.751	0.345	0.612	0.761	0.762

Note: The best and worst scores are marked green and red respectively.

In addition to providing a more realistic representation of the forecast error, we argue that RMSSE_{VM} more accurately represents performance in relation to the naive forecasting method. Scaled errors have the benefit that they can be interpreted as being better than the reference method when the error is less than one and worse when greater than one [17]. However, for multi-step forecasting, the naive multi-step forecast will likely produce very bad forecasts for periods which are easy to forecast, e.g., baseline traffic or before sunrise and after sunset. Thus, when critical periods are weighted higher, the metric more closely represents the improvement over the reference method as models no longer have the benefit of major metric improvements when forecasting uninteresting periods. In the next section, we examine differences in how the models perform across the prediction horizon.

6.3 Evaluation Over the Forecast Horizon

When evaluating multi-step forecasts, an important aspect that is not considered explicitly when looking at aggregated metrics is the forecast horizon. In this section, we evaluate our results in terms of RMSSE over the forecast horizon and the delta horizon metric, Δ_H, proposed in chapter 4.

Figure 6.2 shows RMSSE values per prediction horizon $\tau \in \{1, \dots, T\}$ for each dataset. Like we observed in Figure 6.1, models trained on the same dataset tend to follow the same pattern in error distribution. Models that perform better in terms of aggregated error also perform better across the prediction horizon compared to a model with higher errors. We find little evidence of individual differences. Differences are most noticeable for electricity, where TFT is the worst model at prediction time $t = 0$, but better relative to other models later in the prediction horizon. Conversely, SARIMA is second best at time $t = 0$, but second worst

[3] One could argue that volatility is slightly seasonal in that the first and last day of the business week are more volatile as they constitute the opening and closing of stock exchanges around the weekend. Thus, variance weighting impacts the volatility dataset slightly more than wind.

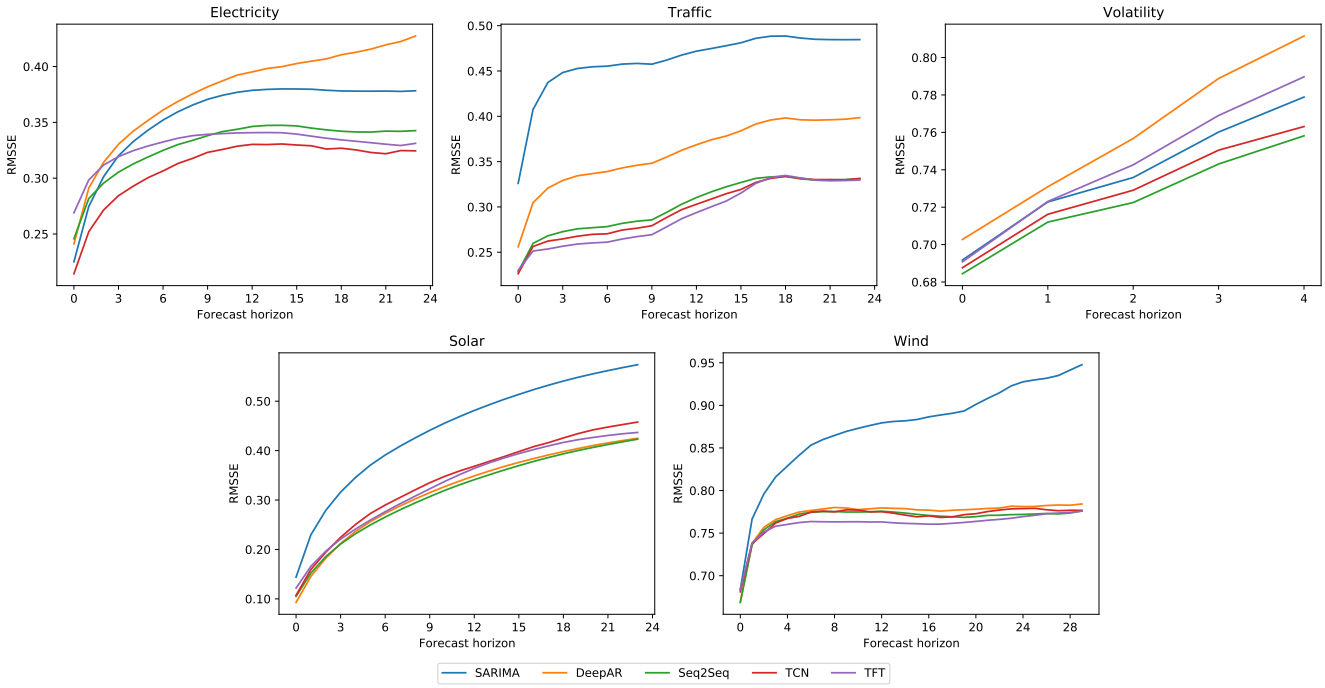


Figure 6.2: Results in terms of RMSSE computed per forecast horizon for all datasets

after a few time steps. In terms of datasets, models on electricity and solar exhibit a nearly concave increase in error across the horizon. For volatility, the increase in error over the horizon appears to be close to linear, which is what we should expect for a highly noisy dataset. Interestingly, on the wind dataset, models are no worse at predicting wind 4 days in advance than 30 days in advance. This could indicate that, as long as the models derive a reasonable prediction for the average wind level a few days in advance, the same prediction will yield equally good results for the next 30 days as well. This, in combination with the law of large numbers, is likely the source of this effect. What this shows is that, at the very least on an aggregated level, models tend to keep their performance advantages across the entire forecast horizon. The exception being during the first few time steps.[4]

Despite largely similar and indifferent behavior in terms of errors across the forecast horizon, we find large differences in the Δ_H metric. Table 6.4 shows the mean Δ_H metric for each model and dataset. Recall that Δ_H measures how much the model adjusts its estimate of y_t as t approaches over the forecast horizon. First, we find that although TCN and TFT perform similarly in terms of aggregated RMSSE (0.312 and 0.330, respectively), they are significantly different in terms of Δ_H (12.8274 and 4.9080, respectively). In other words, TCN is highly sensitive over the horizon and frequently updates its initial estimates. In contrast, TFT is more stable and follows the initial forecasted value more closely over the horizon, effectively making less adjustments. It is difficult to decide which behavior is favorable. For example, one could argue that having more stable forecasts is favorable, as it implies that the model will make less abrupt changes to its initial forecast. However, it could also imply that it is less reactive to new information. Reactivity to new information is likely favorable in situations where arriving information drives the time series (e.g., volatility forecasting which

[4] If the first few time steps of a multi-step forecast are of paramount importance to a practitioner, that would beg to question why a multi-step model was implemented in the first place.

is highly impacted by newsfeeds). Therefore, a lower Δ_H value could be favorable for situations where a stable forecasted trajectory is of higher importance than the individual forecasts, e.g., visualizing trends. In contrast, higher Δ_H and more reactivity could be useful in applications such as stock market predictions.

Table 6.4: Results for the Δ_H metric for each model and dataset

	Electricity	Traffic	Volatility	Solar	Wind
SARIMA	5.9976	0.0009	0.1168	0.3890	0.0062
DeepAR	11.0895	0.0022	0.1036	0.3360	0.0058
Seq2Seq	10.4219	0.0009	0.1070	0.3104	0.0030
TCN	12.8274	0.0023	0.0863	0.3170	0.0044
TFT	4.9080	0.0003	0.1182	0.2434	0.0020

Note: The highest and lowest scores are marked green and red, respectively.

Lastly, instead of reporting the aggregate Δ_H metric, we can plot the Δ_H metric across the prediction horizon to show at which points each model makes the most adjustments (on average). Figure 6.3 shows the Δ_H metric plotted across the forecast horizon. We observe that there are significant differences as to how and where the models update their forecasts. We note that Δ_H of the SARIMA model produces a strictly increasing function over the horizon, which makes sense as it is an iterative multi-step model. On the other hand, direct deep learning models occasionally update forecasts more at the beginning and end of the horizon, as seen on electricity and traffic.

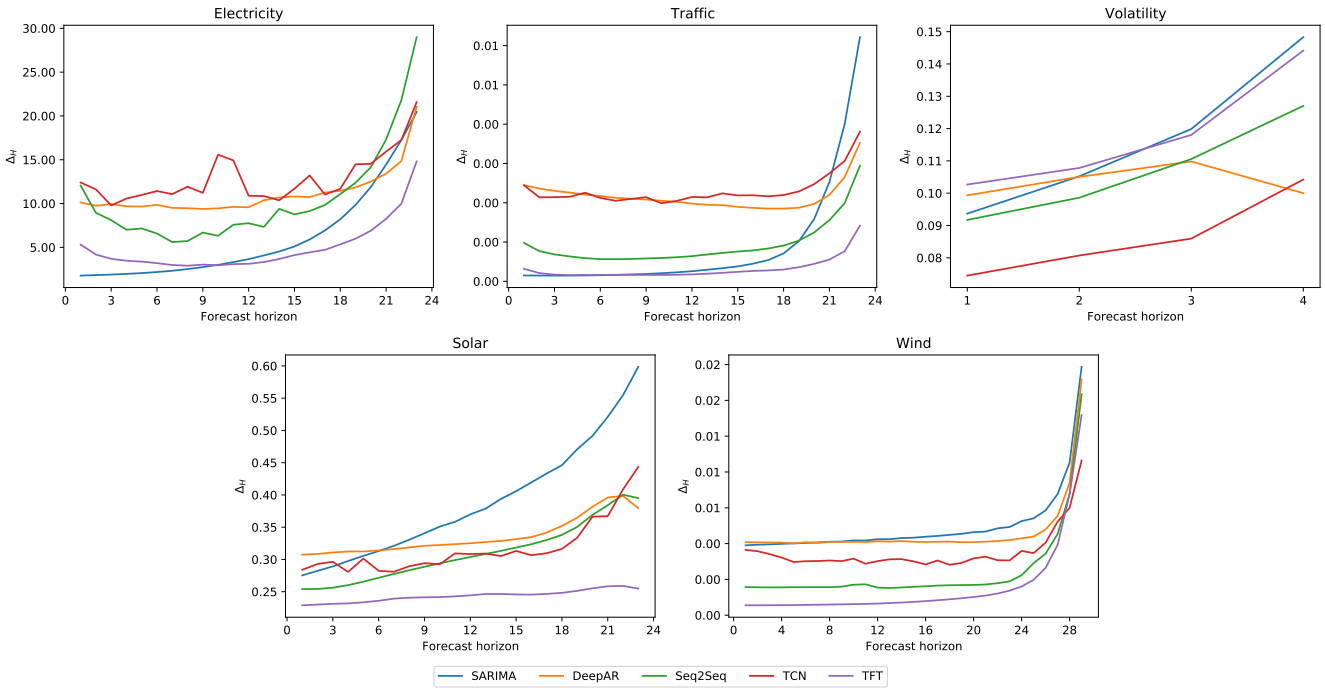


Figure 6.3: Results in terms of the delta horizon metric computed per forecast horizon for all datasets

6.4 Error Decomposition

Lastly, we present results in terms of the decomposed error metrics as proposed in chapter 4. Table 6.5 shows decomposed errors in terms of trend, season and remainder for the path the T-step-ahead forecasts), p_T, and for the path the 1-step-ahead forecasts, p_1. We will refer to the remainder component as noise, although it can contain other elements than white noise as it comprises everything that is not captured by the trend and seasonal components.

Table 6.5: Results in terms of decomposed errors for p_T and p_1

	Electricity		Traffic		Volatility		Solar		Wind	
	p_T	p_1	p_T	p_1	p_T	p_1	p_T	p_1	p_T	p_1
SARIMA										
- Trend	0.241	0.049	0.210	0.079	0.511	0.254	0.168	0.027	0.589	0.112
- Season	0.179	0.141	0.276	0.192	0.249	0.333	0.312	0.062	0.355	0.363
- Noise	0.190	0.154	0.258	0.218	0.440	0.474	0.440	0.124	0.634	0.561
DeepAR										
- Trend	0.321	0.059	0.156	0.059	0.554	0.331	0.153	0.010	0.281	0.119
- Season	0.211	0.185	0.225	0.171	0.257	0.310	0.318	0.059	0.400	0.358
- Noise	0.163	0.130	0.171	0.144	0.433	0.450	0.219	0.070	0.587	0.542
Seq2Seq										
- Trend	0.214	0.062	0.111	0.046	0.482	0.263	0.129	0.013	0.277	0.103
- Season	0.183	0.181	0.155	0.144	0.253	0.330	0.314	0.074	0.391	0.360
- Noise	0.175	0.138	0.162	0.142	0.438	0.460	0.248	0.073	0.589	0.543
TCN										
- Trend	0.205	0.049	0.105	0.042	0.490	0.291	0.132	0.012	0.263	0.113
- Season	0.157	0.141	0.145	0.135	0.262	0.320	0.369	0.074	0.385	0.369
- Noise	0.174	0.141	0.169	0.148	0.43	0.453	0.234	0.075	0.599	0.549
TFT										
- Trend	0.208	0.126	0.101	0.062	0.519	0.276	0.127	0.020	0.239	0.185
- Season	0.175	0.165	0.146	0.138	0.252	0.327	0.326	0.085	0.404	0.36
- Noise	0.167	0.144	0.154	0.141	0.445	0.465	0.254	0.084	0.587	0.541

Note: Noise refers to the remainder error as presented in chapter 4.

First, we observe that in general the trend error decreases over the forecast horizon (from p_T to p_1). Furthermore, we observe a decrease in the season error and noise error, however, the decrease is substantially lower than what we observe in the trend error. In other words, this indicates that the improvements in accuracy as the forecast horizon decreases stems primarily from improvements when predicting the correct trend. The improvements when predicting seasonality and noise tends to be minor in comparison. If we compare results on electricity and traffic (seasonal data) to volatility and wind (noisy data), we observe that the noise error is significantly higher relative to trend and season on the noisy data. Conversely, on seasonal data, the error of trend and season is higher relative to noise (when observing the p_T values). Thus, the error metrics indicate which components of each dataset are difficult to predict far ahead in time. On seasonal data, this tends to be the long-term trend, while season and noise tend to be as difficult to predict tomorrow as it is in the future. The same

is true for the noisy data, however, the trend component is smaller relative to the noise component. We observe that the noise component in general rarely decreases over the horizon, which is to be expected as true noise in theory is unpredictable.

One considerable disadvantage of the decomposed errors is interpretability between models. A prominent example of this is that TFT on electricity produces trend error 0.126, which is worse relative to the other models between 0.062 and 0.049. Yet, TFT is the second best model in terms of aggregated RMSSE. In other words, TFT likely performs better during the other time steps between the first (step T) and last (step 1) forecasted value, which is not captured by the metrics. A solution could be to include the average of all steps in addition to the first and last steps, however, this also increases the number of metrics.

6.5 Temporal Distortion

In addition to error metrics, we evaluate the forecasts in terms of the TDI and TDM. Table 6.6 shows the mean TDI and TDM for each time series and dataset. A low (high) TDI indicates that the forecast is aligned (unaligned) with the ground truth. A negative (positive) TDM indicates an in advance (late) forecast compared to ground truth.

We observe that for electricity, traffic and volatility, the best (worst) performing model in terms of TDI corresponds with the best (worst) model in terms of RMSSE and MASE (see Table 6.1). This is, however, not the case for solar and wind. The TFT model seems to perform well on solar, electricity and traffic, and worse on volatility and wind. This makes sense, as the TFT model has access to date time covariates that should be useful when accurately predicting events and shifts. This is especially visible on the solar dataset, where the identification of sunrise and sunset when power production ramps up. This is also where we assume date-time information to be particularly useful. The TFT temporal alignment is worse on volatility and wind, which are noisy datasets where covariates likely are irrelevant.

Table 6.6: Results in terms of mean TDI and mean TDM computed for every model and dataset

	Electricity		Traffic		Volatility		Solar		Wind	
	TDI	TDM	TDI	TDM	TDI	TDM	TDI	TDM	TDI	TDM
SARIMA	0.100	-0.003	0.108	0.052	0.175	-0.077	0.119	0.045	0.401	-0.251
DeepAR	0.110	0.076	0.068	-0.005	0.195	-0.042	0.133	0.094	0.351	0.001
Seq2Seq	0.100	-0.030	0.055	-0.080	0.169	-0.034	0.129	0.127	0.37	-0.223
TCN	0.087	0.050	0.052	0.099	0.185	-0.054	0.156	0.129	0.384	-0.006
TFT	0.089	-0.054	0.038	0.179	0.187	-0.060	0.110	0.281	0.409	-0.094

Note: The best and worst scores are marked green and red respectively.

In general, we note that aggregating the TDI and TDM metric has inherent problems. During evaluation, we find that the TDI and TDM applied to individual forecasts from the same model can vary significantly, especially in the TDM. We see this in distributional plots, an example of which is attached in appendix A.3, the distribution the TDI and TDM metrics on the traffic dataset.[5] Therefore, the aggregated metrics will only provide a measure of the general tendency for the temporal alignment of the model forecasts. One could, of course, also argue that this is the case when computing aggregated error metrics.

[5] Distributional plots from the other models and datasets show similar patterns.

7 | Conclusion and Future Work

In this thesis we investigate ways to provide additional insights when evaluating multi-step point forecasting models. Having identified that the current literature on performance metrics mainly focus on evaluating models in terms of aggregated metrics, we hypothesize that this neglects valuable information in the evaluation process. We test our hypothesis by conducting empirical experiments and propose four novel metrics to provide insights in the evaluation process that would be valuable to practitioners and industry.

First, we evaluate our results in terms of aggregated MASE and RMSSE. Our results show that the best model depends on the chosen performance metric, supporting the findings of the current literature [18]. Furthermore, we evaluate the results in terms of the proposed win-loss ranking metric and show that important information is lost when errors are aggregated. For example, the TCN model is deemed best on electricity in terms of MASE (0.252) and is the best model on 109 series. In contrast, TFT is second best in terms of MASE (0.260), yet it is the best model on 146 of the series in the dataset. This information would be important in industry applications, as it does not make sense to apply TCN on *all* the series even though it is the best model in terms of MASE. Furthermore, one would probably not consider SARIMA to be that good relative to TCN and TFT (MASE of 0.289). However, SARIMA is in fact the best model on 80 of the series. Thus, the best model is highly dependent on the individual time series.

Second, we find that variance weighted errors can be useful for datasets where there are large variations over the seasonal period and where performance during uncertain periods should be weighted higher. The applicability of the metric is especially visible on solar and traffic, where Seq2Seq and DeepAR are the best models in terms of RMSSE_{VW} due to better performance where variance is high. However, we also find that the difference between models over the seasonal period tends to be minor and that models in general follow the same error pattern. In other words, we do not find that some models perform especially well during uncertain periods. Thus, the metric might only be applicable for special cases, limiting the usefulness of the metric.

Third, as with the seasonal period, we find that models in general follow the same error pattern over the forecast horizon. If model X is better than model Y on average, the same conclusion tends to hold for all forecast horizons. Thus, a metric that measures performance over the forecast horizon does not seem to be useful. We do, however, find significant differences in how much each model updates its forecasted value for a point in time over the horizon. The proposed delta horizon metric allows practitioners to differentiate between models that produce stable forecasts over the forecast horizon and models that react more aggressively to arriving information.

Lastly, we provide additional insights into how the errors relate to trend, season, noise, and time alignment. The decomposed error metrics indicate that improved performance

over the forecast horizon is largely due to improvements in forecasting the correct trend. This is true for both seasonal and noisy datasets, however, we find the noise component to account for a larger portion of the error on noisy datasets, which agrees with intuition. Relating errors to time alignment by the aggregated TDI and TDM metrics, we find that the TDI and RMSSE agree on the best model for three of the five datasets. However, as time alignment is an elusive concept and more difficult to grasp than simple errors, we find the applicability of this metric to be somewhat limited, especially on an aggregated level.

In what follows, we provide some interesting directions for future work. First is to further develop the proposed metrics. For example, variations of the win-loss ranking metric could be developed to show the average rank or average quantile a model places for each time series. It is also possible to improve the interpretability of the decomposed errors and delta horizon metric, as they currently are only meaningful in comparison to the metrics of other models. Thus, providing upper and lower bounds for these metrics or incorporating relative scaling could increase the interpretability and be promising directions for future research.

Second, as this empirical study was conducted with deep learning models, the size of the selected training and testing sets are rather large due to the models requiring a lot of data to be trained effectively. It would be interesting to perform the same experiments and test the applicability of metrics on shorter time series and test series. With fewer observations, the law of large numbers might not be as prevalent considering that we have observed very smooth error patterns over the seasonal period and forecast horizon.

Similar to running the experiments on shorter datasets, it would be interesting to test the impact of using other types of models. As we have observed, the deep learning models produce largely the same error patterns. Therefore, training the deep learning models with other loss functions could provide interesting differences, e.g., the DILATE loss function which explicitly differentiates between shape and time [100]. Furthermore, testing whether our conclusions hold for other state space models and forecasting techniques would be a possible extension of this work.

Lastly, it would be interesting to analytically investigate why the models tend to follow the same general error patterns across both seasonality and forecast horizon. The results show that models just get better overall rather than improving on specific seasonal periods or forecast horizons. We have mentioned the law of large numbers but have not yet seen this investigated in the literature. It would be an interesting subject for further research.

References

[1] J. G. De Gooijer and R. J. Hyndman, '25 years of time series forecasting,' *International Journal of Forecasting*, vol. 22, no. 3, pp. 443–473, 2006. DOI: 10.1016/j.ijforecast.2006.01.001.

[2] M. Lei, L. Shiyan, J. Chuanwen *et al.*, 'A review on the forecasting of wind speed and generated power,' *Renewable and Sustainable Energy Reviews*, vol. 13, no. 4, pp. 915–920, 2009. DOI: 10.1016/j.rser.2008.02.002.

[3] R. H. Inman, H. T. Pedro and C. F. Coimbra, 'Solar forecasting methods for renewable energy integration,' *Progress in Energy and Combustion Science*, vol. 39, no. 6, pp. 535–576, 2013. DOI: 10.1016/j.pecs.2013.06.002.

[4] E. I. Vlahogianni, M. G. Karlaftis and J. C. Golias, 'Short-term traffic forecasting: Where we are and where we're going,' *Transportation Research Part C: Emerging Technologies*, vol. 43, pp. 3–19, 2014. DOI: 10.1016/j.trc.2014.01.005.

[5] J.-H. Böse, V. Flunkert, J. Gasthaus *et al.*, 'Probabilistic demand forecasting at scale,' *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1694–1705, 2017. DOI: 10.14778/3137765.3137775.

[6] M. Mudelsee, 'Trend analysis of climate time series: A review of methods,' *Earth-Science Reviews*, vol. 190, no. December 2018, pp. 310–322, 2019. DOI: 10.1016/j.earscirev.2018.12.005.

[7] E. J. Topol, 'High-performance medicine: the convergence of human and artificial intelligence,' *Nature Medicine*, vol. 25, no. 1, pp. 44–56, 2019. DOI: 10.1038/s41591-018-0300-7.

[8] O. Bustos and A. Pomares-Quimbaya, 'Stock market movement forecast: A Systematic review,' *Expert Systems with Applications*, vol. 156, 2020. DOI: 10.1016/j.eswa.2020.113464.

[9] S. B. Taieb and A. F. Atiya, 'A Bias and Variance Analysis for Multistep-Ahead Time Series Forecasting,' *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 1, pp. 62–76, 2016. DOI: 10.1109/TNNLS.2015.2411629.

[10] B. Lim and S. Zohren, 'Time series forecasting with deep learning: A survey,' *arXiv*, 2020.

[11] R. Fildes and K. Ord, 'Forecasting Competitions: Their Role in Improving Forecasting Practice and Research,' in *A Companion to Economic Forecasting*, John Wiley & Sons, Ltd, 2004, ch. 15, pp. 322–353. DOI: https://doi.org/10.1002/9780470996430.ch15.

[12] S. Makridakis and M. Hibon, 'The M3-Competition: results, conclusions and implications,' *International Journal of Forecasting*, vol. 16, no. 4, pp. 451–476, 2000. DOI: 10.1016/S0169-2070(00)00057-1.

[13] M. P. Clements and D. F. Hendry, 'Explaining the Results of the M3 Forecasting Competition,' *International Journal of Forecasting*, vol. 17, no. January 2001, pp. 550–554, 2001.

[14] S. F. Crone, M. Hibon and K. Nikolopoulos, 'Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction,' *International Journal of Forecasting*, vol. 27, no. 3, pp. 635–660, 2011. DOI: 10.1016/j.ijforecast.2011.04.001.

[15] S. Makridakis, E. Spiliotis and V. Assimakopoulos, 'Statistical and Machine Learning forecasting methods: Concerns and ways forward,' *PLOS ONE*, vol. 13, no. 3, 2018. DOI: 10.1371/journal.pone.0194889.

[16] R. Fildes, M. Hibon, S. Makridakis *et al.*, 'Generalising about univariate forecasting methods: further empirical evidence,' *International Journal of Forecasting*, vol. 14, no. 3, pp. 339–358, 1998. DOI: 10.1016/S0169-2070(98)00009-0.

[17] 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.

[18] S. Kolassa, 'Why the "best" point forecast depends on the error or accuracy measure,' *International Journal of Forecasting*, vol. 36, no. 1, pp. 208–211, 2020. DOI: 10.1016/j.ijforecast.2019.02.017.

[19] R. J. Hyndman, 'A brief history of forecasting competitions,' *International Journal of Forecasting*, vol. 36, no. 1, pp. 7–14, 2020. DOI: 10.1016/j.ijforecast.2019.03.015.

[20] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, 3rd. Australia: OTexts, 2021.

[21] R. B. Cleveland, W. S. Cleveland, J. E. McRae *et al.*, 'STL: A Seasonal-Trend Decomposition Procedure Based on Loess,' *Journal of Official Statistics*, vol. 6, pp. 3–73, 1990.

[22] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to Sequence Learning with Neural Networks,' *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 3104–3112, 2014.

[23] J. Armstrong and F. Collopy, 'Error measures for generalizing about forecasting methods: Empirical comparisons,' *International Journal of Forecasting*, vol. 8, no. 1, pp. 69–80, 1992. DOI: 10.1016/0169-2070(92)90008-W.

[24] J. S. Armstrong, 'Evaluating forecasting methods,' in *Principles of forecasting*, Springer, 2001, pp. 443–472.

[25] S. Makridakis, E. Spiliotis and V. Assimakopoulos, 'The M5 accuracy competition: Results, findings and conclusions,' *International Journal of Forecasting*, 2020.

[26] S. Makridakis, A. Andersen, R. Carbone *et al.*, 'The accuracy of extrapolation (time series) methods: Results of a forecasting competition,' *Journal of Forecasting*, vol. 1, no. 2, pp. 111–153, 1982. DOI: 10.1002/for.3980010202.

[27] C. Chatfield, 'Apples, oranges and mean square error,' *International Journal of Forecasting*, vol. 4, no. 4, pp. 515–518, 1988. DOI: 10.1016/0169-2070(88)90127-6.

[28] C. Chen, J. Twycross and J. M. Garibaldi, 'A new accuracy measure based on bounded relative error for time series forecasting,' *PLOS ONE*, vol. 12, no. 3, 2017. DOI: 10.1371/journal.pone.0174202.

[29] H. Theil, *Applied economic forecasting*. North-Holland Pub. Co., 1971.

[30] S. Ben Taieb, G. Bontempi, A. F. Atiya *et al.*, 'A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition,' *Expert Systems with Applications*, vol. 39, no. 8, pp. 7067–7083, 2012. DOI: 10.1016/j.eswa.2012.01.039.

[31] A. R. S. Parmezan, V. M. Souza and G. E. Batista, 'Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model,' *Information Sciences*, vol. 484, pp. 302–337, 2019. DOI: 10.1016/j.ins.2019.01.076.

[32] J. Zhang, A. Florita, B.-M. Hodge *et al.*, 'A suite of metrics for assessing the performance of solar power forecasting,' *Solar Energy*, vol. 111, pp. 157–175, 2015. DOI: 10.1016/j.solener.2014.10.016.

[33] L. Vallance, B. Charbonnier, N. Paul *et al.*, 'Towards a standardized procedure to assess solar forecast accuracy: A new ramp and time alignment metric,' *Solar Energy*, vol. 150, pp. 408–422, 2017. DOI: 10.1016/j.solener.2017.04.064.

[34] J. A. Ward, P. Lukowicz and H. W. Gellersen, 'Performance metrics for activity recognition,' *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 1, pp. 1–23, 2011. DOI: 10.1145/1889681.1889687.

[35] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods: Second Edition*, ser. Oxford Statistical Science Series. OUP Oxford, 2012.

[36] G. E. P. Box and G. M. Jenkins, 'Some Recent Advances in Forecasting and Control,' *Applied Statistics*, vol. 17, no. 2, p. 91, 1968. DOI: 10.2307/2985674.

[37] R. Hyndman, A. B. Koehler, J. K. Ord *et al.*, *Forecasting with Exponential Smoothing: The State Space Approach*, ser. Springer Series in Statistics. Springer Berlin Heidelberg, 2008.

[38] D. Kwiatkowski, P. C. Phillips, P. Schmidt *et al.*, 'Testing the null hypothesis of stationarity against the alternative of a unit root,' *Journal of Econometrics*, vol. 54, no. 1-3, pp. 159–178, 1992. DOI: 10.1016/0304-4076(92)90104-Y.

[39] R. J. Hyndman and Y. Khandakar, 'Automatic Time Series Forecasting: The forecast Package for R,' *Journal of Statistical Software*, vol. 27, no. 3, p. 22, 2008. DOI: 10.18637/jss.v027.i03.

[40] S. S. Rangapuram, M. Seeger, J. Gasthaus *et al.*, 'Deep state space models for time series forecasting,' *Advances in Neural Information Processing Systems*, no. NeurIPS, pp. 7785–7794, 2018.

[41] Y. Bengio, A. Courville and P. Vincent, 'Representation learning: A review and new perspectives,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013. DOI: 10.1109/TPAMI.2013.50.

[42] D. E. Rumelhart, G. E. Hinton and R. J. Williams, 'Learning representations by back-propagating errors,' *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[43] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[44] J. L. Elman, 'Finding structure in time,' *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. DOI: https://doi.org/10.1016/0364-0213(90)90002-E.

[45] A. Graves, 'Generating Sequences With Recurrent Neural Networks,' pp. 1–43, 2013.

[46] M. P. Clements and D. F. Hendry, 'On the limitations of comparing mean square forecast errors,' *Journal of Forecasting*, vol. 12, no. 8, pp. 617–637, 1993. DOI: 10.1002/for.3980120802.

[47] S. Makridakis, 'Accuracy measures: theoretical and practical concerns,' *International Journal of Forecasting*, vol. 9, no. 4, pp. 527–529, 1993. DOI: 10.1016/0169-2070(93)90079-3.

[48] J. S. Armstrong and R. Fildes, 'Correspondence on the selection of error measures for comparisons among forecasting methods,' *Journal of Forecasting*, vol. 14, no. 1, pp. 67–71, 1995. DOI: 10.1002/for.3980140106.

[49] Y. Kang, R. J. Hyndman and K. Smith-Miles, 'Visualising forecasting algorithm performance using time series instance spaces,' *International Journal of Forecasting*, vol. 33, no. 2, pp. 345–358, 2017. DOI: 10.1016/j.ijforecast.2016.09.004.

[50] P. H. Franses, 'A note on the Mean Absolute Scaled Error,' *International Journal of Forecasting*, vol. 32, no. 1, pp. 20–22, 2016. DOI: 10.1016/j.ijforecast.2015.03.008.

[51] S. Makridakis, A. Andersen, R. Carbone *et al.*, 'The accuracy of extrapolation (time series) methods: Results of a forecasting competition,' *Journal of Forecasting*, vol. 1, no. 2, pp. 111–153, 1982. DOI: https://doi.org/10.1002/for.3980010202.

[52] N. H. An and D. T. Anh, 'Comparison of Strategies for Multi-step-Ahead Prediction of Time Series Using Neural Network,' *Proceedings - 2015 International Conference on Advanced Computing and Applications, ACOMP 2015*, pp. 142–149, 2016. DOI: 10.1109/ACOMP.2015.24.

[53] C. Chatfield, 'Neural networks: Forecasting breakthrough or passing fad?' *International Journal of Forecasting*, vol. 9, no. 1, pp. 1–3, 1993. DOI: 10.1016/0169-2070(93)90043-M.

[54] M. Adya and F. Collopy, 'How effective are neural networks at forecasting and prediction? A review and evaluation,' *Journal of Forecasting*, vol. 17, no. 5-6, pp. 481–495, 1998. DOI: 10.1002/(SICI)1099-131X(1998090)17:5/6<481::AID-FOR709>3.0.CO;2-Q.

[55] A. Krizhevsky, I. Sutskever and G. E. Hinton, 'ImageNet Classification with Deep Convolutional Neural Networks,' in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012.

[56] D. Silver, A. Huang, C. J. Maddison *et al.*, 'Mastering the game of Go with deep neural networks and tree search,' *Nature*, vol. 529, no. 7587, pp. 484–489, 2016. DOI: 10.1038/nature16961.

[57] J. Devlin, M.-W. Chang, K. Lee *et al.*, 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,' *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, vol. 1, pp. 4171–4186, 2018.

[58] N. Laptev, J. Yosinski, L. Erran Li *et al.*, 'Time-series Extreme Event Forecasting with Neural Networks at Uber,' in *International Conference on Machine Learning (ICML)*, 2017, pp. 1–5.

[59] R. Yu, S. Zheng, A. Anandkumar *et al.*, 'Long-term Forecasting using Higher Order Tensor RNNs,' *arXiv*, vol. 1, pp. 1–24, 2017. URL: http://arxiv.org/abs/1711.00073.

[60] R. Wen, K. Torkkola, B. Narayanaswamy *et al.*, 'A Multi-Horizon Quantile Recurrent Forecaster,' *arXiv*, no. Nips 2017, 2017. URL: http://arxiv.org/abs/1711.11053.

[61] I. Fox, L. Ang, M. Jaiswal *et al.*, 'Deep Multi-Output Forecasting,' in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA: ACM, 2018, pp. 1387–1395. DOI: 10.1145/3219819.3220102.

[62] Y. N. Dauphin, A. Fan, M. Auli *et al.*, 'Language Modeling with Gated Convolutional Networks,' *34th International Conference on Machine Learning, ICML 2017*, vol. 2, pp. 1551–1559, 2016.

[63] J. Gehring, M. Auli, D. Grangier *et al.*, 'Convolutional Sequence to Sequence Learning,' *34th International Conference on Machine Learning, ICML 2017*, vol. 3, pp. 2029–2042, 2017.

[64] S. Bai, J. Z. Kolter and V. Koltun, 'An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling,' *arXiv*, 2018. URL: http://arxiv.org/abs/1803.01271.

[65] E. Choi, M. T. Bahadori, J. A. Kulas *et al.*, 'RETAIN: An Interpretable Predictive Model for Healthcare using Reverse Time Attention Mechanism,' *Advances in Neural Information Processing Systems*, no. Nips, pp. 3512–3520, 2016. URL: http://arxiv.org/abs/1608.05745.

[66] Y. Qin, D. Song, H. Chen *et al.*, 'A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction,' *arXiv*, pp. 2627–2633, 2017. URL: http://arxiv.org/abs/1704.02971.

[67] G. Lai, W.-C. Chang, Y. Yang *et al.*, 'Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks,' in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, New York, NY, USA: ACM, 2018, pp. 95–104. DOI: 10.1145/3209978.3210006.

[68] B. Lim, S. O. Arik, N. Loeff *et al.*, 'Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting,' *arXiv*, no. Bryan Lim, pp. 1–27, 2019. URL: http://arxiv.org/abs/1912.09363.

[69] H. F. Yu, N. Rao and I. S. Dhillon, 'Temporal regularized matrix factorization for high-dimensional time series prediction,' in *Advances in Neural Information Processing Systems*, 2016, pp. 847–855.

[70] Y. Wang, A. Smola, D. C. Maddix *et al.*, 'Deep Factors for Forecasting,' *36th International Conference on Machine Learning, ICML 2019*, pp. 11 460–11 475, 2019. URL: http://arxiv.org/abs/1905.12417.

[71] M. Seeger, D. Salinas and V. Flunkert, 'Bayesian intermittent demand forecasting for large inventories,' *Advances in Neural Information Processing Systems*, no. Nips, pp. 4653–4661, 2016.

[72] D. Salinas, V. Flunkert and J. Gasthaus, 'DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks,' *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2017. DOI: 10.1016/j.ijforecast.2019.07.001.

[73] N. C. Schwertman, A. J. Gilks and J. Cameron, 'A Simple Noncalculus Proof That the Median Minimizes the Sum of the Absolute Deviations,' *The American Statistician*, vol. 44, no. 1, pp. 38–39, 1990. DOI: 10.1080/00031305.1990.10475690.

[74] S. Kolassa, 'Evaluating predictive count data distributions in retail sales forecasting,' *International Journal of Forecasting*, vol. 32, no. 3, pp. 788–803, 2016. DOI: 10.1016/j.ijforecast.2015.12.004.

[75] L. Frías-Paredes, F. Mallor, T. León *et al.*, 'Introducing the Temporal Distortion Index to perform a bidimensional analysis of renewable energy forecast,' *Energy*, vol. 94, pp. 180–194, 2016. DOI: 10.1016/j.energy.2015.10.093.

[76] H. Akaike, 'Information Theory and an Extension of the Maximum Likelihood Principle,' in *Selected Papers of Hirotugu Akaike*, New York, NY: Springer New York, 1998, pp. 199–213. DOI: 10.1007/978-1-4612-1694-0{_}15.

[77] T. Young, D. Hazarika, S. Poria *et al.*, 'Recent Trends in Deep Learning Based Natural Language Processing,' *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2017. DOI: 10.1109/MCI.2018.2840738.

[78] 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.

[79] A. v. d. Oord, S. Dieleman, H. Zen *et al.*, 'WaveNet: A Generative Model for Raw Audio,' pp. 1–15, 2016. URL: http://arxiv.org/abs/1609.03499.

[80] J. Long, E. Shelhamer and T. Darrell, 'Fully Convolutional Networks for Semantic Segmentation,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2014. DOI: 10.1109/TPAMI.2016.2572683.

[81] K. He, X. Zhang, S. Ren *et al.*, 'Deep Residual Learning for Image Recognition,' *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2016-Decem, pp. 770–778, 2015. DOI: 10.1109/CVPR.2016.90.

[82] T. Salimans and D. P. Kingma, 'Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks,' *Advances in Neural Information Processing Systems*, pp. 901–909, 2016. URL: http://arxiv.org/abs/1602.07868.

[83] V. Nair and G. Hinton, 'Rectified Linear Units Improve Restricted Boltzmann Machines,' in *Proceedings of ICML*, vol. 27, 2010, pp. 807–814.

[84] N. Srivastava, G. Hinton, A. Krizhevsky *et al.*, 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting,' *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[85] S. Li, X. Jin, Y. Xuan *et al.*, 'Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting,' *Advances in Neural Information Processing Systems*, vol. 32, no. NeurIPS, 2019.

[86] A. Vaswani, N. Shazeer, N. Parmar *et al.*, 'Attention Is All You Need,' *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. Nips, pp. 5999–6009, 2017. URL: http://arxiv.org/abs/1706.03762.

[87] D.-A. Clevert, T. Unterthiner and S. Hochreiter, 'Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),' *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–14, 2015. URL: http://arxiv.org/abs/1511.07289.

[88] J. L. Ba, J. R. Kiros and G. E. Hinton, 'Layer Normalization,' 2016. URL: http://arxiv.org/abs/1607.06450.

[89] D. Dua and C. Graff, *UCI Machine Learning Repository*, 2017. URL: http://archive.ics.uci.edu/ml.

[90] G. Heber, A. Lunde, N. Shephard *et al.*, *Oxford-Man Institute's realized library*, 2009. URL: https://realized.oxford-man.ox.ac.uk/.

[91] M. M. Dacorogna, U. A. Müller, R. J. Nagler *et al.*, 'A geographical model for the daily and weekly seasonal volatility in the foreign exchange market,' *Journal of International Money and Finance*, vol. 12, no. 4, pp. 413–438, 1993. DOI: 10.1016/0261-5606(93)90004-U.

[92] Z. R. Lai, D. Q. Dai, C. X. Ren *et al.*, 'A peak price tracking-based learning system for portfolio selection,' *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 7, pp. 2823–2832, 2018. DOI: 10.1109/TNNLS.2017.2705658.

[93] H. Zhou, S. Zhang, J. Peng *et al.*, 'Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting,' 2020. URL: http://arxiv.org/abs/2012.07436.

[94] T. Akiba, S. Sano, T. Yanase *et al.*, 'Optuna: A Next-generation Hyperparameter Optimization Framework,' in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA: ACM, 2019, pp. 2623–2631. DOI: 10.1145/3292500.3330701.

[95] A. Paszke, S. Gross, F. Massa *et al.*, 'PyTorch: An Imperative Style, High-Performance Deep Learning Library,' in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[96] J. Beitner, *PyTorch Forecasting: Time series forecasting with PyTorch*, 2020. URL: https://github.com/jdb78/pytorch-forecasting.

[97] W. Falcon, J. Borovec, A. Wälchli *et al.*, 'PyTorch Lightning,' vol. 3, 2019. URL: https://github.com/PyTorchLightning/pytorch-lightning.

[98] R Core Team, *R: A Language and Environment for Statistical Computing*, Vienna, Austria, 2017. URL: https://www.r-project.org/.

[99] M. Själander, M. Jahre, G. Tufte *et al.*, *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure*, 2019.

[100] V. Le Guen and N. Thome, 'Shape and Time Distortion Loss for Training Deep Time Series Forecasting Models,' *arXiv*, no. NeurIPS, pp. 1–13, 2019.

A | Supplementary Material

A.1 Final Hyperparameters for Deep Learning Models

Table A.1-A.4 shows the optimized hyperparameters for every deep learning model on every dataset. The hyperparameter tuning process is described in section 5.2.

Table A.1: Final hyperparameters for the DeepAR model

	Electricity	Traffic	Volatility	Solar	Wind
Learning rate	1×10^{-3}	1×10^{-3}	1×10^{-5}	3×10^{-4}	4×10^{-4}
Num. layers	3	3	3	2	2
Hidden size	40	40	208	48	256
Dropout	0.1	0.1	0.3	0.3	0.9
Max gradient norm.	1×10^{-2}	1×10^{-2}	10	11	0.2
Batch size	64	64	64	128	64

Table A.2: Final hyperparameters for the Seq2Seq model

	Electricity	Traffic	Volatility	Solar	Wind
Learning rate	3×10^{-3}	1×10^{-4}	3×10^{-5}	1×10^{-3}	7×10^{-3}
Num. layers	3	2	1	3	2
Hidden size	96	240	224	64	144
Dropout	0.1	0.8	0.3	0.5	0.4
Max gradient norm.	0.1	0.3	3×10^{-2}	2	20
Batch size	64	64	64	128	64

Table A.3: Final hyperparameters for the TCN model

	Electricity	Traffic	Volatility	Solar	Wind
Learning rate	1×10^{-4}	8×10^{-5}	2×10^{-6}	5×10^{-4}	9×10^{-6}
Num. layers	5	4	5	4	6
Hidden size	256	224	224	176	64
Dropout	0.1	0.6	0.9	0.7	0.2
Max gradient norm.	3×10^{-2}	0.9	0.6	48	3×10^{-2}
Batch size	64	64	64	128	64
Kernel Size	4	10	10	6	12

Table A.4: Final hyperparameters for the TFT model

	Electricity	Traffic	Volatility	Solar	Wind
Learning rate	1×10^{-3}	1×10^{-3}	1×10^{-2}	4×10^{-6}	6×10^{-3}
Hidden size	160	320	160	224	48
Dropout	0.1	0.3	0.3	0.2	0.7
Max gradient norm.	1×10^{-2}	1×10^{2}	1×10^{-2}	13	0.7
Batch size	64	128	64	128	64
Attention head size	4	4	1	1	1

A.2 Variance Weighted Errors on Wind and Solar

Figure A.1 shows RMSSE and RMSSE_{VW} computed for the volatility and wind datasets, grouped by day of week. As we discussed in section 6.2, errors are not significantly impacted by the variance weighting procedure because there is close to constant variance over the weekly period.

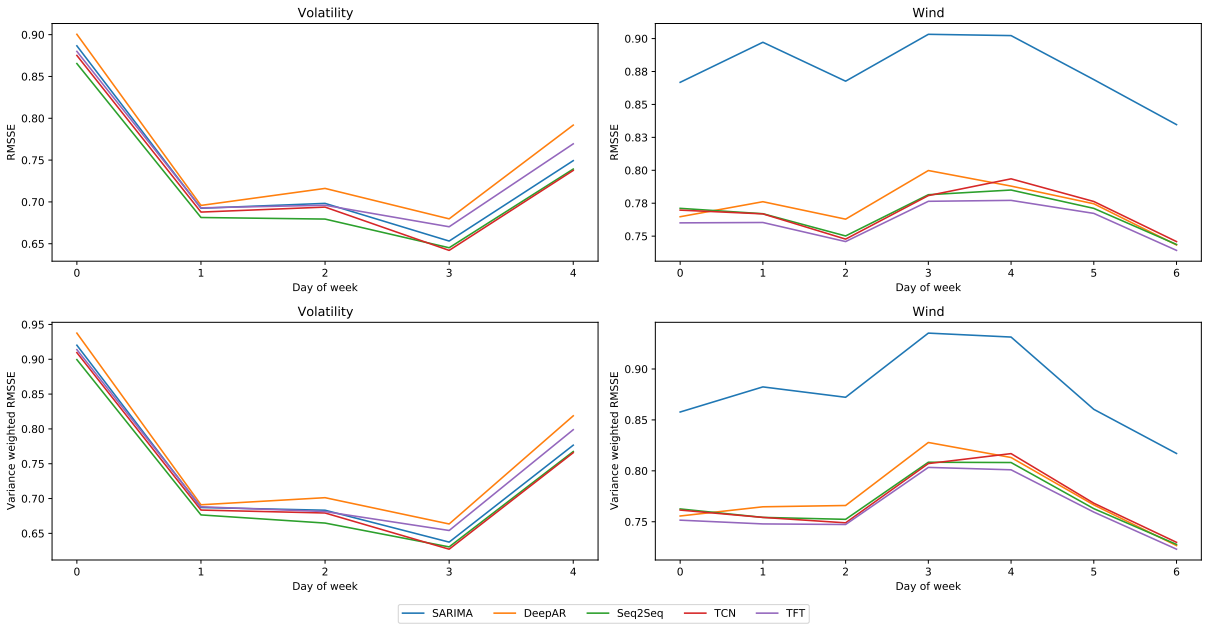


Figure A.1: Results in terms of RMSSE (first row) and variance weighted RMSSE (second row) computed per day of week for the volatility and wind datasets

A.3 Distribution of TDI and TDM Metrics on Traffic

Figure A.2 shows distributional plots for the TDI and TDM metric computed on the traffic dataset. As discussed in section 6.5, the interpretation of the TDM metric is difficult as most forecasts end up at the extremes of ± 1.

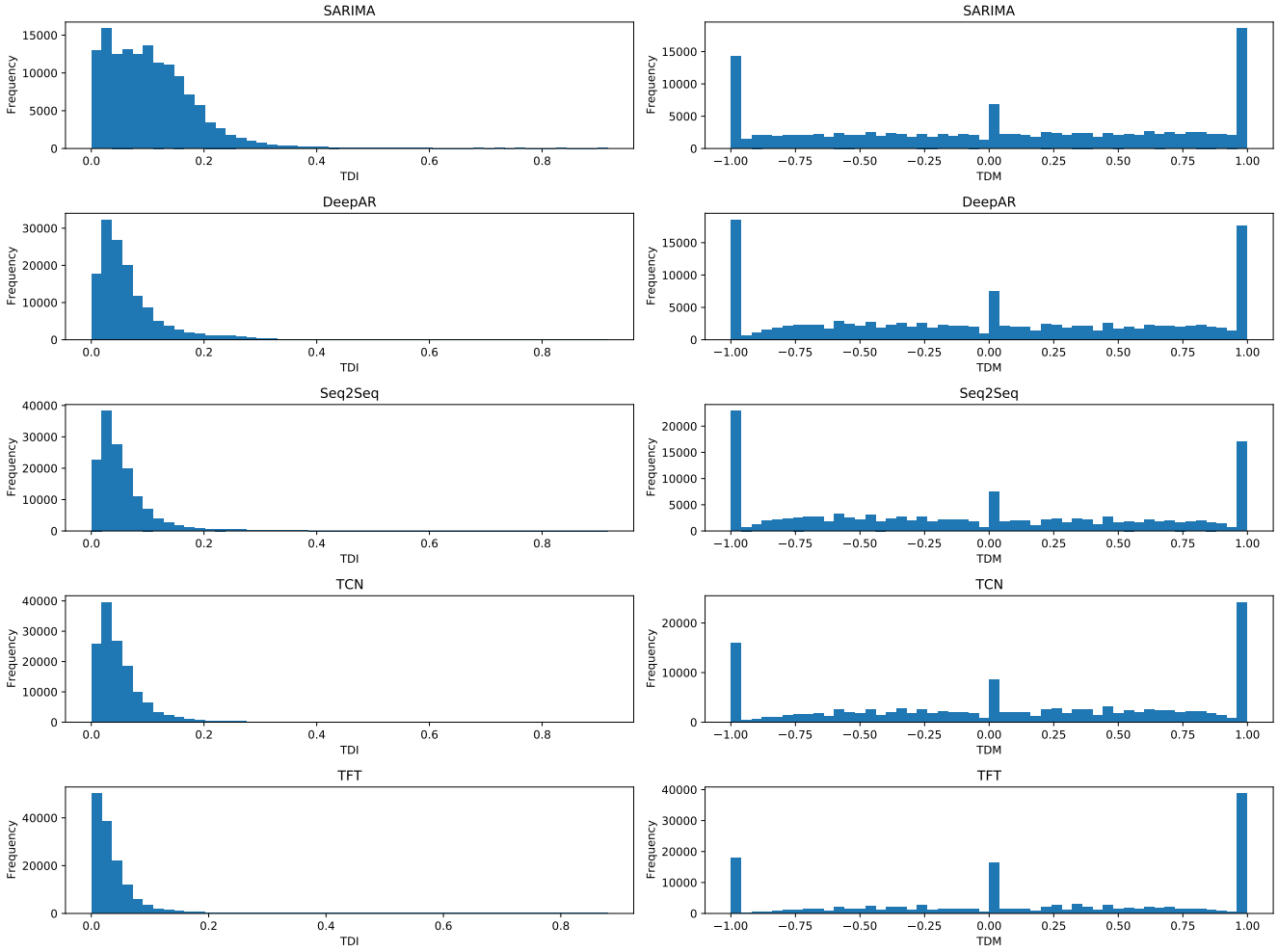
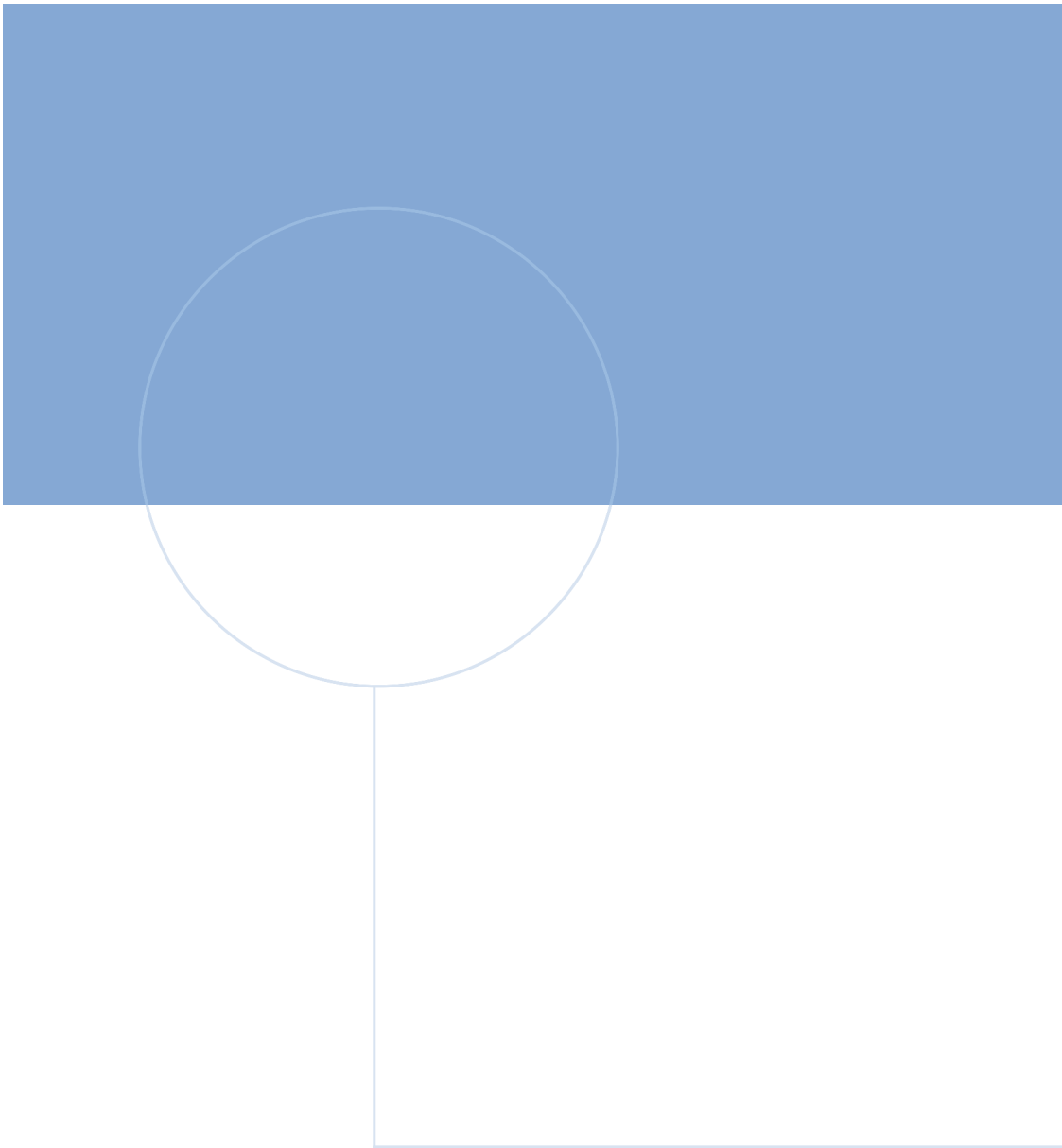


Figure A.2: The distribution of the TDI and TDM metrics computed on the traffic dataset



NTNU

Norwegian University of
Science and Technology