National Technical University of Athens
School of Electrical and Computer Engineering
Division of

Machine Learning for Forecasting: A Comparative Analysis

Performance Assessment on Electricity Load and Traffic Datasets

Diploma Thesis

of

KONSTANTINOS PAPADAKIS

Supervisor: Stephanos Kollias
Professor Emeritus

Athens, March 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of

Machine Learning for Forecasting: A Comparative Analysis

Performance Assessment on Electricity Load and Traffic Datasets

Diploma Thesis

of

KONSTANTINOS PAPADAKIS

Supervisor: Stephanos Kollias
Professor Emeritus

Approved by the examination committee on 15th March 2024.

(Signature)	*(Signature)*	*(Signature)*
....	....	....
Stephanos Kollias	Georgios Stamou	Athanasios Voulodimos
Professor Emeritus	Professor	Assistant Professor

Athens, March 2024

National Technical University of Athens
School of Electrical and Computer Engineering
Division of

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Abstract

This thesis investigates the performance of advanced machine learning models for time series forecasting. Prophet, N-BEATS, DeepAR, DeepVAR, and the Temporal Fusion Transformer are applied to the Electricity Load Diagrams and PEMS-SF datasets. Results are rigorously evaluated using appropriate forecasting metrics. The study highlights the strengths and weaknesses of each model in handling real-world data complexities, offering insights for choosing optimal forecasting methods based on data characteristics and problem domain.

Keywords

Timeseries, Forecasting, Machine Learning, Deep Learning, Neural Networks, N-BEATS, DeepAR, DeepVAR, Temporal Fusion Transformer, Prophet

to my family

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Prof. Stephanos Kolias, and cosupervisor, Dr. Paraskevi Tzouveli, for their invaluable guidance, insightful feedback, and unwavering support throughout this research project. Their expertise and encouragement were instrumental in shaping this work.

Finally, a heartfelt thank you to my family, my girlfriend and my friends for their unconditional love, patience, and continuous belief in me. This accomplishment would not have been possible without their support.

Athens, February 2024

Konstantinos Papadakis

Table of Contents

List of Figures

List of Tables

Preface

My interest in the field of forecasting stems from a desire to understand the complex patterns that shape our world and to develop tools that can inform future decision-making. Traditional forecasting methods, while valuable, often fall short when dealing with the nonlinearities and multiple dependencies present in real-world data. Machine learning presents a compelling avenue for creating more adaptable and accurate forecasting models.

This thesis delves into the application of several cutting-edge machine learning techniques to the task of time series forecasting. I was particularly drawn to the diverse capabilities of models like Prophet, N-BEATS, DeepAR, DeepVAR, and the Temporal Fusion Transformer. My goal was to investigate how these models perform on datasets of practical importance, such as energy consumption and traffic flow.

Throughout this research journey, I've gained a deeper appreciation for the challenges and nuances of forecasting. This thesis represents the culmination of my efforts, and I hope it offers valuable insights into the potential of machine learning in this field.

Chapter 1

Introduction

The ability to accurately forecast future trends in time series data holds immense value across diverse fields. From businesses seeking to optimize inventory and supply chains to policymakers aiming for efficient energy distribution and infrastructure planning, reliable forecasting underpins informed decision-making. Traditional statistical methods like ARIMA and exponential smoothing [7] have long provided a foundation for time series analysis. However, these methods often face limitations when confronting the complexities of real-world data, particularly in scenarios marked by nonlinear patterns, multiple interacting variables, and irregular seasonality.

The rise of machine learning (ML) has ushered in a new era of possibilities for time series forecasting. ML algorithms possess an inherent capacity to uncover intricate patterns and relationships within vast and complex datasets. This enables them to model nonlinearities, handle high-dimensional data, and potentially incorporate exogenous variables – aspects that pose challenges for traditional methods. Consequently, ML-based forecasting models have garnered significant interest within the research community and across industries.

This thesis delves into the application of several state-of-the-art machine learning models specifically designed for time series forecasting. The selected models encompass a range of methodologies:

Prophet: Developed by Facebook, Prophet offers flexibility and the ability to handle data characterized by trends, multiple seasonalities, and the impact of holidays [1].

DeepAR: A deep learning model from Amazon that leverages recurrent neural networks to model probabilistic distributions of future values, providing insights into potential uncertainty [5].

DeepVAR: A more sophisticated probabilistic deep learning model extending the capabilities of DeepAR [8].

N-BEATS: A deep neural architecture employing a unique combination of backward and forward residual links and interpretable basis functions [4].

Temporal Fusion Transformer: A cutting-edge attention-based model designed to handle high-dimensional time series with long-range dependencies and a multitude of variables [6].

This research will subject these models to a rigorous evaluation on two real-world datasets: the *Electricity Load Diagrams* [9] and the *PeMS-SF* [10] traffic dataset. These datasets offer distinct forecasting challenges, allowing for a comprehensive assessment of model strengths and weaknesses under varying conditions.

It should be mentioned that many deep neural network models have been developed and applied for forecasting and prediction of time series in a variety of applications by members of the Artificial Intelligence and Learning Systems Lab of the National Technical University of Athens. In particular, CNN-RNN architectures, Bayesian architectures with capsules and uncertainty estimation, semi and self-supervised learning algorithms, domain adaptation, augmentation, transformers and attention methodologies have been developed and used in applications, such as medical imaging [11, 12], image captioning [13], fault detection in nuclear power stations [14, 15], agri-food production prediction [16, 17, 18], load demand forecasting of power systems [19], human behavior prediction [20, 21], as well as for capsule networks [22, 23, 24] and transparency [25, 26].

Chapter 2

Forecasting Models

This chapter provides a comprehensive exploration of the machine learning models central to this thesis: Prophet, DeepAR, DeepVAR, N-BEATS, and the Temporal Fusion Transformer. These diverse models represent a range of approaches to forecasting, from probabilistic formulations to advanced neural network architectures. Understanding the underlying mechanisms of each model is crucial for both their effective application and interpreting their results.

The chapter will delve into the core concepts, mathematical foundations, unique strengths, and potential limitations of each model. This exploration will lay the groundwork for the subsequent evaluation of these models on real-world datasets. By critically examining their theoretical basis and practical design, we can gain valuable insights into their performance, make informed choices, and drive their effective application in solving complex forecasting problems.

2.1 Prophet

2.1.1 Model Overview

Prophet [1] is a forecasting tool developed by Facebook's Core Data Science team, designed to address the challenges of forecasting time series data that exhibit strong seasonal effects. It is particularly well-suited for data with clear patterns of seasonality and holidays, which are common in business metrics like sales, website traffic, and demand forecasting. Prophet's intuitive approach and ease of use make it accessible to both data scientists and analysts, facilitating its adoption in a wide range of applications.

The time series in Figure 2.1 provides a useful illustration of the difficulties in producing reasonable forecasts with fully automated methods [2, 27, 7]. Figure 2.2 shows forecasts using several automated procedures from the R forecast package [28]. The forecasts are shown at three different points in the history of the time series, and it is clear that the automated procedures are not able to capture the complex patterns in the data. In contrast, Figure 2.3 shows forecasts made by Prophet, which can capture the complex patterns in the data. Tuning these traditional methods requires a lot of expertise and manual effort,

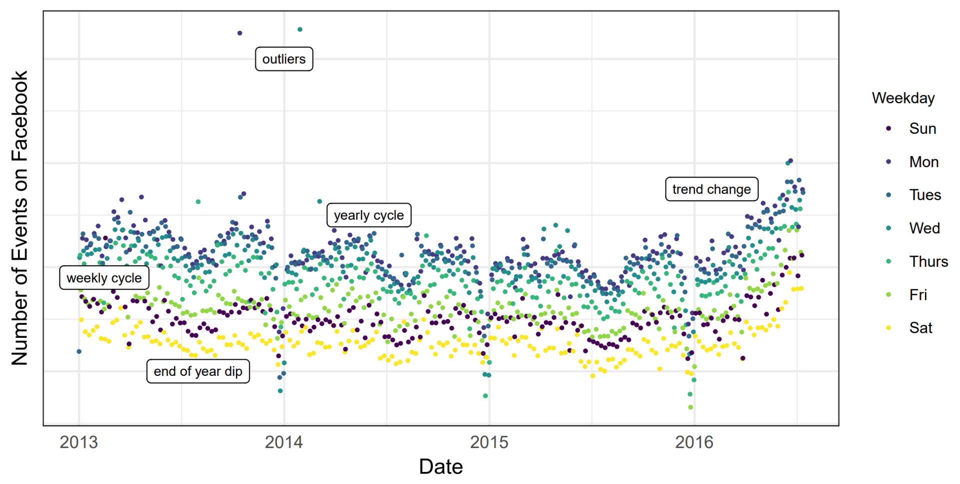


Figure 2.1. *The number of events created on Facebook. There is a point for each day, and points are color-coded by day of the week to show the weekly cycle. The features of this time series are representative of many business time series: multiple strong seasonalities, trend changes, outliers, and holiday effects. Figure from [1].*

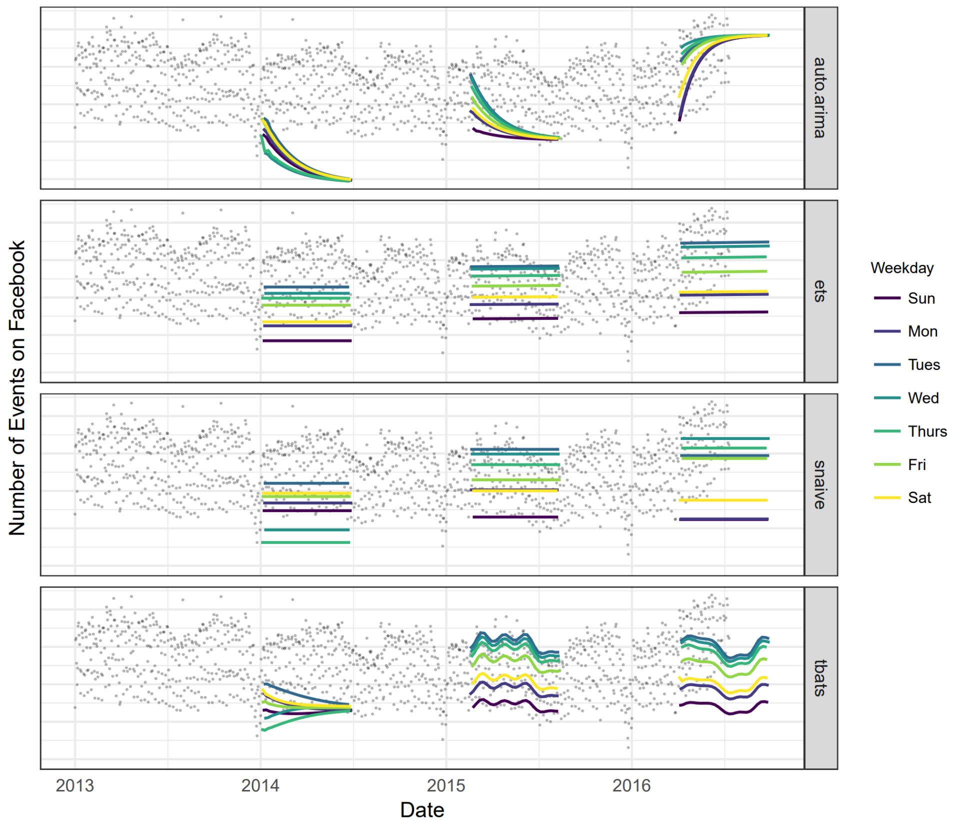


Figure 2.2. *Forecasts on the time series from Figure 2.1 using a collection of automated forecasting procedures [2]. Forecasts were made at three illustrative points in history, each using only a portion of the time series up to that point. Forecasts for each day are grouped and colored by day of the week to visualize weekly seasonality. Figure from [1].*

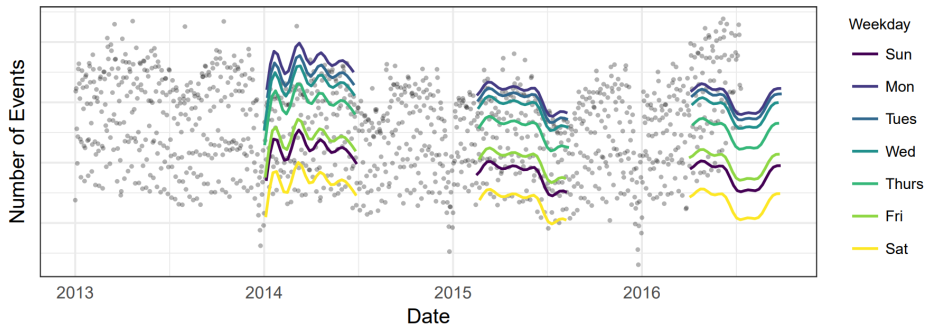


Figure 2.3. *Forecasts on the time series from Figure 2.1 using Prophet. Figure from [1].*

while Prophet can produce high-quality forecasts with minimal tuning. This allows the Prophet model to be used at scale, which fits our case.

Prophet is a decomposable time series model [29] that consists of three main components: trend, seasonality, and holidays. The trend component models non-periodic changes over time, the seasonality component captures periodic changes driven by seasonality (e.g. weekly, monthly, or yearly), and the holidays component allow for the modeling of irregular events that can impact the time series. The model is formulated as follows:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t \tag{2.1}$$

where $g(t)$ is the trend component, $s(t)$ is the seasonality component, $h(t)$ is the holiday component, and ϵ_t is the error term, assumed to be normally distributed around 0.

This specification is similar to a Generalized Additive Model (GAM) [30], a class of regression models with potentially non-linear smoothers applied to the regressors. In Prophet only time is used as a regressor but possibly several linear and non-linear functions of time as components. Modeling seasonality as an additive component is the same approach taken by exponential smoothing [31]. Multiplicative seasonality, where the seasonal effect is a factor that multiplies $g(t)$, can be accomplished through a log transform. The GAM approach is flexible and allows for the inclusion of additional components, and it also fits very quickly with the L-BGFS algorithm.

Prophet is framing the forecasting problem as a curve-fitting exercise, which is inherently different from time series models that explicitly account for the temporal dependence structure in the data. While this approach lacks some important advantages of using a generative model such as ARIMA [32], it provides several practical advantages

- Flexibility: Seasonality with multiple periods is easily accommodated, and many different assumptions about the trend can be imposed.
- Robustness to missing values: The measurements do not need to be regularly spaced, and we do not need to interpolate missing values e.g. from removing outliers.
- Fast fitting: Allowing explorations of many possible models.

- Easily Interpretable Parameters: The parameters of the model have a clear interpretation, which can be useful for introducing domain knowledge to the model.

2.1.2 Model Architecture

The Trend Model

The trend model can be either a *saturating growth* model, or a *piecewise linear* model.

Saturating growth is found in many growth processes, where there is nonlinear growth that saturates at a carrying capacity. This sort of growth is typically modeled using the logistic growth model, which in its most basic form is

$$g(t) = \frac{C}{1 + \exp(-k(t - m))} \tag{2.2}$$

with C being the carrying capacity, k the growth rate, and m an offset parameter.

In many real-world scenarios, the carrying capacity as well as the growth are not constant. Prophet allows for a more flexible model by allowing the carrying capacity to change over time, and by allowing the growth rate to be a piecewise linear function of time.

The carrying capacity is then modeled as a time-varying capacity $C(t)$.

The variations of growth are modeled as a piecewise linear function $k + \mathbf{a}(t)^T\boldsymbol{\delta}$, where

$$a_j(t) = \begin{cases} 1, & \text{if } t \geq s_j \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

with $s_j,\ j = 1, \ldots, S$ being the changepoint times, and δ_j being the change in rate that occurs at time s_j. When the rate k is adjusted, the offset parameter m must also be adjusted to ensure that the function is continuous at the change points. The correct adjustment at changepoint j is

$$\gamma_j = \left(s_j - m - \sum_{l<j} \gamma_l\right)\left(1 - \frac{k + \sum_{l<j} \delta_l}{k + \sum_{l \leq j} \delta_l}\right) \tag{2.4}$$

The piecewise logistic growth model is then

$$g(t) = \frac{C(t)}{1 + \exp(-(k + \mathbf{a}(t)^T\boldsymbol{\delta})(t - (m + \mathbf{a}(t)^T\boldsymbol{\gamma})))} \tag{2.5}$$

For forecasting problems where saturating growth is not observed, a piece-wise constant rate of growth offers a simple and often effective model. In this case, the trend model can be expressed as follows:

$$g(t) = (k + \mathbf{a}(t)^T\boldsymbol{\delta})t + (m + \mathbf{a}(t)^T\boldsymbol{\gamma}) \tag{2.6}$$

where $\gamma_j = -s_j\delta_j$.

The change points s_j can either be manually set or automatically selected by the model given a set of candidates. Automatic selection can be done for both the saturating growth and piecewise linear models by imposing a sparsity-inducing prior on the change points, for example, $\delta_j = \text{Laplace}(0, \tau)$. Here τ directly controls the flexibility of the model in altering its rate. As the prior has no impact on the primary growth rate k, we have that as $\tau \to 0$ the fit reduces to standard non-piecewise logistic or linear growth.

After the model is fitted and attempts to extrapolate a forecast, the trend will have a constant rate. Uncertainty in the trend can then be estimated by replacing the rate scale parameter τ with its maximum likelihood estimate $\lambda = \frac{1}{S}\sum_{j=1}^{S} |\delta_j|$. Alternatively, in a fully Bayesian framework, this could be done with a hierarchical prior on τ to obtain its posterior. Future change points are then generatively sampled in such a way that the average frequency of the change points matches that in the history:

$$\forall j > T, \quad \begin{cases} \delta_j = 0 & \text{w.p. } 1 - \frac{S}{T} \\ \delta_j \sim \text{Laplace}(0, \lambda) & \text{w.p. } \frac{S}{T} \end{cases} \tag{2.7}$$

Uncertainty in the forecast trend is measured by assuming that future trends will exhibit the same average frequency and magnitude of rate changes observed in the historical data. The inferred λ is then used to simulate potential future trends, which are subsequently used to compute uncertainty intervals.

The assumption that the trend will continue to change with the same frequency and magnitude as observed in the past is quite strong. Therefore, the uncertainty intervals are not expected to have exact coverage. However, they provide a useful indication of the level of uncertainty and can serve as an indicator of overfitting. Increasing the flexibility of the model by adjusting τ allows for better fitting of the historical data, resulting in lower training error. However, this increased flexibility can lead to wider uncertainty intervals when projecting into the future.

The Seasonality Model

The seasonality component $s(t)$ is modeled as a Fourier series:

$$s(t) = \sum_{n=1}^{N} \left(a_n \cos\left(\frac{2\pi nt}{P}\right) + b_n \sin\left(\frac{2\pi nt}{P}\right) \right) \tag{2.8}$$

where P is the period of the seasonality, and N is the number of Fourier terms. The number of Fourier terms N is a hyperparameter of the model. The seasonality model is flexible and can capture complex seasonal patterns, including multiple seasonalities with different periods. The seasonality model can also be extended to include additional seasonalities, such as daily, weekly, monthly, and yearly seasonality, as well as custom seasonalities.

Fitting seasonality requires estimating the $2N$ coefficients $\boldsymbol{\beta} = [a_1, b_1, \ldots, a_N, b_N]^T$ of the

Fourier components $\mathbf{X}(t) = \left[\cos\left(\frac{2\pi(1)t}{P}\right), \ldots, \sin\left(\frac{2\pi(N)t}{P}\right)\right]$. Using this notation we can rewrite 2.8 as $s(t) = \mathbf{X}(t)\boldsymbol{\beta}$. A smoothing prior is imposed to the seasonality by considering $\boldsymbol{\beta} \sim \text{Normal}(0, \sigma^2)$. Truncating the series at N applies a low-pass filter to the seasonality, therefore increasing N allows for fitting more complex seasonal patterns, but with the risk of overfitting.

The Holiday Model

The incorporation of holidays into the model is simplified by assuming that the effects of holidays are independent. Each holiday i, is associated with a set of past and future dates D_i. An indicator function is introduced to determine if a given time t, falls within holiday i. Additionally, each holiday is assigned a parameter κ_i, which represents the corresponding change in the forecast. The holiday component is then defined as:

$$h(t) = Z(t)\boldsymbol{\kappa} \tag{2.9}$$

where

$$Z(t) = [\mathbf{1}_{t \in D_1}, \ldots, \mathbf{1}_{t \in D_L}] \tag{2.10}$$

As with the seasonality component, a smoothing prior is imposed on the holiday effects by considering $\boldsymbol{\kappa} \sim \text{Normal}(0, \nu^2)$.

It is often necessary to consider the effects of a range of days around a specific holiday. In order to capture these effects, additional parameters are included for the days surrounding the holiday. This approach treats each day within the window around the holiday as if it were a holiday itself.

Model fitting

The entire model can be expressed in a few lines of Stan code [33], given in Listing 1. For model fitting, Stan's L-BFGS [34] is used to find a maximum a-posteriori estimate but also can do full posterior inference to include model parameter uncertainty in the forecast uncertainty.

2.1.3 Strengths and Limitations

Prophet's strengths lie in its simplicity and effectiveness in dealing with common challenges in time series forecasting, such as seasonality and holiday effects. It performs exceptionally well on daily and weekly data, making it a popular choice for business and financial metrics forecasting. Prophet automatically detects changes in trends and seasonality, making it robust to missing data and shifts in patterns. This adaptability is crucial for real-world datasets that often exhibit non-linear trends and abrupt changes. With its intuitive API and minimal need for manual tuning, Prophet allows users to quickly generate forecasts. This ease of use democratizes access to advanced forecasting techniques, enabling users from various backgrounds to leverage sophisticated time series analysis. Prophet provides

Listing 1: Fitting a time series with Prophet

```
model {
    // Priors
    k ~ normal(0, 5);
    m ~ normal(0, 5);
    epsilon ~ normal(0, 0.5);
    delta ~ double_exponential(0, tau);
    beta ~ normal(0, sigma);

    // Logistic likelihood
    y ~ normal(C ./ (1 + exp(-(k + A * delta) .* (t - (m + A * gamma)))) +
               X * beta, epsilon);

    // Linear likelihood
    y ~ normal((k + A * delta) .* t + (m + A * gamma) + X * beta, sigma);
}
```

built-in functionality for estimating uncertainty intervals in forecasts (although limited), which is essential for risk management and planning purposes.

While Prophet offers significant advantages, it has limitations, particularly when dealing with highly irregular time series, multivariate datasets, or when the forecasting task requires capturing complex interactions between variables. It is primarily designed for univariate time series forecasting, which may limit its applicability in scenarios requiring the consideration of external factors or interdependencies between multiple series.

2.2 N-BEATS

2.2.1 Model Overview

Neural basis Expansion Analysis for Interpretable Time Series Forecasting (N-BEATS) [4] is a deep neural architecture based on backward and forward residual links and a deep stack of fully connected layers. The architecture has many desirable properties, being interpretable, applicable without modification to a wide array of target domains, and fast to train. It represents a significant departure from traditional time-series forecasting models, and eschewing recurrent or convolutional layers commonly found in other deep learning approaches for time series analysis. It was the first architecture to empirically demonstrate that pure Deep Learning using no time-series specific components outperforms well-established statistical approaches on datasets like M3 [35], M4 [36] and the tourism dataset [37]. It is also feasible to design an N-BEATS architecture with an interpretable output composed of a trend and a seasonality component.

The architecture is based on a stack of fully connected layers, with each layer having several blocks. Each block consists of a backcast and forecast sub-network, with the backcast sub-network predicting the residuals of the target time series and the forecast sub-network partially predicting the future values of the target time series. The architecture is trained

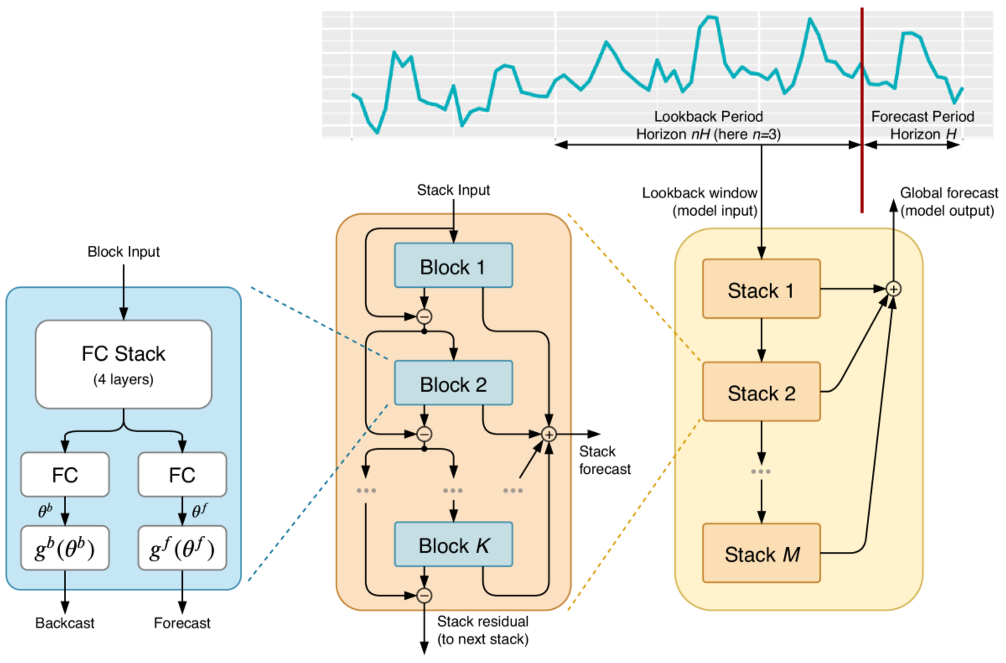


Figure 2.4. *The N-BEATS Architecture. The basic building block is a multi-layer fully connected network with ReLU nonlinearities [3]. It predicts basis expansion coefficients both forward, $\boldsymbol{\theta}^f$, (forecast) and backward, $\boldsymbol{\theta}^b$, (backcast). Blocks are organized into stacks using the doubly residual stacking principle. A stack may have layers with shared g^b and g^f functions. Forecasts are aggregated hierarchically. This enables building a very deep neural network with interpretable outputs. Figure from [4].*

using a combination of L1 and L2 loss functions, and the backcast and forecast sub-networks are trained jointly.

The model can be trained on multiple time series. If every time series is interpreted as a separate task, this can be linked back to multitask learning and to meta-learning [38], in which a neural network is regularized by learning on multiple tasks to improve generalization.

2.2.2 Model Architecture

The basic building block has a fork architecture and is depicted in Figure 2.4 (left). The index l is omitted in the image for brevity. The l-th block of a stack accepts its respective input $\mathbf{x}_l$ and outputs two vectors $\hat{\mathbf{x}}_l$ and $\hat{\mathbf{y}}_l$. At the first Block ($l = 1$) $\mathbf{x}_1 = \mathbf{x}$ and for the other blocks it is $\mathbf{x}_l = \mathbf{x}_{l-1} - \hat{\mathbf{x}}_{l-1}$. The forecast $\hat{\mathbf{y}}_l$ is a partial forecast. Partial forecasts are then summed to produce the stack forecast $\sum_l \hat{\mathbf{y}}_l$, and stack forecasts are summed to produce the final forecast $\hat{\mathbf{y}}$. Each block l performs the following operations:

$$\mathbf{h}_{l,1} = \mathrm{FC}_{l,1}(\mathbf{x}_l), \quad \mathbf{h}_{l,2} = \mathrm{FC}_{l,2}(\mathbf{h}_{l,1}), \quad \mathbf{h}_{l,3} = \mathrm{FC}_{l,3}(\mathbf{h}_{l,2}), \quad \mathbf{h}_{l,4} = \mathrm{FC}_{l,4}(\mathbf{h}_{l,1}), \tag{2.11}$$

$$\boldsymbol{\theta}_l^b = \mathrm{LINEAR}_l^b(\mathbf{h}_{l,4}), \quad \boldsymbol{\theta}_l^f = \mathrm{LINEAR}_l^f(\mathbf{h}_{l,4}) \tag{2.12}$$

$$\hat{\mathbf{y}}_l = \sum_{i=1}^{\dim(\boldsymbol{\theta}_l^f)} \boldsymbol{\theta}_{l,i}^f \mathbf{v}_i^f, \quad \hat{\mathbf{x}}_l = \sum_{i=1}^{\dim(\boldsymbol{\theta}_i^b)} \boldsymbol{\theta}_{l,i}^b \mathbf{v}_i^b \tag{2.13}$$

This part of the architecture attempts to find forecast and backcast coefficients $\boldsymbol{\theta}_l^f$ and $\boldsymbol{\theta}_l^b$ that are used as projection coefficients for the basis vectors $\mathbf{v}_i^f$ and $\mathbf{v}_i^b$ of the basis layers g_l^f and g_l^b so that the result is an accurate estimation of the partial stack forecast and backast $\hat{\mathbf{y}}_l$ and $\hat{\mathbf{x}}_l$. The input $\mathbf{x}_l$ is the residual of the target time series at the l-th block. The basis vectors can be chosen to be learnable, or they can have a fixed structure to impose problem-specific biases and/or make the results interpretable, as we will discuss in Section 2.2.2.

The partial and residual inputs are then passed to the next stack. Each stack has its own (learnable or not) basis vectors. The final forecast is the (double) sum of the partial forecasts of each block of each stack. This approach is called doubly residual stacking and it results in interpretable results unlike other residual connection approaches [39] [40].

There are two main configurations of the N-BEATS architecture, the *Generic* architecture and the *Interpretable* architecture.

Generic Architecture

The *Generic architecture* does not rely on any specific time series knowledge (e.g. weekly seasonality). The basis layers g_l^f and g_l^b are set to be a linear projection of the previous layer output, thus the outputs of block l are:

$$\hat{\mathbf{y}}_l = \mathbf{V}_l^f \boldsymbol{\theta}_l^f + \mathbf{b}_l^f \quad \hat{\mathbf{x}}_l = \mathbf{V}_l^b \boldsymbol{\theta}_l^b + \mathbf{b}_l^b \tag{2.14}$$

The interpretation of this model is that the FC layers in the basic building block depicted in Figure 2.4 learn the predictive decomposition of the partial forecast by $\hat{\mathbf{y}}_l$ in the basis $\mathbf{V}_l^f$ learned by the network. Matrix $\mathbf{V}_l^f$ has dimensionality $H \times \dim(\boldsymbol{\theta}_l^f)$. Therefore, the first dimension of $\mathbf{V}_l^f$ has the interpretation of a discrete-time index in the forecast domain. The second dimension of the matrix has the interpretation of the indices of the basis functions, with $\boldsymbol{\theta}_l^f$ being the expansion coefficients for this basis. Thus the columns of $\mathbf{V}_l^f$ can be thought of as waveforms in the time domain. Because no additional constraints are imposed on the form of $\mathbf{V}_l^f$, the waveforms learned by the deep model do not have an inherent structure. This leads to $\hat{\mathbf{y}}_l$ not being interpretable.

Interpretable Architecture

The *Interpretable architecture* can be constructed by adding structure to the basis layers at stack level. A common practice in forecasting is to decompose a time series into trend and seasonality. This decomposition can be achieved in N-BEATS by using two stacks: a *trend stack* followed by a *seasonality stack*, with both stacks having fixed basis vectors which we now describe.

The basis vectors of the *trend stack* are $\mathbf{T} := [\mathbf{1}, \mathbf{t}, \dots \mathbf{t}^p]$ where $\mathbf{t} := [0, \dots, H-1]^T / H$, which makes each element of $\hat{\mathbf{y}}_{s,l}^{\text{trend}}$ a polynomial in time (typically of low degree e.g. 2 or 3).

The basis vectors of the *seasonality stack* are such so that each element of $\hat{\mathbf{y}}_{s,l}^{\text{seasonality}}$ is a Fourier series, thus the basis vectors are $\mathbf{S} := [\cos(2\pi k\mathbf{t}),\ \sin(2\pi k\mathbf{t});\quad k = 0, \dots, \lfloor H/2 - 1 \rfloor]$, that is each basis vector is a discretized sinusoid.

In matrix form, we have:

$$\hat{\mathbf{y}}_{s,l}^{\text{trend}} = \mathbf{T}\boldsymbol{\theta}_{s,l}^{f} \tag{2.15}$$

$$\hat{\mathbf{y}}_{s,l}^{\text{seasonality}} = \mathbf{S}\boldsymbol{\theta}_{s,l}^{f} \tag{2.16}$$

The trend stack finds a (low) polynomial trend, then the detrended series is fed into the seasonality stack. The number of blocks is typically 3 for both trend and seasonality and the block weights are typically shared within a stack.

2.2.3 Strengths and Limitations

One of the most significant strengths of N-BEATS is its flexibility and adaptability to various time series forecasting challenges without requiring extensive model reconfiguration. Moreover, the model's architecture promotes ease of use and interpretability.

While N-BEATS excels in many forecasting scenarios, the model's purely data-driven approach means it may not fully incorporate domain-specific knowledge or exogenous variables without modifications to its architecture. Additionally, N-BEATS does not natively provide uncertainty quantification, which could be an essential requirement in risk management and decision-making scenarios.

2.3 DeepAR

2.3.1 Model Overview

DeepAR [5], developed by Amazon Research, is a forecasting model that uses autoregressive recurrent networks [41, 42, 43] for probabilistic forecasting, aiming to predict the future *distribution* of a time series based on its past. It is particularly effective for datasets with many related time series, by learning a *global* model from historical data of *all time series* in the data set. Its probabilistic forecasts allow for the generation of uncertainty intervals, which are crucial for risk management and decision-making.

DeepAR is adept at managing the varied scale of time series datasets commonly encountered in real-world scenarios, such as retail sales data, which often exhibit a power-law distribution in their magnitude. An example of such a distribution is shown in Figure 2.5 which shows the distribution of sales velocity (i.e. average weekly sales of an item) across millions of items sold by Amazon.

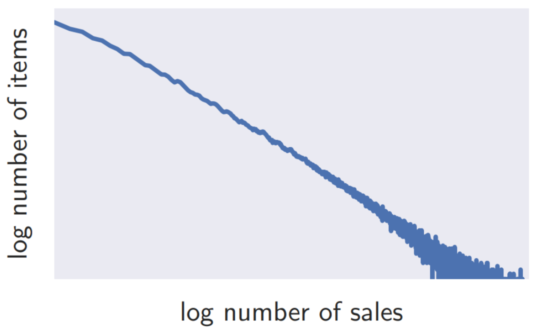


Figure 2.5. *The distribution of sales velocity (i.e. average weekly sales of an item) across millions of items sold by Amazon. Figure from [5].*

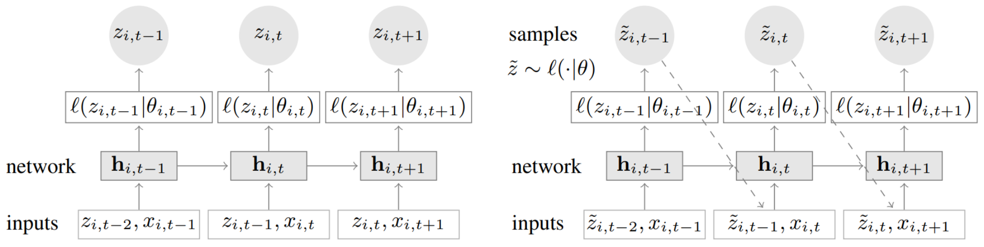


Figure 2.6. *The DeepAR Architecture. Training (left): At each time step t, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, as well as the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(\mathbf{h}_{i,t}, \Theta)$ of the likelihood $\ell(z_i \mid \theta)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\hat{z}_{i,t} \sim \ell(\cdot \mid \theta_{i,t})$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$ generating one sample trace. Repeating this prediction process yields many traces representing the joint predicted distribution. Figure from [5].*

DeepAR excels at capturing seasonal patterns and dependencies on covariates across time series, reducing the need for manual feature engineering. It leverages information from similar items, enabling accurate forecasts even for items with limited or no historical data, a scenario where traditional single-item forecasting methods struggle.

2.3.2 Model Architecture

Likelihood Model

The model's goal is to estimate for each time series i the conditional distribution of its future $\mathbf{z}_{i,t_0:T}$ given its past $\mathbf{z}_{i,1:t_0-1}$ as well as any available covariates $\mathbf{x}_{i,1:T}$, which are assumed to be known at all time points:

$$P\left(\mathbf{z}_{i,t_0:T} \mid \mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T}\right) \tag{2.17}$$

The past and future time ranges are also referred to as conditioning and prediction ranges, respectively.

The distribution is assumed to have a parametric form that consists of a product of likelihood factors:

$$Q_\Theta(\mathbf{z}_{i,t_0:T} \mid \mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} Q_\Theta(z_{i,t} \mid \mathbf{z}_{i,1:t-1}, \mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} \ell(z_{i,t} \mid \theta(\mathbf{h}_{i,t}, \Theta)) \quad (2.18)$$

parametrized by

$$\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta) \quad (2.19)$$

where h is a function implemented by a multi-layer recurrent neural network with LSTM cells.

The model is autoregressive and recurrent, in the sense that it consumes the observation at the last time step $z_{i,t-1}$ as input, and the previous output of the network $\mathbf{h}_{i,t}$ is fed back as input at the next time step. The likelihood $\ell(z_{i,t} \mid \theta(\mathbf{h}_{i,t}))$ is a fixed distribution whose parameters are given by a function $\theta(\mathbf{h}_{i,t}, \Theta)$ of the network output $\mathbf{h}_{i,t}$. Information about the observations in the conditioning range is encapsulated in the network state $\mathbf{h}_{i,t_0-1}$.

Given the model parameters Θ, joint samples $\tilde{\mathbf{z}}_{i,t_0:T} \sim Q_\Theta(\mathbf{z}_{i,t_0:T} \mid \mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$ are obtained through ancestral sampling by sampling $\tilde{z}_{i,t} \sim \ell(\cdot \mid \theta(\tilde{\mathbf{h}}_{i,t}, \Theta))$ where

$$\tilde{\mathbf{h}}_{i,t} = h(\mathbf{h}_{i,t-1}, \tilde{z}_{i,t-1}, \mathbf{x}_{i,t}, \Theta) \quad (2.20)$$

$$\tilde{\mathbf{h}}_{i,t_0-1} = \mathbf{h}_{i,t_0-1} \quad (2.21)$$

$$\tilde{z}_{i,t_0-1} = z_{i,t_0-1} \quad (2.22)$$

The likelihood $\ell(z \mid \theta)$ determines the "noise model" and is chosen to match the statistical properties of the data. It can be any parametric distribution, as long as samples from the distribution can cheaply be obtained, and the log-likelihood and its gradients with respect to the parameters can be evaluated. Typically, the Gaussian distribution is often used for real-valued data, the Beta Distribution for data in the unit interval, the Bernoulli distribution for binary data, the Negative Binomial distribution for count data, as well as mixtures for distributions when the statistical properties are more complex. The parameters of the likelihood are computed from the network output by appropriate transformations, for example in the Gaussian case the variance is computed by an affine transformation followed by a softplus transformation of the network output to ensure positivity.

Training

Given a dataset of time series $\{\mathbf{z}_{i,1:T}\}_{i=1,\dots,N}$ and associated covariates $\mathbf{x}_{i,1:T}$ the model is trained by maximizing the log-likelihood of the observed data,

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{t=t_0}^{T} \log \ell(z_{i,t} \mid \theta(\mathbf{h}_{i,t})) \quad (2.23)$$

with respect to the parameters of the RNN $h(\cdot)$ and the parameters of $\theta(\cdot)$ using stochastic gradient descent.

For each time series in the dataset, multiple training instances are generated by selecting windows with different starting points from the original time series. In practice, the total length T, as well as the relative length of the conditioning and prediction ranges, are kept fixed for all training examples. When selecting these windows, the goal is to ensure that the entire prediction range is always covered by the available ground truth data. It is possible to choose $t = 1$ to be before the start of the time series and left-pad with zeros, allowing the model to learn the behavior of a "new" time series while considering all other available features. By augmenting the data using this windowing procedure, the model receives information about absolute time only through covariates, rather than through the relative position of $z_{i,t}$ in the time series.

Scale Handling

Datasets that exhibit a power-law of scales, present two main challenges. The first challenge is that the autoregressive input $z_{i,t-1}$ and the network output (e.g. μ) in the model are directly influenced by the observations $z_{i,t}$, but the non-linearities of the network have a limited operating range. Therefore, the network needs to learn how to scale the input appropriately in the input layer and then invert this scaling at the output. This issue is addressed by using an item-dependent scaling factor ν_i that is used to divide the input and multiply the output (by adjusting the distribution parameters appropriately). This scaling factor is typically chosen to be the heuristic $\nu_i = 1 + \frac{1}{t_0} \sum_{t=1}^{t_0} z_{i,t}$. The second challenge arises from the imbalance of the data. This is usually counteracted by sampling non-uniformly from the dataset.

2.3.3 Strengths and Limitations

DeepAR's ability to generate probabilistic forecasts provides valuable insights into future uncertainties, enabling effective risk management. The model effectively handles varying magnitudes in time series data through techniques like input scaling, making it particularly useful for datasets with skewed distributions. Extensive empirical testing on real-world datasets has consistently demonstrated the versatility and accuracy of DeepAR in producing reliable forecasts.

The sophistication of DeepAR may lead to more complex model tuning and require higher computational resources compared to simpler models. For optimal performance, DeepAR relies on the availability of large datasets with multiple related time series, which may not always be available. As with many deep learning models, the interpretability of DeepAR can be limited, making it difficult to understand the model's decision-making process.

2.4 DeepVAR

2.4.1 Model Overview

DeepVAR [8] extends the DeepAR model by jointly forecasting multiple related time series, explicitly capturing the dependencies among them through a joint distribution. This vector autoregressive approach enhances forecasting accuracy for systems where the time series influence one another, such as anomaly detection, demand forecasting, traffic networks and power grids.

The model combines an RNN-based time series model with a Gaussian copula process [44] output model utilizing a low-rank-plus-diagonal covariance structure to handle non-Gaussian marginal distributions. This approach significantly reduces computational complexity and enables the modeling of time-varying correlations among thousands of time series by reducing the number of parameters.

In forecasting tasks, it is common to predict multiple related time series, such as various metrics for a compute fleet or multiple products within the same category for demand forecasting. However, in high-dimensional settings, estimating large covariance matrices poses a challenge, leading to the assumption of conditional independence among these time series. However, in scenarios where the correlations between time series are significant, assuming independence is not appropriate. For instance, in anomaly detection, the simultaneous deviation of multiple nodes from their expected behavior can be a cause for concern, even if no individual node exhibits clear signs of anomalous behavior.

Classical univariate methods have been expanded to accommodate multivariate analysis, with vector autoregressions (VAR) expanding upon autoregressive models [45], alongside the development of multivariate state-space models [46] and multivariate generalized autoregressive conditional heteroskedasticity (MGARCH) [47] models. However, as the dimensionality of these problems increases, so does the complexity of estimating these models, primarily due to a surge in the number of parameters. This complexity has historically limited their application to low-dimensional scenarios. To overcome these challenges, researchers have implemented dimensionality reduction techniques and regularization strategies, particularly for VAR [48, 49] and MGARCH [50, 51] models. Despite these efforts, these models still struggle to handle applications that involve more than a few hundred dimensions [52].

Deep Learning approaches usually fit a global (i.e. shared) sequence-to-sequence model to a collection of time series but generate statistically independent predictions.

DeepVAR solves the issue of the quadratic complexity of estimating the covariance matrix by utilizing the low-rank plus diagonal covariance structure of the factor analysis model [53, 54, 55, 56] in combination with Gaussian copula processes [44] to an LSTM [41] to jointly learn temporal dynamics and the time-varying covariance structure, while significantly reducing the number of parameters that need to be estimated.

The challenge of handling varying magnitudes in time series, as discussed in Section 2.3, is tackled by separately modeling the marginal distribution of each time series using a non-parametric estimate of its cumulative distribution function (CDF). By utilizing this CDF estimate as the marginal transformation in a Gaussian copula, the issue of scaling is effectively addressed, as it allows for the independent estimation of marginal distributions while considering the temporal dynamics and dependency structure.

2.4.2 Model Architecture

Overall Structure

DeepVAR tries to estimate the conditional distribution $P\left(\mathbf{z}_{T+1},\ldots,\mathbf{z}_{T+\tau} \mid \mathbf{z}_1,\ldots\mathbf{z}_T\right)$, where $\mathbf{z}_t$ is the vector of time series values at time t, $[1,T]$ is the conditioning range and $[T+1,\tau]$ the prediction range. The model is structured as a non-linear, deterministic state space model, where the state $\mathbf{h}_{i,t}$ evolves independently for each time series i based on the transition dynamics φ.

$$\mathbf{h}_{i,t} = \varphi_{\theta_h}(\mathbf{h}_{i,t-1}, z_{i,t-1}), \quad i = 1,\ldots,N \tag{2.24}$$

The transition dynamics φ are parameterized using an LSTM [41]. The LSTM is unrolled for each time series separately, but the parameters are shared across all the time series.

Given the state values $\mathbf{h}_{i,t}$ for all time series $i = 1,\ldots,N$, and denoting with $\mathbf{h}_t$ the collection of state values for all series at time t, the joint emission distribution is parametrized using a Gaussian copula,

$$p(\mathbf{z}_t \mid \mathbf{h}_t) = \mathcal{N}\left([f_1(z_{1,t}),\ldots,f_N(z_{N,t})]^T \mid \boldsymbol{\mu}(\mathbf{h}_t),\, \Sigma(\mathbf{h}_t)\right) \tag{2.25}$$

where $f_i = \Phi^{-1} \circ \hat{F}_i$ with Φ^{-1} being the inverse of the standard normal CDF and $\hat{F}_i$ being an estimate of the CDF of the marginal distribution of the i-th time series $z_{i,1},\ldots,z_{i,T}$. The functions $\boldsymbol{\mu}(\cdot)$ and $\Sigma(\cdot)$ map the state $\mathbf{h}_t$ to the mean and covariance of a Gaussian distribution over the transformed observations.

The joint distribution of the observations is assumed to be factorized as

$$p(\mathbf{z}_1,\ldots,\mathbf{z}_{T+\tau}) = \prod_{t=1}^{T+\tau} p(\mathbf{z}_t \mid \mathbf{z}_1,\ldots,\mathbf{z}_{t-1}) = \prod_{t=1}^{T+\tau} p(\mathbf{z}_t \mid \mathbf{h}_t) \tag{2.26}$$

The state update function φ and the mappings $\boldsymbol{\mu}(\cdot)$ and $\Sigma(\cdot)$ have free parameters θ_φ, θ_μ and θ_Σ respectively that are learned from the data. The vector of all free parameters is denoted by θ. Given θ and $\mathbf{h}_{T+1}$, Monte Carlo samples can be produced from the joint distribution

$$p(\mathbf{z}_{T+1},\ldots,\mathbf{z}_{T+\tau} \mid \mathbf{z}_1,\ldots,\mathbf{z}_T) = p(\mathbf{z}_{T+1},\ldots,\mathbf{z}_{T+\tau} \mid \mathbf{h}_{T+1}) = \prod_{t=T+1}^{T+\tau} p(\mathbf{z}_t \mid \mathbf{h}_t) \tag{2.27}$$

by sequentially sampling from $P(\mathbf{z}_t \mid \mathbf{h}_t)$ and updating the state $\mathbf{h}_t$ using the state update function φ. The free parameters θ are learned by maximizing the log-likelihood of the observed data $\mathbf{z}_1,\ldots,\mathbf{z}_T$, that is by minimizing the loss function

$$-\log p(\mathbf{z}_1,\ldots,\mathbf{z}_T) = -\sum_{t=1}^{T} \log p(\mathbf{z}_t \mid \mathbf{h}_t) \tag{2.28}$$

using stochastic gradient descent. For long time series, the model is trained on random slices of fixed length L and predictions are computed using the first $L-\tau$ time points.

The Gaussian Copula

Each summand of the log-likelihood $p(\mathbf{z} \mid \mathbf{h})$ in equation 2.28 (t is omitted for brevity) can be computed by a simple formula, whose derivation is given below.

A copula function $C : [0,1]^N \to [0,1]$ is the CDF of a multivariate distribution with uniform marginals. Sklar's theorem [57] states that any multivariate distribution can be represented by a copula function and its marginals, and that if the marginals are continuous, the representation is unique, and it is given by the formula $F(\mathbf{z}) = C(F_1(z_1), \ldots, F_N(z_N))$, where F_i is the CDF of the i-th marginal distribution. (Note that $F(X) \sim \mathcal{U}(0,1)$ if F is the CDF of a continuous random variable X.)

A common modeling choice for the copula function is the Gaussian copula, which is given by the formula $C(F_1(z_1), \ldots, F_1(z_1)) = \phi_{\boldsymbol{\mu},\Sigma}(\Phi^{-1}(F_1(z_1)), \ldots, \Phi^{-1}(F_N(z_N)))$, where $\phi_{\boldsymbol{\mu},\Sigma}$ is the CDF of the multivariate normal distribution with mean $\boldsymbol{\mu}$ and covariance matrix Σ, and Φ is the CDF of the standard normal distribution. The variables are then related in the following manner:

$$\mathbf{x} \overset{\Phi}{\mapsto} \mathbf{u} \overset{F^{-1}}{\mapsto} \mathbf{z} \qquad \mathbf{z} \overset{F}{\mapsto} \mathbf{u} \overset{\Phi^{-1}}{\mapsto} \mathbf{x} \tag{2.29}$$

This is how the formula in 2.25 is derived. Because the marginals F_i are not known, they are replaced by their linearly interpolated (for piecewise differentiability) empirical estimates $\hat{F}_i(v) = \frac{1}{m}\sum_{t=1}^{m} \mathbb{1}_{z_{i,t} \leq v}$. The sample size m used to estimate the empirical CDFs is a hyperparameter of the model, but a size of 100 is typically a good choice.

This allows for a derivation of the log-likelihood formula of each summand in equation 2.28:

$$\begin{aligned}
\log p(\mathbf{z}; \boldsymbol{\mu}, \Sigma) &= \log \phi_{\boldsymbol{\mu},\Sigma}(\Phi^{-1}(\hat{F}(\mathbf{z}))) + \log \frac{d}{d\mathbf{z}}\Phi^{-1}(\hat{F}(\mathbf{z})) && (2.30)\\
&= \log \phi_{\boldsymbol{\mu},\Sigma}(\Phi^{-1}(\hat{F}(\mathbf{z}))) + \log \frac{d}{d\mathbf{u}}\Phi^{-1}(\mathbf{u}) + \log \frac{d}{d\mathbf{z}}\hat{F}(\mathbf{z}) && (2.31)\\
&= \log \phi_{\boldsymbol{\mu},\Sigma}(\Phi^{-1}(\hat{F}(\mathbf{z}))) - \log \phi(\boldsymbol{\Phi}^{-1}(\hat{F}(\mathbf{z}))) + \log \hat{F}'(\mathbf{z}) && (2.32)
\end{aligned}$$

where ϕ is the probability density function of the standard normal distribution.

Covariance Structure

Estimating a covariance matrix of full rank is computationally expensive in high-dimensional settings. DeepVAR addresses this issue by using a low-rank-plus-diagonal parametrization of the covariance. For notational simplicity, let $\mathbf{x}_t := f(\mathbf{z}_t) = [f_1(z_{1,t}), \ldots, f_N(z_{N,t})]$. We then have $p(\mathbf{x_t} \mid \mathbf{h}_t) = \mathcal{N}(\mathbf{x}_t \mid \boldsymbol{\mu}(\mathbf{h_t}), \Sigma(\mathbf{h}_t))$. The covariance matrix $\Sigma(\mathbf{h}_t)$ is parametrized as

$$\Sigma(\mathbf{h}_t) = \mathbf{D}(\mathbf{h}_t) + \mathbf{V}(\mathbf{h}_t)\mathbf{V}(\mathbf{h}_t)^T \tag{2.33}$$

where

$$\mathbf{D}(\mathbf{h}_t) = \begin{bmatrix} d_1(\mathbf{h}_{1,t}) & & 0 \\ & \ddots & \\ 0 & & d_N(\mathbf{h}_{N,t}) \end{bmatrix} \in \mathbb{R}^{N \times N} \tag{2.34}$$

$$\mathbf{V}(\mathbf{h}_t) = \begin{bmatrix} \mathbf{v}_1(\mathbf{h}_{1,t}) \\ \vdots \\ \mathbf{v}_N(\mathbf{h}_{N,t}) \end{bmatrix} \in \mathbb{R}^{N \times r} \tag{2.35}$$

The mappings μ_i, d_i and $\mathbf{v}_i$ are parametetrized in terms of shared functions $\tilde{\mu}$, $\tilde{d}$ and $\tilde{\mathbf{v}}$ that depend on an E-dimensional feature vector $\mathbf{e}_i \in \mathbb{R}^E$ for each individual time series i. These vectors can be known covariates or learned embeddings. By concatenating the state and the feature vectors $\mathbf{y}_{i,t} := [\mathbf{h}_{i,t}; \mathbf{e}_i]^T \in \mathbb{R}^p$ the parametrization is as follows:

$$\mu_i(\mathbf{h}_{i,t}) = \tilde{\mu}(\mathbf{y}_{i,t}) = \mathbf{w}_\mu^T \mathbf{y}_{i,t}, \tag{2.36}$$

$$d_i(\mathbf{h}_{i,t}) = \tilde{d}(\mathbf{y}_{i,t}) = s(\mathbf{w}_d^T \mathbf{y}_{i,t}), \tag{2.37}$$

$$\mathbf{v}_i(\mathbf{h}_{i,t}) = \tilde{\mathbf{v}}(\mathbf{y}_{i,t}) = W_\mathbf{v} \mathbf{y}_{i,t}, \tag{2.38}$$

where $s(x) = \log(1 + e^x)$ maps to positive values, and $\mathbf{w}_\mu \in \mathbb{R}^{p \times 1}$, $\mathbf{w}_d \in \mathbb{R}^{p \times 1}$, $W_\mathbf{v} \in \mathbb{R}^{r \times p}$ are parameters.

All learnable parameters are shared across all the time series and thus the distribution of $\mathbf{x}_t$ can be viewed as a Gaussian Process evaluated at points $\mathbf{y}_{i,t}$; symbolically:

$$\mathbf{x}_{i,t} = g_t(\mathbf{y_{i,t}}) \tag{2.39}$$

$$g_t \sim \mathcal{GP}(\tilde{\mu}(\cdot), k(\cdot, \cdot)) \tag{2.40}$$

$$k(\mathbf{y}, \mathbf{y}') = \mathbb{1}_{\mathbf{y}=\mathbf{y}'} \tilde{d}(\mathbf{y}) + \tilde{\boldsymbol{v}}(\mathbf{y})^T \tilde{\boldsymbol{v}}(\mathbf{y}') \tag{2.41}$$

By considering this perspective, we can observe that the model can be trained by calculating the Gaussian terms in the loss function on randomly selected subsets of the time series during each iteration. In other words, we can train the model using batches of size $B \ll N$. Additionally, if there is prior knowledge about the covariance structure, such as in the case of spatial data where the covariance is related to the distance between points, this information can be easily incorporated into the kernel. This can be done by either using a pre-specified kernel exclusively or by combining it with the learned, time-varying kernel mentioned earlier.

2.4.3 Strengths and Limitations

DeepVAR's strength lies in its ability to capture the dependencies among multiple related time series efficiently, making it suitable when parallel time series share significant information.

The LSTM component of the model can slow down its speed due to its non-parallelizable

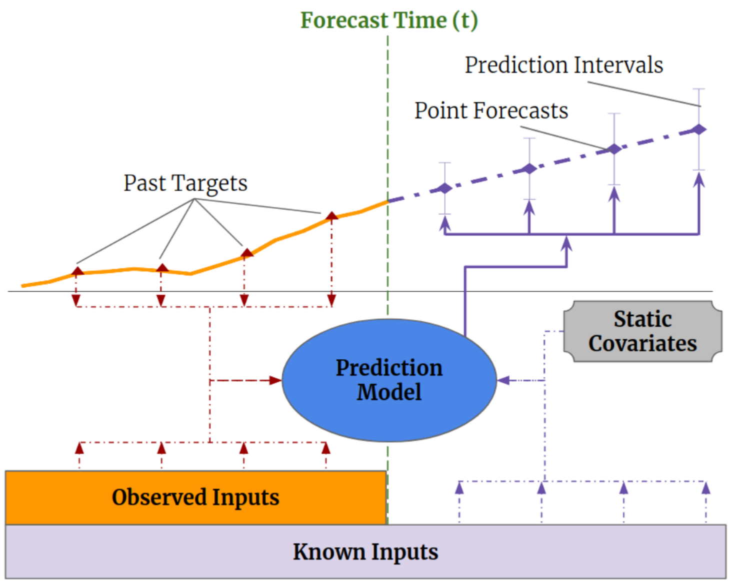


Figure 2.7. *Illustration of multi-horizon forecasting with static covariates, past-observed and apriori-known future time-dependent inputs. Figure from [6].*

nature. Additionally, the model's interpretability is constrained, and it often requires a substantial amount of data to achieve optimal performance, which can be difficult in domains with limited or expensive data availability.

2.5 Temporal Fusion Transformer (TFT)

2.5.1 Model Overview

The Temporal Fusion Transformer (TFT) [6] is an architecture that uses attention mechanisms to achieve effective multi-horizon forecasting, i.e. the prediction of variables of interest at multiple future time steps, and provides interpretable insights into temporal dynamics. It employs recurrent layers for analyzing short-term patterns, and self-attention layers for understanding long-term relationships. The TFT also includes specific components for choosing important features and gating layers to eliminate non-essential elements, which helps it perform well in various situations.

In practical multi-horizon forecasting scenarios, it is common to have access to various data sources, as depicted in Figure 2.7. These sources include information about the future (such as upcoming holiday dates), other external time series (such as historical customer foot traffic), and static metadata (such as the location of a store). However, there is often limited knowledge about how these different data sources interact with each other. This diversity of data sources, coupled with the lack of information about their interactions, poses significant challenges for multi-horizon time series forecasting.

In many Deep Neural Network (DNN) forecasting architectures, the consideration of different types of inputs commonly found in multi-horizon forecasting is often overlooked. These architectures either assume that all exogenous inputs are known in advance (a com-

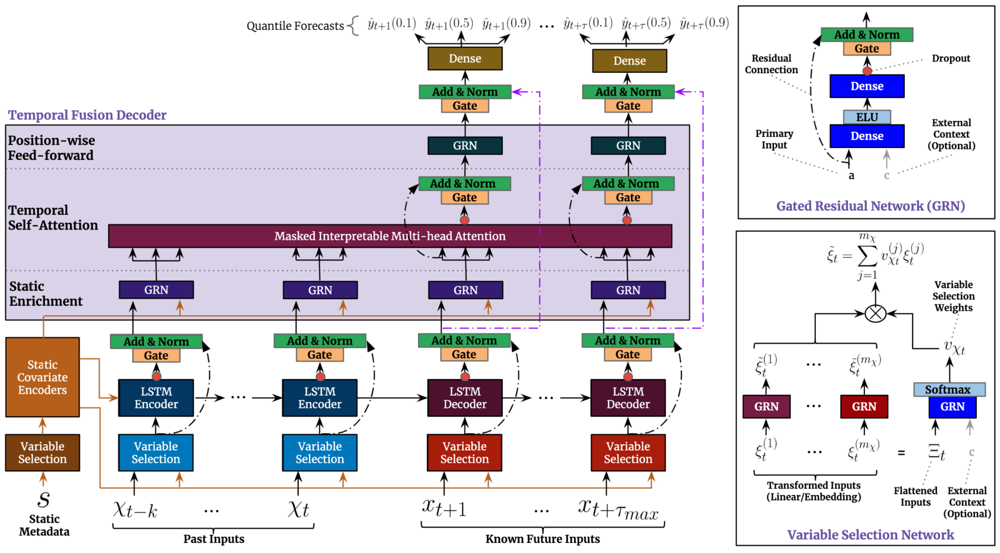


Figure 2.8. *The architecture of the Temporal Fusion Transformer. Figure from [6].*

mon assumption in autoregressive models) or neglect important static covariates, which are simply concatenated with other time-dependent features at each step. Additionally, Most DNN architectures are considered “black-box” models, as they rely on intricate nonlinear interactions among numerous parameters. Consequently, explaining the reasoning behind model predictions becomes challenging, making it difficult for users to trust the model’s outputs and for developers to debug it. Common explainability methods like LIME [58] and SHAP [59], in their conventional form, do not account for the temporal ordering of input features.

The Temporal Fusion Transformer tackles these challenges through the integration of static covariate encoders that generate context vectors for utilization across the network, the employment of gating mechanisms and sample-dependent variable selection to reduce the impact of non-essential inputs, the inclusion of a sequence-to-sequence layer for processing both known and observed inputs on a local scale, and the implementation of a temporal self-attention decoder designed to uncover and learn from any long-term dependencies within the dataset. This allows for the identification of globally significant variables for the prediction task, persistent temporal patterns, and significant events.

2.5.2 Model Architecture

A time series dataset can be viewed as being composed of I distinct entities (e.g. different stores in retail, or patients in healthcare), each associated with a set of static covariates $\mathbf{s}_i \in \mathbb{R}^{m_s}$, as well as inputs $\chi_{i,t} \in \mathbb{R}^{m_\chi}$ and scalar targets $y_{i,t} \in \mathbb{R}$ at each time-step $t \in [0, T_i]$. Inputs that vary over time are subdivided into two categories $\chi_{i,t} = \left[\mathbf{z}_{i,t}^T, \mathbf{x}_{i,t}^T\right]^T$: observed inputs $\mathbf{z}_{i,t} \in \mathbb{R}^{m_z}$, which can only be measured at each step and are unknown beforehand, and known inputs $\mathbf{x}_{i,t} \in \mathbb{R}^{m_x}$, which can be predetermined (e.g. the day-of-week at time t).

TFT does quantile regression (for example outputting the 10th, 50th, and 90th percentiles at each time step). Each quantile forecast takes the form:

$$\hat{y}_i(q,t,\tau) = f_q(\tau, y_{i,t-k:t}, \mathbf{z}_{i,t-k:t}, \mathbf{x}_{i,t-k:t+\tau}, \mathbf{s}_i), \tag{2.42}$$

where $\hat{y}_{i,t+\tau}(q,t,\tau)$ is the predicted q^{th} sample quantile of the τ-step-ahead forecast at time t, and $f_q(\cdot)$ is a prediction model. Forecasts are outputted simultaneously for $\tau_{\max}$ time steps and all past information is incorporated within a finite look-back window k, using target and known inputs up till and including the forecast start time t, and known inputs across the entire range.

The major components of TFT are:

1. *Gating mechanisms* that filter any non-essential parts of the structure, thus adapting the depth and complexity of the network to fit a broad spectrum of datasets and situations.

2. *Variable selection networks* implemented to select the most relevant input variables at every timestep.

3. *Static covariate encoders* designed to assimilate static attributes into the network by generating context vectors that modulate the temporal dynamics.

4. *Temporal processing* capabilities that facilitate the learning of both long-term and short-term temporal relationships derived from both observed and known time-varying inputs. This involves the use of a sequence-to-sequence layer for local processing, while an interpretable multi-head attention block is utilized to learn long-term dependencies.

5. *Prediction intervals* through quantile forecasts to ascertain the probable range of target outcomes at each forecast horizon.

The high-level structure of TFT is shown in Figure 2.8.

Gating Mechanisms

Determining the precise relationship between exogenous inputs and desired outputs is not always straightforward, as it is not always evident beforehand which variables will be pertinent. It is equally challenging to ascertain the degree of non-linear processing that is necessary, and there are cases where simpler models may be more suitable, for instance, when datasets are small or noisy. To provide a model with the versatility to engage non-linear processing selectively and only as needed, the Gated Residual Network (GRN) is introduced (depicted in Figure 2.8) as an integral element of the Temporal Fusion Transformer (TFT). The GRN accepts a primary input **a** along with an optional context vector $\mathbf{c}$, resulting in:

$$\mathrm{GRN}_\omega(\mathbf{a},\mathbf{c}) = \mathrm{LayerNorm}(\mathbf{a} + \mathrm{GLU}_\omega(\boldsymbol{\eta}_1)) \tag{2.43}$$

$$\boldsymbol{\eta}_1 = \mathbf{W}_{1,\omega}\boldsymbol{\eta}_2 + \mathbf{b}_{1,\omega} \tag{2.44}$$

$$\boldsymbol{\eta}_2 = \mathrm{ELU}(\mathbf{W}_{2,\omega}\mathbf{a} + \mathbf{W}_{3,\omega}\mathbf{c} + \mathbf{b}_{2,\omega}) \tag{2.45}$$

where ELU is the Exponential Linear Unit activation function [60], $\boldsymbol{\eta}_1, \boldsymbol{\eta}_2 \in \mathbb{R}^{d_{\text{model}}}$, are layers in between, LayerNorm is a standard layer normalization [61], and ω is an index to denote weight sharing. When $\mathbf{W}_{2,\omega}\mathbf{a} + \mathbf{W}_{3,\omega}\mathbf{c} + \mathbf{b}_{2,\omega}$ is significantly positive ($\gg 0$), the ELU function behaves similarly to an identity function, and when it is considerably negative ($\ll 0$), it will generate a constant output. Gating layers based on Gated Linear Units (GLUs) [62] offer the flexibility to minimize any superfluous components of the architecture for a given dataset. Assuming $\boldsymbol{\gamma}$ to be the input, the GLU is defined as:

$$\mathrm{GLU}_\omega(\boldsymbol{\gamma}) = \sigma(\mathbf{W}_{4,\omega}\boldsymbol{\gamma} + \mathbf{b}_{4,\omega}) \odot (\mathbf{W}_{5,\omega}\boldsymbol{\gamma} + \mathbf{b}_{5,\omega}), \tag{2.46}$$

where $\sigma(\cdot)$ is the sigmoid activation function, $\mathbf{W}_{(\cdot)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$, $\mathbf{b}_{(\cdot)} \in \mathbb{R}^{d_{\text{model}}}$ represent the weights and biases, $\odot$ symbolizes the element-wise Hadamard product, and d_{model} is the TFT's common hidden state size. The GLU allows for the modulation of the GRN's impact on the original input $\mathbf{a}$, potentially skipping over the layer entirely if necessary as the GLU outputs could be all close to 0 in order to suppress the nonlinear contribution. For instances without a context vector, the GRN simply treats the context input as zero. During training, dropout is applied before the gating layer and layer normalization – i.e. to $\boldsymbol{\eta}_1$.

Variable Selection Networks

Although numerous variables may be accessible, their significance and specific contribution to the output are often indeterminate. TFT is designed to perform instance-wise variable selection via variable selection networks that apply to both static covariates and time-dependent covariates. This selection process provides insights into which variables are paramount for the predictive task at hand. Additionally, it allows TFT to discard any unnecessary and noisy inputs that might adversely affect performance. Real-world time series datasets often contain features with varying levels of predictive relevance; hence, variable selection can optimize model performance by leveraging learning capacity primarily on the most significant variables.

Entity embeddings [63] are utilized for categorical variables to serve as feature representations, and perform linear transformations on continuous variables, transforming each variable into a $\mathbb{R}^{d_{\text{model}}}$-dimensional vector that aligns with the dimensionality required for subsequent layers, including skip connections. TFT uses separate variable selection networks for static, past and future inputs, each denoted by distinct colors in Figure 2.8. Without loss of generality, the variable selection network for past inputs is presented below; for future and static inputs the form is the same:

Let $\boldsymbol{\xi}_t^{(j)} \in \mathbb{R}^{d_{\text{model}}}$ be the transformed input of the j-th variable at time t, with $\boldsymbol{\Xi}_t = [\xi_t^{(1)^T}, \ldots, \xi_t^{(m_\chi)^T}]^T$ representing the concatenated vector of all past inputs at time t. The variable selection weights are generated by processing both $\boldsymbol{\Xi}_t$ and an external context vector $\mathbf{c}_s$ through a GRN, followed by a softmax layer:

$$\boldsymbol{v}_{\chi_t} = \text{Softmax}(\text{GRN}_{\boldsymbol{v}_\chi}(\boldsymbol{\Xi}_t, \mathbf{c}_s)), \tag{2.47}$$

where $\boldsymbol{v}_{\chi_t} \in \mathbb{R}^{m_\chi}$ is a vector of variable selection weights, and $\mathbf{c}_s$ is derived from a static covariate encoder. For static variables, the context vector $\mathbf{c}_s$ is omitted since they already have access to static information. At each time step, an additional layer of non-linear processing is implemented by passing each $\boldsymbol{\xi}_t^{(j)}$ through its own GRN:

$$\tilde{\boldsymbol{\xi}}_t^{(j)} = \text{GRN}_{\tilde{\boldsymbol{\xi}}^{(j)}}(\boldsymbol{\xi}_t^{(j)}), \tag{2.48}$$

where $\tilde{\boldsymbol{\xi}}_t^{(j)}$ is the processed feature vector for variable j. Notably, each variable has its GRN with weights shared across all time steps t. The processed features are then weighted by their variable selection weights and combined as follows:

$$\tilde{\boldsymbol{\xi}}_t = \sum_{j=1}^{m_\chi} v_{\chi_t}^{(j)} \tilde{\boldsymbol{\xi}}_t^{(j)}, \tag{2.49}$$

where $v_{\chi_t}^{(j)}$ is the j-th element of the vector $\boldsymbol{v}_{\chi_t}$.

Static Covariate Encoders

Differing from other time series forecasting models, the Temporal Fusion Transformer (TFT) is architected to assimilate information from static metadata through the utilization of separate GRN encoders. These encoders are tasked with generating four distinct context vectors, $\mathbf{c}_s$, $\mathbf{c}_e$, $\mathbf{c}_c$, and $\mathbf{c}_h$. These vectors are integrated into various junctures within the temporal fusion decoder, where static variables are crucial to processing the sequence. This includes the provision of contexts for temporal variable selection ($\mathbf{c}_s$), localized processing of temporal features ($\mathbf{c}_c, \mathbf{c}_h$), and enrichment of temporal features with static information ($\mathbf{c}_e$). For instance, considering $\boldsymbol{\zeta}$ to be the output of the static variable selection network, contexts for temporal variable selection would be formulated as $\mathbf{c}_s = \text{GRN}_{\mathbf{c}_s}(\boldsymbol{\zeta})$.

Interpretable Multi-Head Attention

The Temporal Fusion Transformer (TFT) employs a self-attention mechanism for capturing long-term dependencies across various time steps, adapted from the multi-head attention mechanism found in transformer-based architectures [64, 65] to enhance explainability. Generally, attention mechanisms scale values $\mathbf{V} \in \mathbb{R}^{N \times d_V}$ based on the relationships inferred from keys $\mathbf{K} \in \mathbb{R}^{N \times d_{\text{attn}}}$ and queries $\mathbf{Q} \in \mathbb{R}^{N \times d_{\text{attn}}}$ as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{A}(\mathbf{Q}, \mathbf{K})\mathbf{V} \tag{2.50}$$

where A denotes a normalization function, often chosen to be the scaled dot-product attention [64]:

$$\text{A}(\mathbf{Q}, \mathbf{K}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_{attn}}}\right). \tag{2.51}$$

To enhance the learning capability of this standard attention mechanism, TFT utilizes multi-head attention [64], deploying different heads across distinct representation subspaces:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = [\mathbf{H}_1, \ldots, \mathbf{H}_{m_H}]\mathbf{W}_H \tag{2.52}$$

$$\mathbf{H}_h = \text{Attention}(\mathbf{Q}\mathbf{W}_Q^{(h)}, \mathbf{K}\mathbf{W}_K^{(h)}, \mathbf{V}\mathbf{W}_V^{(h)}) \tag{2.53}$$

where $\mathbf{W}_Q^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $\mathbf{W}_K^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{attn}}}$, $\mathbf{W}_V^{(h)} \in \mathbb{R}^{d_{\text{model}} \times d_V}$ are head-specific weights for queries, keys, and values, and $\mathbf{W}_H \in \mathbb{R}^{(m_H \cdot d_V) \times d_{\text{model}}}$ linearly combines outputs from all heads $\mathbf{H}_h$.

However, when different values are used in each head, attention weights alone might not be indicative of a particular feature's importance. Therefore, TFT modifies multi-head attention to share values in each head and uses additive aggregation of all heads:

$$\text{InterpretableMultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \tilde{\mathbf{H}}\mathbf{W}_H, \tag{2.54}$$

$$\begin{aligned}
\tilde{\mathbf{H}} &= \tilde{\text{A}}(\mathbf{Q}, \mathbf{K})\mathbf{V}\mathbf{W}_V && (2.55)\\
&= \left\{\frac{1}{H}\sum_{h=1}^{m_H} \text{A}(\mathbf{Q}\mathbf{W}_Q^{(h)}, \mathbf{K}\mathbf{W}_K^{(h)})\right\}\mathbf{V}\mathbf{W}_V && (2.56)\\
&= \frac{1}{H}\sum_{h=1}^{m_H} \text{Attention}(\mathbf{Q}\mathbf{W}_Q^{(h)}, \mathbf{K}\mathbf{W}_K^{(h)}, \mathbf{V}\mathbf{W}_V) && (2.57)
\end{aligned}$$

where $\mathbf{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_V}$ are value weights shared across all heads, and $\mathbf{W}_H \in \mathbb{R}^{d_{\text{attn}} \times d_{\text{model}}}$ is used for linear mapping. Equation 2.56 shows that each head is capable of discerning diverse temporal patterns while attending to a shared group of input features. This process can be viewed as a simple ensemble over attention weights into the consolidated matrix $\tilde{\text{A}}(\mathbf{Q}, \mathbf{K})$ as illustrated in Equation 2.55. In contrast to $\text{A}(\mathbf{Q}, \mathbf{K})$ shown in Equation 2.51, $\tilde{\text{A}}(\mathbf{Q}, \mathbf{K})$ yields an increased representation capacity efficiently.

Temporal Fusion Decoder

Points of significance in a time series are often identified in the context of their surrounding values, such as anomalies, change points and seasonal patterns. Leveraging local context on top of point-wise values can thus lead to increased performance. For instance, [66] uses a convolutional layer to enhance locality, but this is not suitable when the past and future inputs differ in numbers. TFT addresses this by using a sequence-to-sequence model that naturally handles these differences by feeding $\tilde{\boldsymbol{\xi}}_{t-k:t}$ to the encoder and $\tilde{\boldsymbol{\xi}}_{t+1:t+\tau_{\max}}$ to the decoder. This then generates temporal features $\boldsymbol{\phi}(t,n),\ n=-k,\dots,\tau_{\max}$. The encoder and decoder are typically implemented by an LSTM network. This process also serves as a replacement for standard positional encoding, since it provides an inherent bias for the chronological sequence of the inputs. Static metadata is integrated into the local computation by initializing the first LSTM's cell and hidden states with $\mathbf{c}_e, \mathbf{c}_h$ context vectors derived from static covariate encoding. A gated skip connection is also employed over this layer:

$$\tilde{\boldsymbol{\phi}}(t,n) = \mathrm{LayerNorm}(\tilde{\boldsymbol{\xi}}_{t+n} + \mathrm{GLU}_{\tilde{\phi}}(\boldsymbol{\phi}(t,n))), \quad n=-k,\dots,\tau_{\max} \tag{2.58}$$

Static Enrichment Layer

Static covariate information is injected via a static enrichment layer:

$$\boldsymbol{\theta}(t,n) = \mathrm{GRN}_{\theta}\left(\tilde{\boldsymbol{\phi}}(t,n), \mathbf{c}_e\right), \quad n=-k,\dots,\tau_{\max} \tag{2.59}$$

where the weights of GRN_θ are shared across the entire layer and $\mathbf{c}_e$ is a context vector derived from static covariate encoding.

Temporal Self Attention Layer

All statically-enriched temporal features are grouped into a matrix $\boldsymbol{\Theta}(t) = [\boldsymbol{\theta}(t,-k),\dots,\boldsymbol{\theta}(t,\tau)]^T$ and interpretable multi-head attention is applied at each forecast time:

$$\begin{aligned}\mathbf{B}(t) &= \mathrm{InterpretableMultiHead}(\boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t), \boldsymbol{\Theta}(t)) \qquad (2.60)\\ &= [\boldsymbol{\beta}(t,-k),\dots,\boldsymbol{\beta}(t,\tau_{\max})] \qquad (2.61)\end{aligned}$$

With m_H being the number of heads, the dimensions are $d_V = d_{\mathrm{attn}} = d_{\mathrm{model}}/m_H$. Decoder masking [64, 66] is applied to the multi-head attention layer to ensure that each temporal dimension can only attend to features preceding it. The self-attention layer can learn long-term dependencies that the LSTM might not be able to. To facilitate training an additional gating layer is applied:

$$\boldsymbol{\delta}(t,n) = \mathrm{LayerNorm}(\boldsymbol{\theta}(t,n) + \mathrm{GLU}_{\delta}(\boldsymbol{\beta}(t,n))), \quad n=-k,\dots,\tau_{\max} \tag{2.62}$$

Position-wise Feed-forward Layer

Additional non-linear processing is applied to the output of the self-attention layer:

$$\boldsymbol{\psi}(t,n) = \mathrm{GRN}_{\psi}(\boldsymbol{\delta}(t,n)), \quad n = -k, \ldots, \tau_{\max} \tag{2.63}$$

where the weights of this GRN are shared across the entire layer. A gated residual connection that skips the entire transformer block is applied after the GRN to yield a simpler model if additional complexity is unnecessary:

$$\tilde{\boldsymbol{\psi}}(t,n) = \mathrm{LayerNorm}\left(\tilde{\boldsymbol{\phi}}(t,n) + \mathrm{GLU}_{\tilde{\psi}}(\boldsymbol{\psi}(t,n))\right), \quad n = -k, \ldots, \tau_{\max} \tag{2.64}$$

Quantile Ouputs

TFT generates prediction intervals, by outputting quantiles, e.g. the 10th, 50th, and 90th percentiles at each time step. Quantile forecasts are generated by an affine transformation on the output of the temporal fusion decoder:

$$\hat{y}_i(q,t,\tau) = \mathbf{W}_q \tilde{\boldsymbol{\psi}}(t,\tau) + b_q \tag{2.65}$$

where $\mathbf{W}_q \in \mathbb{R}^{1\times d}$, $b_q \in \mathbb{R}$ are specified per quantile q.

Loss Functions

TFT is trained jointly by minimizing the quantile loss [67] summed across all quantile outputs:

$$\mathcal{L}(\Omega, \mathbf{W}) = \sum_{y_t\in\Omega} \sum_{q\in\mathcal{Q}} \sum_{\tau=1}^{\tau_{\max}} \frac{\mathrm{QL}(y_t,\, \hat{y}(1, t-\tau, \tau),\, q)}{M\tau_{\max}} \tag{2.66}$$

$$\mathrm{QL}(y_t,\, \hat{y},\, q) = q(y-\hat{y})_+ + (1-q)(\hat{y}-y)_+ \tag{2.67}$$

where Ω is the domain of the training data containing M samples, $\mathbf{W}$ represents the weights of the TFT, $\mathcal{Q}$ is the set of output quantiles (a typical choice is $\mathcal{Q} = \{0.1, 0.5, 0.9\}$), and $(\cdot)_+ = \max(0, \cdot)$.

2.5.3 Strengths and Limitations

TFT's state-of-the-art strength stems from its ability to incorporate all types of inputs commonly found in multi-horizon forecasting, including static, past-observed and apriori-known covariates. This, along with its ability to model complex temporal relations by capturing both long-term and short-term dependencies, as well as its interpretability make it a very powerful and accurate tool for time series forecasting.

The complex architecture of TFT can also be a limiting factor, as it requires a substantial amount of data to achieve optimal performance, and it is quite computationally expensive to train and run.

Chapter 3

Datasets

3.1 Introduction

The efficacy of machine learning models in forecasting relies heavily on the quality, nature, and preprocessing of the datasets used for training and evaluation. This thesis evaluates several advanced forecasting models on two distinct datasets, each offering unique challenges and insights into the models' capabilities in handling real-world time series data. These datasets are the Electricity Load Diagrams dataset [9], concerned with electricity consumption, and the PeMS-SF dataset [10], concerned with traffic flow, both of which have been meticulously processed and modified for time series forecasting with the models of Chapter 2. This chapter provides an overview of the original datasets, detailing their nature and importance, it explains how they have been adapted and utilized in this thesis and explores their intricacies.

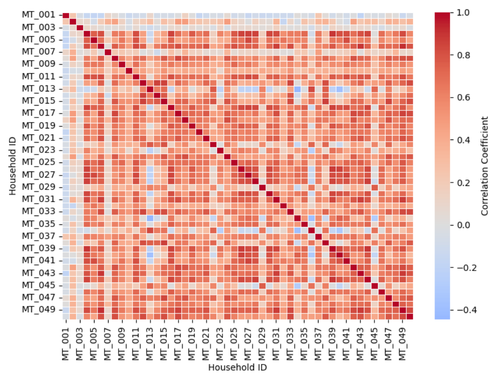
3.2 Electricity Load Diagrams Dataset

3.2.1 Original Dataset

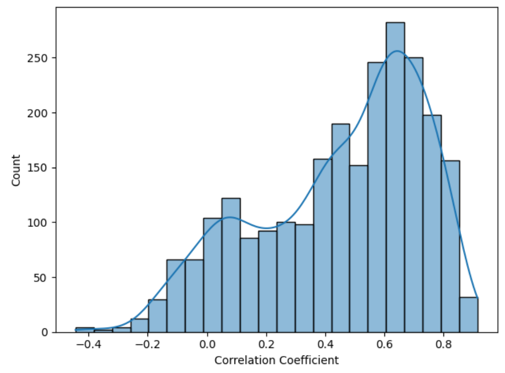
The Electricity Load Diagrams dataset [9] originates from the UCI Machine Learning Repository [68] and contains electricity consumption data in 15-minute intervals from 2011 to 2014. This dataset is composed of data from 370 clients, offering a comprehensive view of electricity demand patterns over various seasons and times of the day. Notably, some clients were created after 2011, thus the time series are not all time-parallel in the original dataset. The significance of this dataset lies in its reflection of real-world electricity usage patterns, making it an invaluable resource for forecasting electricity demand — a critical component in managing and optimizing energy distribution and generation.

3.2.2 Usage in This Thesis

For this thesis, 50 parallel time series have been selected and processed from the Electricity Load Diagrams dataset. The processed series span from January 2, 2014, to September 1, 2014, providing a focused period for analysis that includes variations in electricity demand across different seasons. To adapt the dataset for hourly forecasting, the original 15-



(**a**) *Correlation Heatmap*



(**b**) *Correlation Histogram (values of 1.0 excluded)*

Figure 3.1. *Electricity Load Diagrams Correlation*

minute interval data have been resampled to 1-hour intervals by averaging the values. This preprocessing step simplifies the dataset, making it more suitable for evaluating the forecasting models' performance over longer time horizons.

3.2.3 Exploratory Data Analysis

Exploratory Data Analysis (EDA) on the Electricity Load Diagrams dataset serves as the foundation for understanding the intricate patterns of electricity consumption across different clients and periods. This phase is critical for uncovering underlying trends, seasonalities, and anomalies in electricity usage, which are pivotal for accurate forecasting. The analysis begins with a comprehensive examination of the dataset's statistical properties, including distributions of electricity consumption rates, identification of peak demand periods, and assessment of variability across different days and months. This initial investigation aims to highlight the dataset's characteristics, such as daily and seasonal cycles, outliers, and long-term trends. By visualizing these features through time series plots, heatmaps, and boxplots, we gain invaluable insights into the data's structure, guiding the preprocessing and model selection phases of the thesis.

The first visualization presented in the exploratory data analysis is a correlation heatmap, providing a pairwise comparison of consumption patterns across different clients (Figure 3.1a). Each square in the heatmap represents the Pearson correlation coefficient between two households' electricity consumption. Red tones indicate positive correlations, suggesting similar usage patterns, while blue tones suggest inverse relationships. The strong red diagonal line confirms that each household's series is perfectly correlated with itself, as expected by the definition of the correlation. Notably, most households show a moderate to high positive correlation with each other (Figure 3.1b), which could imply a common underlying factor influencing consumption, such as time of the day or seasonality.

The second visualization is a heatmap showcasing the hourly electricity consumption for one week across different households (Figure 3.2). This visualization uses a color gradient to represent the log-transformed consumption values, where lighter colors correspond to

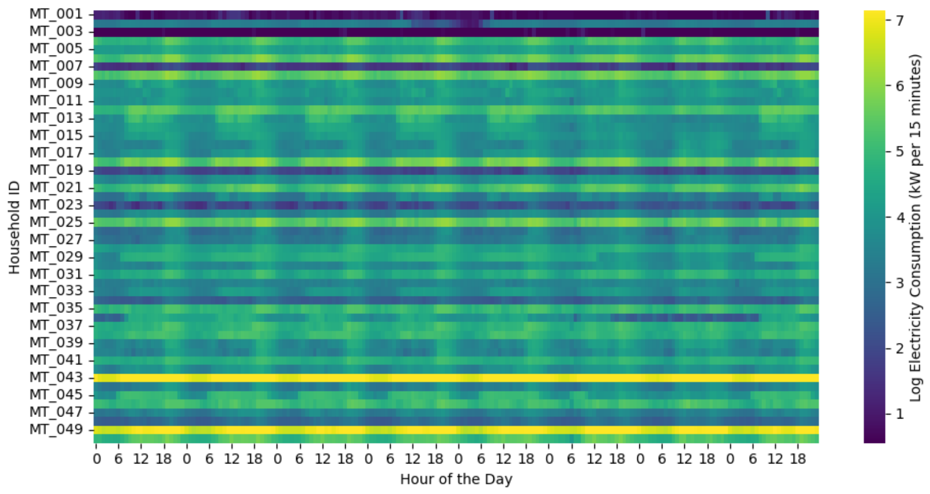


Figure 3.2. *Heatmap of Hourly Electricity Consumption. Data from Monday 2014–02–03 to Sunday 2014–02–09.*

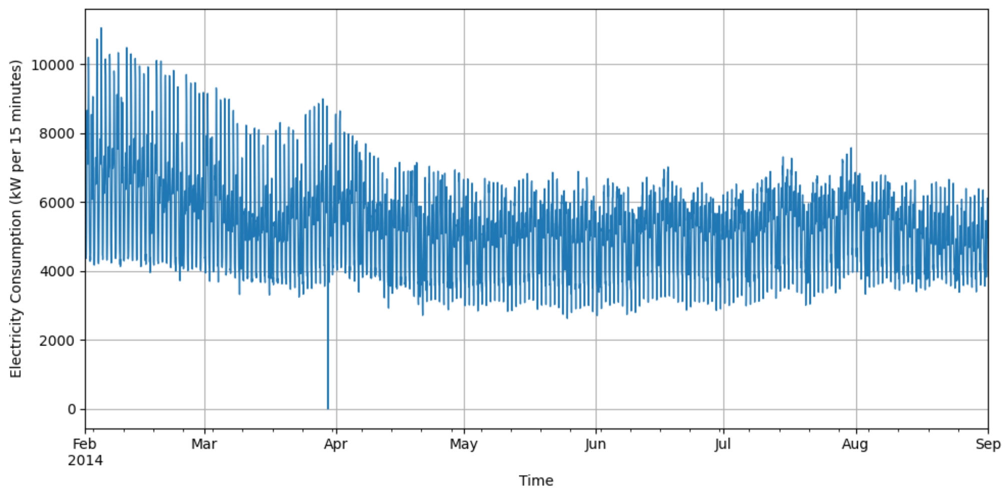
higher usage. The repeating patterns indicate the cyclical nature of electricity usage within a day and across the week. This pattern is especially evident in the consistent light vertical bands, likely representing evening peak usage times when residents are typically at home.

To delve deeper into the hourly patterns, boxplots disaggregate the data by hour (Figure 3.3b). These plots reveal the distribution of consumption at different times, highlighting the presence of outliers and the variability within each hour. The median consumption line within each boxplot indicates the central tendency, which, when observed in conjunction with the median hourly consumption line chart (Figure 3.3d), shows a clear pattern of peak and off-peak usage. The hourly median line chart plots the median value of electricity consumption across all households against the hours of the day, illustrating the typical daily consumption cycle with a steep rise in the morning, a slight dip in the afternoon, and a peak in the evening hours.

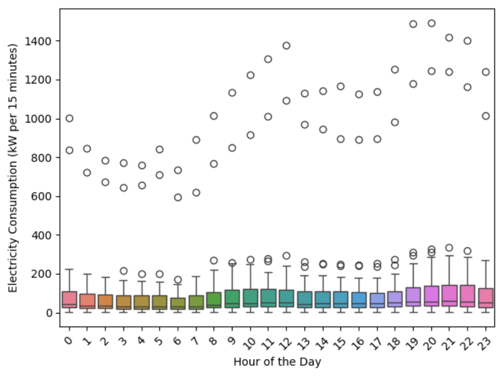
The weekly boxplot visualization (Figure 3.3c) breaks down electricity consumption by the day of the week, providing insights into how usage varies on weekdays versus weekends. The boxplots suggest a lower median and tighter distribution on weekends, which is corroborated by the median weekly electricity consumption line chart (Figure 3.3e). This final chart plots the median electricity consumption against the days of the week, showing a noticeable drop during the weekend, likely reflecting the reduced activity in commercial spaces and changes in residential patterns.

When examining the longer-term trends, a line chart of median electricity consumption over several months (Figure 3.3a) provides insights into broader consumption patterns. This chart shows fluctuations that could be associated with external factors such as temperature changes, holidays, or other seasonal effects. Notably, on the last Sunday of March, the Electricity consumption is zero, which is an artificial value due to the daylight saving time change.

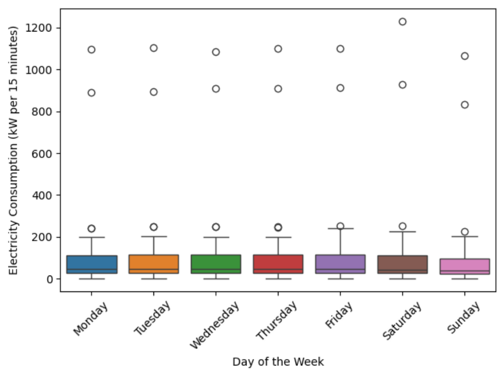
Together, these visualizations form a comprehensive EDA framework that highlights the



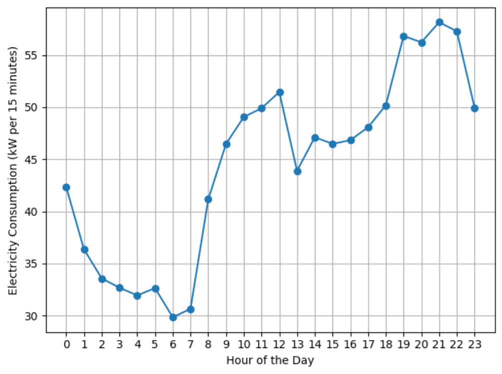
(a) *Median Electricity Consumption*



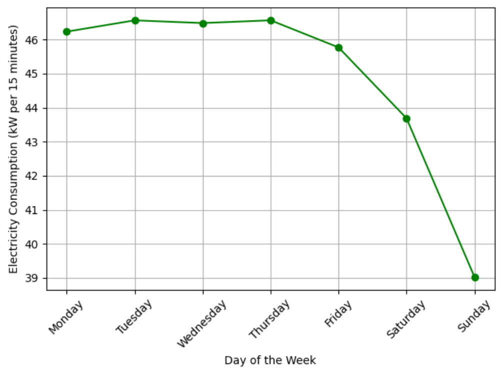
(b) *Hourly Boxplot of Electricity Consumption*



(c) *Weekly Boxplot of Electricity Consumption*



(d) *Median Hourly Electricity Consumption*



(e) *Median Weekly Electricity Consumption*

Figure 3.3. *Electricity Load Diagrams Seasonality Visualization*

complex nature of electricity consumption patterns. The diurnal and weekly cycles, the presence of outliers, and the impact of seasonal and weekly factors on electricity usage are all crucial considerations for any forecasting model. This analysis not only informs the preprocessing and feature engineering stages but also provides a baseline understanding against which the performance of the forecasting models can be evaluated.

3.3 PeMS-SF Dataset

3.3.1 Original Dataset

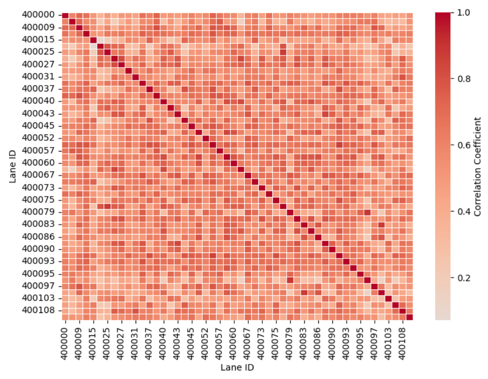
The PeMS-SF dataset [10], also sourced from the UCI Machine Learning Repository, originally serves for time series classification and stems from the Performance Measurement System (PeMS) in California. It comprises traffic occupancy rates collected from sensors in the freeway system of the San Francisco Bay area. Particularly, it includes data from 963 lanes from January 1st, 2008 to March 30th, 2009 with a sampling frequency of 10 minutes, and with missing values at four entire days (January 20, February 17, March 8, and May 25, 2008).

This dataset is originally structured for the task of classifying the day of the week, given data of a single day. For this purpose, it has partitioned the time series into days, which are then shuffled and split into training and testing sets, with each series having a label denoting its corresponding day of the week. The dataset also contains a file with information about the permutation that was used to shuffle the data, as well as a file with the IDs of each lane.

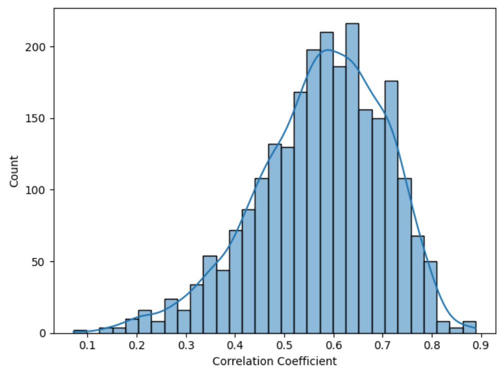
3.3.2 Usage in This Thesis

In this thesis, the PeMS-SF dataset has been repurposed for time series forecasting, focusing on a single time series for each car lane. The original time series can be reconstructed from the day of the week labels, the permutation, and the first date of the dataset (January 1st, 2008). More specifically, this reconstruction is achieved by concatenating the train and test sets, then applying the inverse permutation, and computing the timestamps through incrementing the starting date by the pairwise differences of the day of the week labels modulo 7.

From this dataset, the first 50 time series have been selected and processed, covering the period from January 1, 2008, to June 25, 2008. The dataset has been resampled to a 1-hour frequency by averaging the values, consistent with the Electricity Load Diagrams dataset, to facilitate hourly traffic forecasting. The missing values have been replaced with zeros for the DL models and left as is in the case of the Prophet model. This adjustment allows for a standardized approach to handling missing data across the models evaluated, with Prophet being the exception due to its inherent handling of missing values.



(a) *Correlation Heatmap*



(b) *Correlation Histogram (values of 1.0 excluded)*

Figure 3.4. *PeMS-SF Correlation Visualization*

3.3.3 Exploratory Data Analysis

The Exploratory Data Analysis for the PeMS-SF dataset focuses on understanding traffic flow dynamics and occupancy patterns within the freeway system of the San Francisco Bay area. This analysis is crucial for identifying the key features of traffic behavior, including rush hour peaks, weekly cycles, and the impact of holidays and special events on traffic volumes. By examining the dataset's statistical summaries, visualizing occupancy rates over time, and detecting periods of missing data, this EDA phase aims to uncover the complexities of traffic data that must be considered for effective forecasting. Special attention is given to the treatment of missing values and the identification of anomalies, which could significantly influence the performance of forecasting models. Through this detailed exploration, the analysis sets the stage for selecting appropriate data processing techniques and forecasting strategies that can handle the dataset's specific challenges.

The exploratory data analysis for the PEMS-SF dataset begins with an examination of the correlation between different lane occupancy rates (Figure 3.4a). The heatmap illustrates the Pearson correlation coefficients between the time series of different lanes. A predominance of warm colors suggests that many lanes exhibit similar traffic patterns, indicating possible relationships or dependencies between the flows of adjacent lanes or those within the same traffic control segments.

The next visualization is a heatmap displaying lane occupancy rates over one week (Figure 3.5). This diagram uses a logarithmic scale to represent occupancy rates, where the color intensity reflects higher or lower levels of traffic. Patterns emerge across different times of the day and week, with certain periods showing consistently higher occupancy, potentially corresponding to morning and evening rush hours.

To provide a more granular view, an hourly boxplot disaggregates lane occupancy rates throughout the day (Figure 3.6b). This chart reveals the distribution of traffic volumes at different times and highlights variability and outliers, which could indicate incidents or unusual traffic conditions. The plot shows higher median values during typical peak hours,

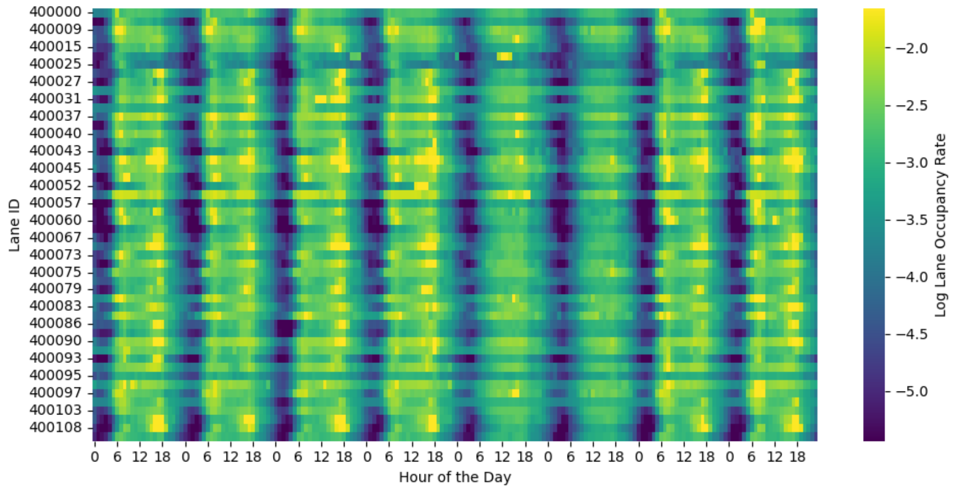


Figure 3.5. *Heatmap of Hourly Lane Occupancy Rates. Data from Monday 2008–01–07 to Sunday 2008–01–14.*

affirming the presence of daily traffic cycles.

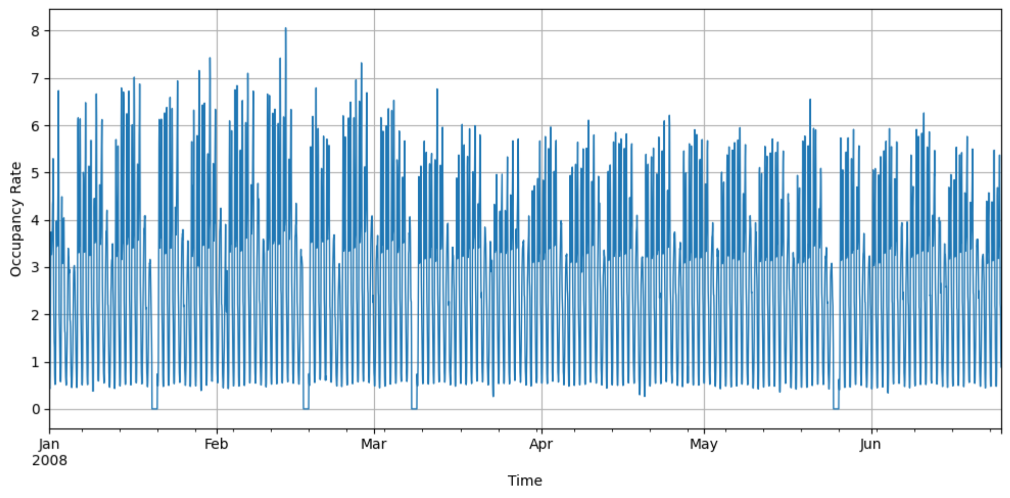
The median hourly lane occupancy rate line chart (Figure 3.6d) further clarifies these daily patterns. The curve rises sharply during peak hours and dips in off-peak times, including late night and early morning, illustrating the ebb and flow of traffic volume.

Weekly traffic patterns are examined using a boxplot by day of the week (Figure 3.6c). This visualization can indicate how traffic volumes shift between weekdays and weekends, with weekdays typically showing higher occupancy rates due to commuter traffic.

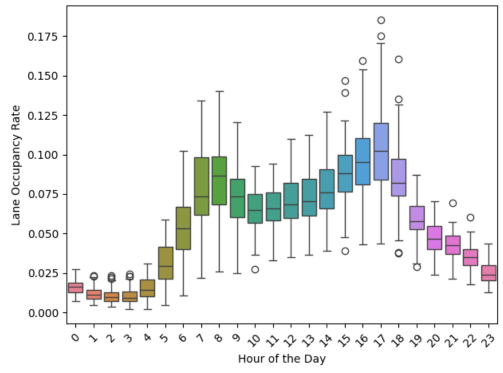
The weekly patterns are further explored with a median weekly lane occupancy rate line chart (Figure 3.6e), which shows the average occupancy rate for each day of the week. The curve typically peaks on weekdays and drops during the weekend, highlighting the reduced traffic volumes when fewer people travel to work or school.

A line chart of the median occupancy rate over several months (Figure 3.6a) provides insights into broader traffic patterns, potentially influenced by external factors such as weather conditions, holidays, or construction work. The chart may reveal days with particularly high or low traffic volumes, which can be critical for forecasting and understanding traffic behavior. It should be noted that the missing values in the dataset are visible in this chart as gaps in the time series.

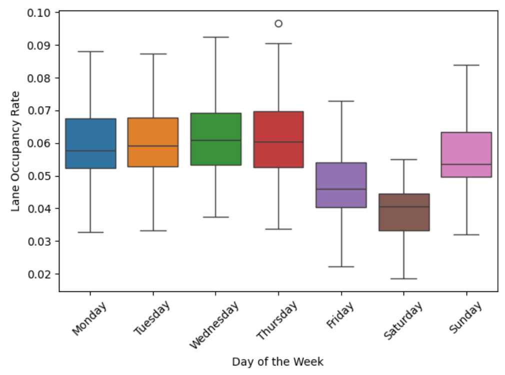
These visualizations collectively provide an in-depth look into the traffic dynamics captured by the PEMS-SF dataset. The observed correlations between lanes, the distinct hourly and weekly patterns, and the impact of longer-term factors on lane occupancy rates are all critical elements that inform the subsequent modeling efforts. Such a detailed exploratory analysis is essential for identifying the most relevant features and patterns to consider when developing and tuning traffic forecasting models.



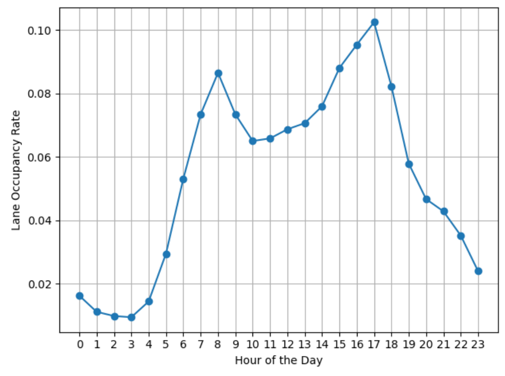
(a) *Median Lane Occupancy Rates*



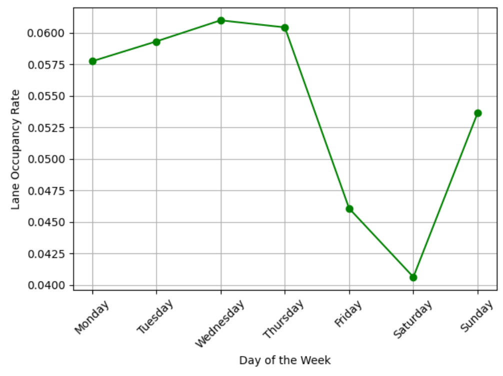
(b) *Hourly Boxplot of Lane Occupancy Rates*



(c) *Weekly Boxplot of Lane Occupancy Rates*



(d) *Median Hourly Lane Occupancy Rates*



(e) *Median Weekly Lane Occupancy Rates*

Figure 3.6. *PeMS-SF Seasonality*

3.4 Conclusion

The careful selection and preprocessing of the Electricity Load Diagrams and PeMS-SF datasets allow for a rigorous evaluation of the forecasting models. By adapting these datasets for time series forecasting, this thesis aims to shed light on the predictive power of the models and flexibility in dealing with different types of real-world data. The modifications made to the datasets, including resampling and handling missing values, are critical for ensuring the datasets' suitability for the forecasting tasks at hand, providing a fair basis for comparison across the models.

Chapter 4

Methodology

This chapter details the methodology applied to preprocess the datasets, tune and train the forecasting models, and evaluate their performance. The aim is to provide a comprehensive understanding of the practical steps taken to ensure the reliability and validity of the forecasting outcomes.

4.1 Method Overview

The thesis aims to forecast the last day of each time series within the datasets. For this purpose, the Prophet model is fitted individually for each time series, utilizing historical data up to the penultimate day. Deep learning models, employ a strategy of fitting on random selections of sequences of 7+1 days (for DeepVAR these selections are synchronized), utilizing the first seven days to forecast the final day. This approach ensures a balance between leveraging historical trends and focusing on the immediate past for prediction accuracy.

Hyperparameter tuning for the Prophet model employs a cross-validation strategy in the form of *Simulated Historical Forecasts* (SHF) [1]. This procedure, depicted in 4.2, is based on classical *rolling origin* methods [69], but uses only a small sequence of cutoff dates rather than making one forecast per historical date, to reduce the computational cost and produce measurements that are not as correlated. Specifically, we start with a cutoff point that is relatively close to the final date and then select more cutoff points by advancing with half-horizon strides.

Deep learning models' hyperparameters except for the learning rate are optimized using the Optuna library [70] and its *Tree-Structured Parzen Estimator* (TPE) [71], a Bayesian optimization method, with 30 trials per model and dataset. The learning rate where a mix of Optuna and Pytorch Lightning's `Tuner` that uses the *Cyclical Learning Rates* (CLR) method [72] was employed. Validation is performed on the last three weeks of data sans the final day, constituting approximately 10% of the total training samples. This validation set also facilitates early stopping, optimizing the training process by preventing overfitting.

For deep learning models, missing values were substituted with zeros, a common practice that can also be viewed as a form of training regularization. In contrast, the Prophet model

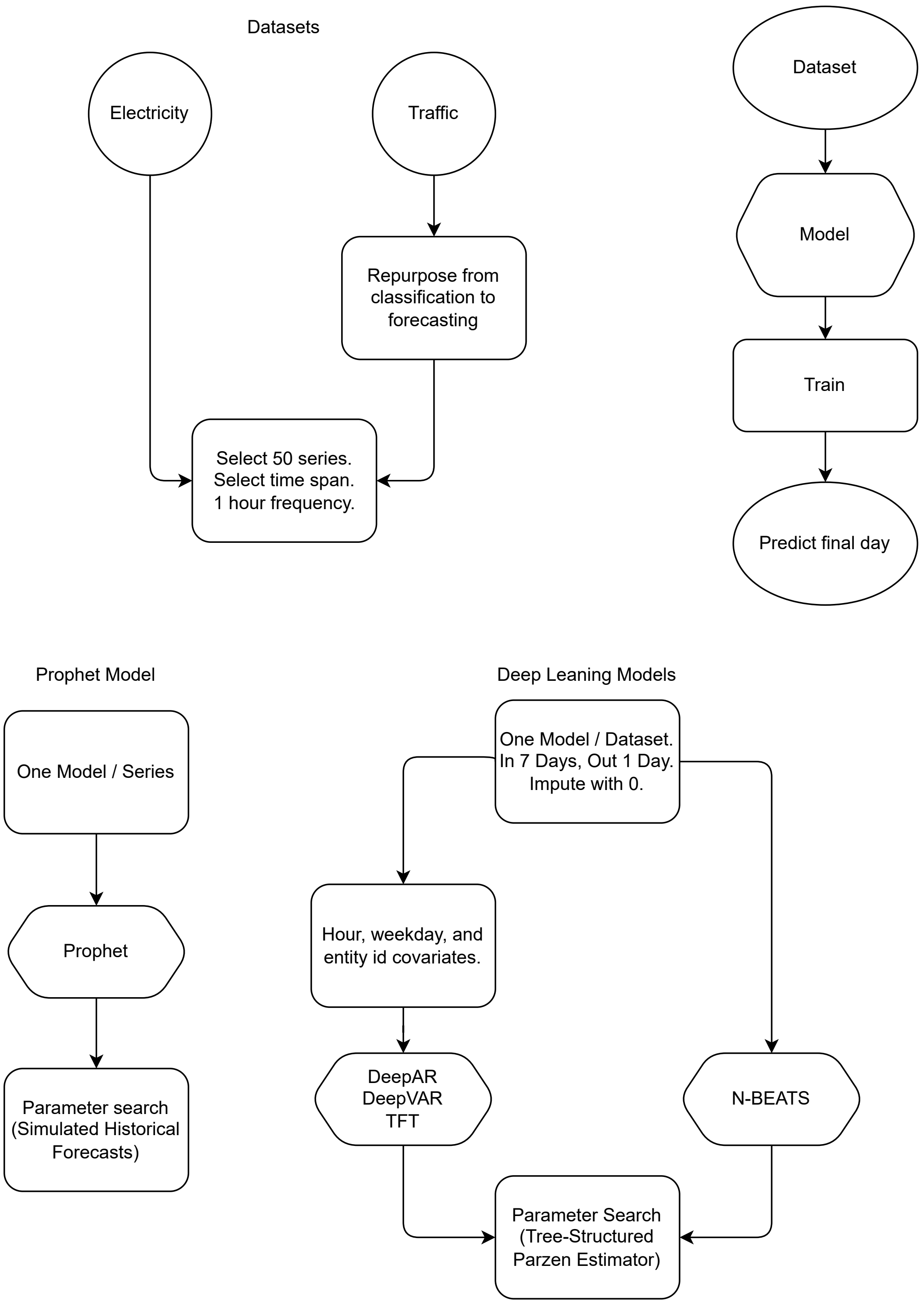

Datasets
Electricity
Traffic
Repurpose from classification to forecasting
Select 50 series. Select time span. 1 hour frequency.
Dataset
Model
Train
Predict final day
Prophet Model
One Model / Series
Prophet
Parameter search (Simulated Historical Forecasts)
Deep Leaning Models
One Model / Dataset. In 7 Days, Out 1 Day. Impute with 0.
Hour, weekday, and entity id covariates.
DeepAR DeepVAR TFT
N-BEATS
Parameter Search (Tree-Structured Parzen Estimator)

Figure 4.1. *Overview of the methodology.*

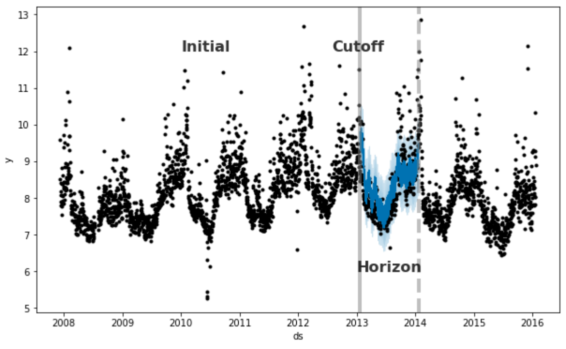


Figure 4.2. *Cross-validation for tuning Prophet hyperparameters.*

inherently manages missing values by simply ignoring them, thus no imputation was used in that case.

Data normalization for Prophet is inherently managed via its Stan backend, requiring no explicit action. Deep learning models employ standard scaling for normalization purposes.

Covariates for the DL models, except N-BEATS, are the time series name as a static categorical feature, as well as the day of the week and the hour of the day as time-varying known categorical covariates. N-BEATS is excepted because its architecture does not support covariates. The Prophet model, fitting each time series individually, does not require the time series name covariate and internally manages temporal covariates.

4.2 Evaluation Metrics

To assess the accuracy and performance of the forecasting models, several metrics are utilized, each capturing different aspects of the forecast errors. These metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), Symmetric Mean Absolute Percentage Error (SMAPE), and Mean Absolute Scaled Error (MASE).

4.2.1 Mean Squared Error (MSE)

MSE measures the average squared difference between the estimated values and the actual value. It is defined as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.1}$$

where y_i represents the actual values, $\hat{y}_i$ are the predicted values, and n is the number of observations. MSE gives a higher weight to larger errors due to the squaring part of the formula, making it particularly useful when large errors are undesirable.

4.2.2 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE, providing an error metric in the same units as the data, which makes interpretation easier. It is calculated as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.2}$$

4.2.3 Mean Absolute Error (MAE)

MAE measures the average magnitude of the errors in a set of forecasts, without considering their direction. It is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.3}$$

MAE is particularly useful for understanding how big the error will be on average.

4.2.4 Mean Absolute Percentage Error (MAPE)

MAPE expresses the error as a percentage of the actual values, providing a clear picture of the error size relative to the true values. It is calculated as:

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4.4}$$

MAPE is easy to interpret but can be undefined or infinite for values of $y_i = 0$ and can disproportionately penalize overestimates.

4.2.5 Symmetric Mean Absolute Percentage Error (SMAPE)

SMAPE adjusts MAPE to be symmetric, ensuring that overestimates and underestimates are penalized equally. It is calculated as:

$$\text{SMAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\frac{2|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|} \tag{4.5}$$

SMAPE is bounded between 0 and 200%.

4.2.6 Mean Absolute Scaled Error (MASE)

MASE measures the accuracy of forecasts relative to a naïve baseline prediction, scaling the MAE by the average absolute difference between consecutive observations in the training dataset. It is defined as:

$$\text{MASE} = \frac{\text{MAE}}{\frac{1}{n-1}\sum_{i=2}^{n}|y_i - y_{i-1}|} \tag{4.6}$$

MASE is particularly useful because it is scale-independent and can be used to compare forecast performance across different data sets.

4.3 Hyperparameter Spaces

The DL models and Prophet are handled differently. For training the DL models, dropout values from 0.1, 0.2, 0.3, 0.4, and 0.5 were tried as well as gradient norm clipping to values in the logarithmic space of $[10^{-1}, 10^{2}]$. The batch size was set to 128 for both datasets and all DL models. Additionally, all the DL models were trained using early stopping with patience 5 and tolerance 10^{-4} on the validation loss. In the case of the learning rates, a mixed method was used: First, we let Pytorch Lightning's `Tuner` class [72] try and find a learning rate in $[10^{-8}, 1]$, and if the attempt was successful and the learning rate was less than 10^{-2}, then we accept it, otherwise, we let Optuna [71] suggest a value from the logarithmic space of $[10^{-4}, 10^{-2}]$. In the case of the DeepVAR model, the `Tuner` was not used, due to implementation limitations (Cholesky Decomposition on a non-positive-definite matrix), so we let Optuna recommend a value in the logarithmic space of $[10^{-5}, 10^{-2}]$.

4.3.1 Prophet

The Prophet model was tuned for the τ parameter of the Laplacian prior distribution of the change points. For both datasets, the values that were tested were 0.001, 0.01, 0.1 and 0.5. Daily and weekly seasonality components were used with Fourier orders of 4 and 3 respectively, which are recommendations from the original paper [1] and defaults of the Prophet package. These seasonality components were set to be multiplicative for the electricity dataset, and additive for the traffic dataset, to accommodate the nature of the data. Linear growth was used for the electricity dataset and logistic growth with saturation points at 0 and 1 for the traffic dataset. The values of the smoothing priors imposed on the seasonalities were set to 10.0 (Prophet's default), resulting in very little smoothing, allowing the models to learn more complex seasonal patterns, and delegating regularization to the Fourier order. The first 80% of the training data were used to find change points, and this relatively conservative choice (also Prophet's default) is due to the relatively stable trend patterns that both datasets exhibit. A holiday component was not added because the country of the electricity dataset is unknown and the traffic dataset has removed data from holidays. For validation, the first 205 days were used as a starting cutoff point for electricity and 169 days for traffic, in both cases iterating into new cutoff points with half-horizon (half-day) steps.

4.3.2 N-BEATS

The interpretable architecture was used for the N-BEATS model, due to the clear seasonal patterns found in both datasets, with the trend polynomial degree set to 3. For both the trend and the seasonality stacks, three blocks were used, with three FC layers each. The FC weights are shared with the other blocks across a stack. Sizes 64 and 256 were tried for the FC layers of the trend stack, while for the seasonality stack the candidate sizes were 512 and 2048.

4.3.3 DeepAR

DeepAR was set to have 2 LSTM layers with their sizes tested to be 80, 160, and 320. The traffic dataset was modeled using the Beta distribution, since its values lie in $[0, 1]$, while the electricity dataset was modeled using the Normal distribution. The number of Monte Carlo samples used for prediction was 100. The embedding size of the covariates was set using the heuristic

$$\min\{\text{round}(1.6n^{0.56}), 100\} \quad \text{if} \quad n > 2 \quad \text{else} \quad 1 \tag{4.7}$$

where n is the number of covariates. This heuristic is used by default by Pytorch Forecasting.

4.3.4 DeepVAR

The rank of the non-diagonal part of the covariance matrix was set to 10, which is what the original paper uses for datasets with a greater number of time series, implying that 10 is sufficient for our case. The LSTM configuration, the embedding size, and the Monte Carlo samples were the same as those in DeepAR, although, in the end, a configuration found through manual experimentation was chosen that performed better than Optuna's suggestions.

4.3.5 Temporal Fusion Transformer

TFT is set to output quantiles 0.1, 0.5 and 0.9. Model sizes 80, 160 and 320 were tested. A single LSTM layer was used and the number of interpretable multihead attention heads was set to 4. The embedding size here is equal to the model size, as is imposed by the architecture.

4.4 Source Code

The source code for the experiments can be found as a Python package named `thesis`, hosted on `https://github.com/k-papadakis/dsml-thesis-code`. This package encapsulates the functionality required for hyperparameter searching and model training. Users can install this package with `pip install --upgrade git+https://github.com/k-papadakis/dsml-thesis-code.git` and typing `thesis -h` in the terminal for detailed usage instructions (to reproduce the results use the argument `--seed 42`). The package leverages the libraries *Pytorch Forecasting* for deep learning models and *Optuna* for hyperparameter optimization, with Facebook's *Prophet* library utilized for the Prophet model. Both datasets are meticulously prepared to align with the APIs of the Pytorch Forecasting and Prophet libraries.

DL experiments were conducted on a single NVIDIA Tesla P100 GPU. The time required to optimize DL hyperparameters, using 30 Optuna trials per model is approximately 16 hours. The time required to train all DL models for both datasets is approximately 5 hours. The time required to find hyperparameters of the Prophet models for all time series of both datasets and train those models is approximately 3 hours.

Chapter 5

Model Applications and Results

This chapter details the results obtained from the application of the models described in Chapter 2 on the datasets described in Chapter 3 using the methodology described in Chapter 4.

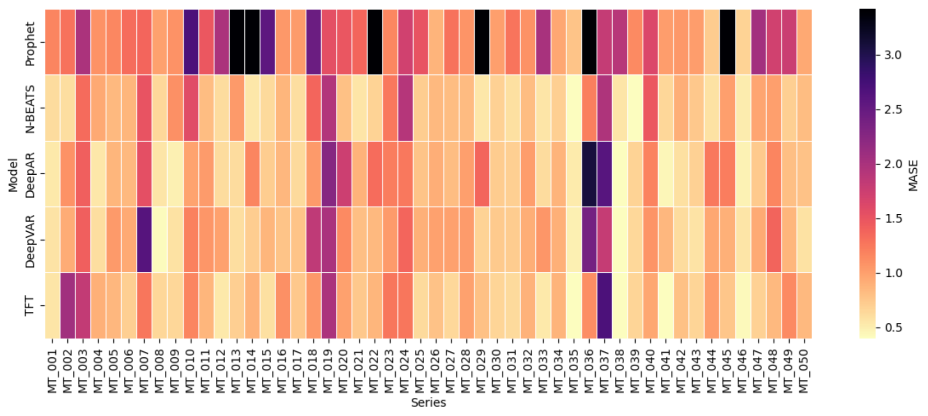
	MASE	SMAPE	RMSE	MAE
Prophet	1.81	0.231	16.8	14.2
N-BEATS	0.84	0.104	18.3	7.6
DeepAR	0.64	0.078	14.0	5.6
DeepVAR	0.66	0.080	15.1	5.9
TFT	0.86	0.103	20.6	7.9

(a) *Performance on Electricity*

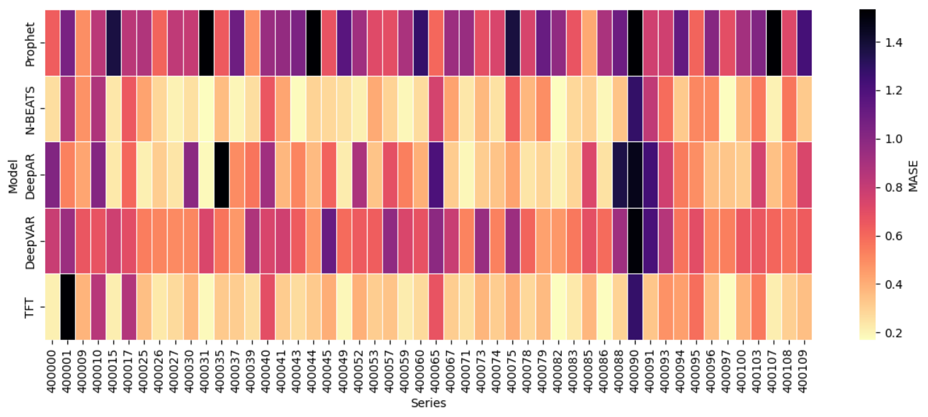
	MASE	SMAPE	RMSE	MAE	MAPE
Prophet	0.943	0.290	0.01547	0.01149	0.345
N-BEATS	0.436	0.101	0.01195	0.00529	0.133
DeepAR	0.438	0.087	0.01083	0.00494	0.106
DeepVAR	0.542	0.161	0.01073	0.00618	0.229
TFT	0.408	0.089	0.01281	0.00495	0.101

(b) *Performance on Traffic*

Table 5.1. *Performance of the models on the datasets. MAPE was omitted for the Electricity dataset due to the presence of zeros in the true values.*



(a) *Electricity MASE*



(b) *Traffic MASE*

Figure 5.1. *MASE of the models on each series of the datasets.*

The performance of each model on each dataset on the metrics described in Section 4.2 is presented in Tables 5.1a and 5.1b. An evaluation of the performance of each architecture per series is shown in Figures 5.1a and 5.1b.

From each dataset, the series with the lowest, median, and highest average values are shown, so that they represent samples from all scales and allow us to see how each model handles them. See also the discussion in 2.3.1 for the discussion on varying scales and the power law.

In the following sections, each model's performance is discussed in detail, with the main focus being on the MASE.

5.1 Prophet

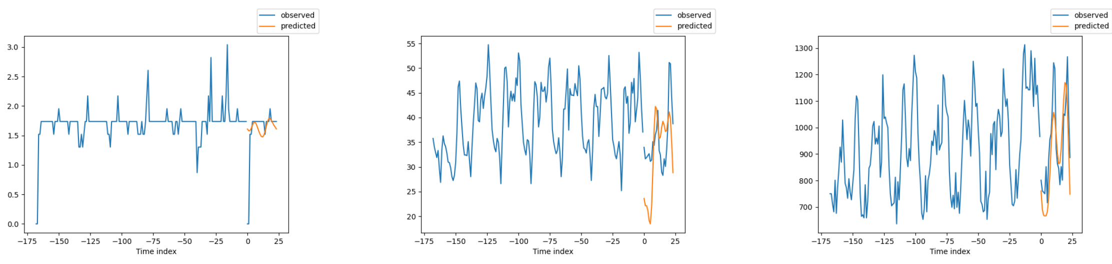
Based on Tables 5.1a and 5.1b, Prophet consistently underperforms compared to the other models across all metrics. This suggests inherent limitations in its ability to capture the complex patterns present in the data, possibly due to its relatively simple, albeit interpretable, additive structure. It should be noted though that the uncertainty intervals do generally cover the true values appropriately, which is useful for further decision making.

While manual adjustments might improve performance in isolated cases, this approach lacks scalability and highlights Prophet's lower learning capacity compared to the more flexible deep learning (DL) models. Prophet's focus on decomposing time series into simpler components likely restricts its ability to model the intricate nonlinearities and dependencies exhibited by the datasets.

Figures 5.2 and 5.6 display Prophet's predictions on the Electricity and Traffic datasets. Figures 5.3 and 5.7 present the errors generated by Simulated Historical Forecasts, offering insights into the model's accuracy. For a deeper understanding of Prophet's modeling process, Figures 5.4 and 5.8 illustrate the trend and seasonality components extracted by the model. Finally, Figures 5.5 and 5.9 highlight the significant change points in the trend identified by the model.

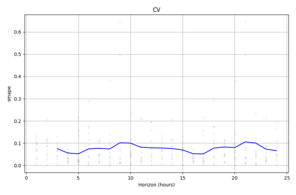
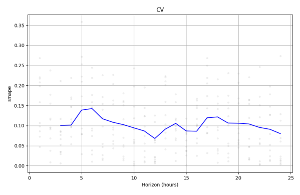
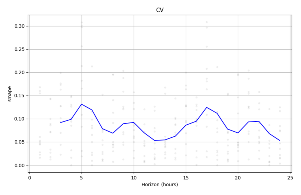
In the electricity dataset, the models demonstrate limitations in accurately capturing the extreme values of MT003 and the volatility of MT015, although they exhibit relatively satisfactory forecasting performance for MT045. While the trend component is adequately captured, the models struggle to precisely capture the underlying seasonality. The daily seasonality of MT015 and MT043 exhibits a significant decrease during sleeping hours, as expected. Furthermore, the consumption of MT015 is lower during weekends, potentially indicating a business-related usage pattern, while MT043's consumption is higher, possibly indicating a household-related usage pattern. However, the pattern for MT003 is not as discernible.

In the traffic dataset, which exhibits a higher degree of regularity, the models exhibit enhanced performance, albeit with certain challenges in accurately capturing the higher values. Furthermore, the models encounter difficulties in effectively capturing the abrupt increase in variance observed in series 400041 from March onwards. The presence of double peaks in the daily seasonality, learned by all three models, corresponds to the morning and evening rush hours. Additionally, the notable decrease during weekends in the weekly component aligns with the expected behavior of the traffic dataset. Finally, from the error plots it is readily seen that the model has trouble accurately predicting 12 to 16 hours ahead.



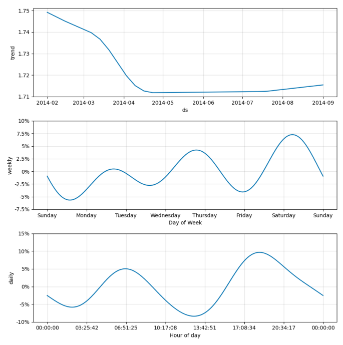
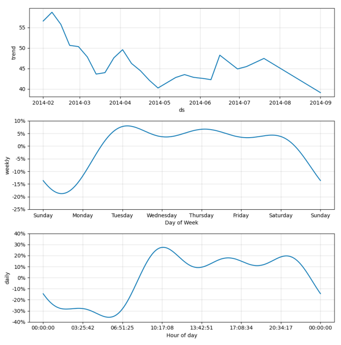
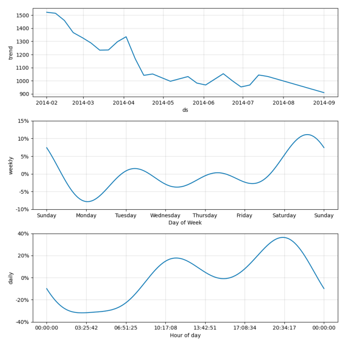
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.2. *Predictions of Prophet on the Electricity dataset.*

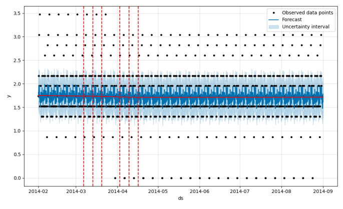
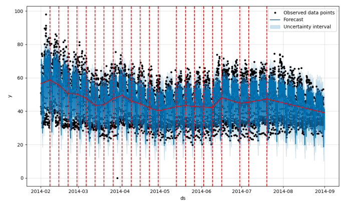
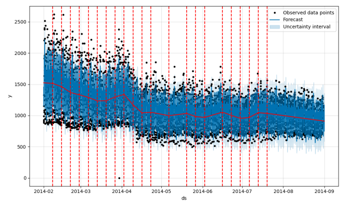
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.3. *Cross-validation SMAPE of Prophet on the Electricity dataset. The grey dots represent SMAPEs on that specific time index, one from each simulated historical forecast, and the blue line represents the average SMAPE on that time index.*

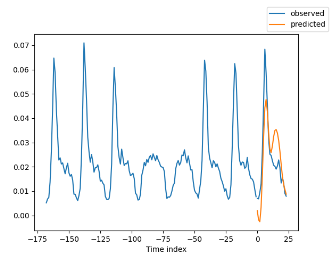
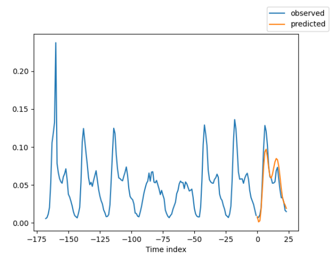
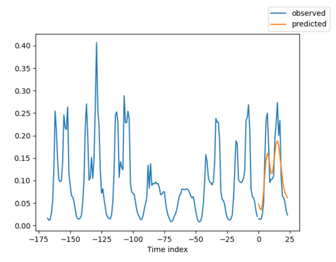
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.4. *Trend and seasonality components of Prophet on the Electricity dataset.*

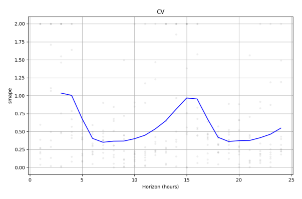
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.5. *Model overview of Prophet on the Electricity dataset. Forecasts in blue lines, uncertainty in light blue, true values in black dots, and important change points in dashed vertical red lines.*

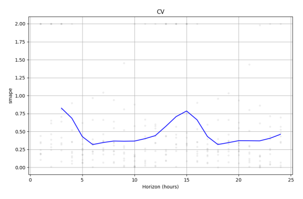




(a) *400015* **(b)** *400071* **(c)** *400041*

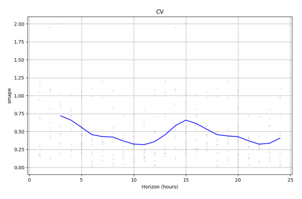
Figure 5.6. *Predictions of Prophet on the Traffic dataset.*



(a) *400015*

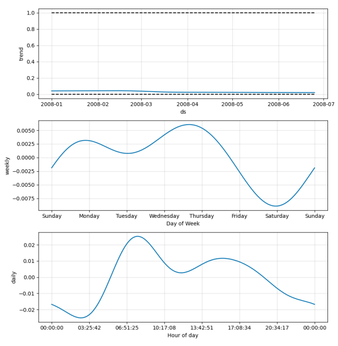


(b) *400071*

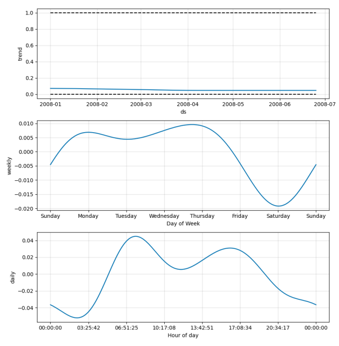


(c) *400041*

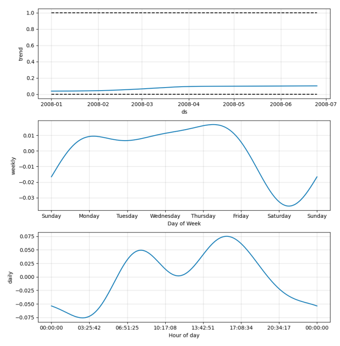
Figure 5.7. *Cross-validation SMAPE of Prophet on the Traffic dataset. The grey dots represent SMAPEs on that specific time index, one from each simulated historical forecast, and the blue line represents the average SMAPE on that time index.*



(a) *400015*

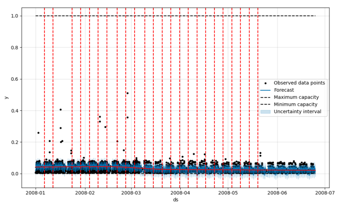


(b) *400071*

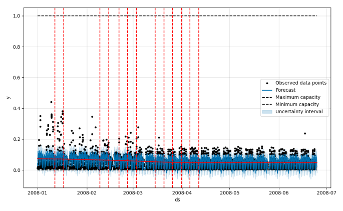


(c) *400041*

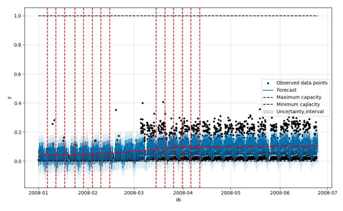
Figure 5.8. *Trend and seasonality components of Prophet on the Traffic dataset.*



(a) *400015*



(b) *400071*



(c) *400041*

Figure 5.9. *Model overview of Prophet on the Traffic dataset.*

5.2 N-BEATS

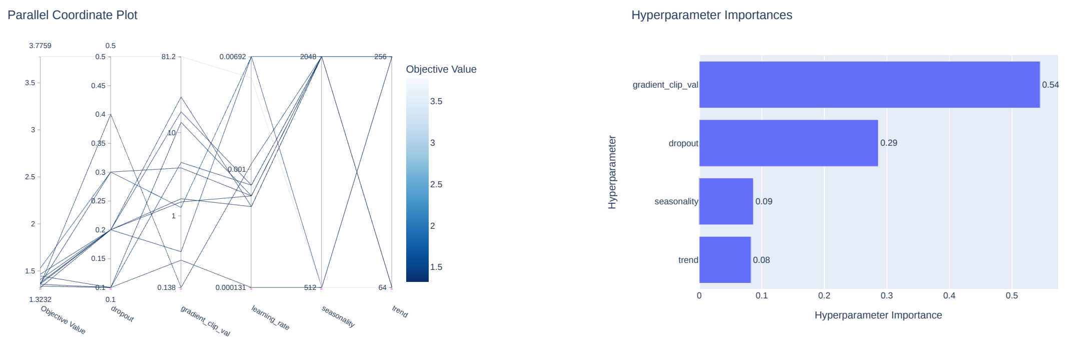
The N-BEATS model yielded robust predictions for the traffic dataset; however, its performance was less effective for the electricity dataset. Hyperparameter optimization revealed a preference for increased sizes in the trend and seasonality FC layers, with an intermediate degree of dropout and restrictions on gradient norms.

Analysis of the electricity dataset outcomes revealed the model's proficiency in capturing the time series pattern but not the scale of the values. The model's learned seasonality aligned with the prediction range but diverged from the conditioning range, a result anticipated due to the exclusion of conditioning range errors from the loss calculation. The trend alignment was satisfactory.

Contrastingly, for the traffic dataset, N-BEATS showcased an adeptness in mirroring both the pattern and scale of the series, benefiting from the dataset's consistent scale. As with the electricity dataset, inaccuracies in the seasonality component within the conditioning range were noted, attributable to the model's loss function configuration.

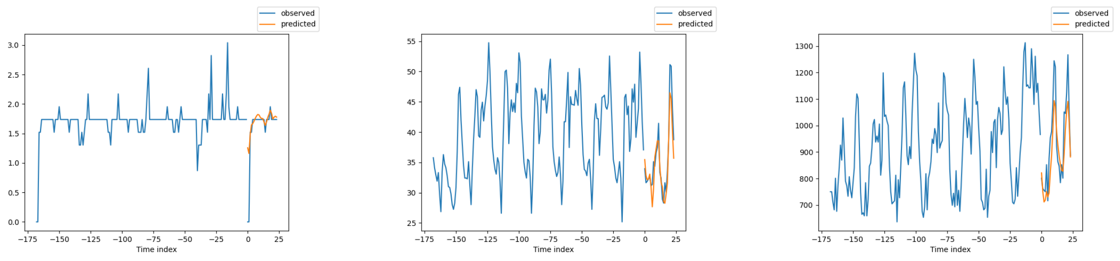
	Grad Clip	Dropout	Learning Rate	Trend size	Seasonality size
electricity	1.60	0.2	0.0005	256	2048
traffic	1.25	0.3	0.0008	256	2048

Table 5.2. *Optuna-found hyperparameters for N-BEATS.*



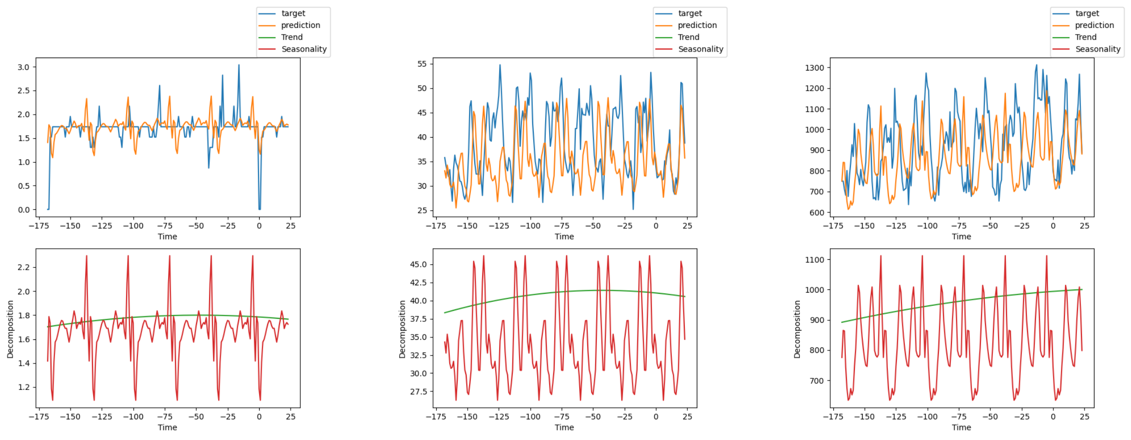
(a) *Parallel Coordinates* **(b)** *Hyperparameter Importances*

Figure 5.10. *Optuna results for the N-BEATS model on the Electricity dataset.*



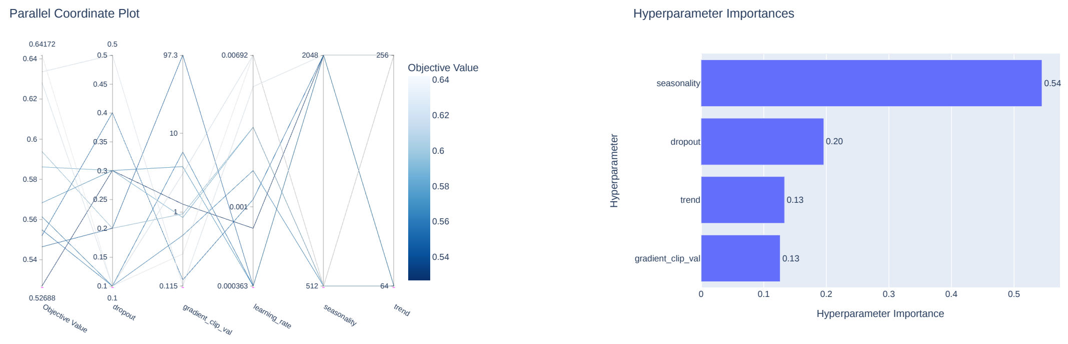
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.11. *Predictions of N-BEATS on the Electricity dataset.*



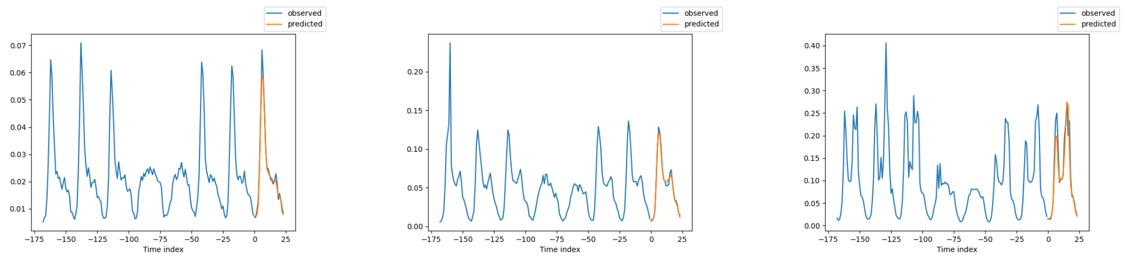
(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.12. *Trend-Seasonality interpretation of N-BEATS on the Electricity dataset.*



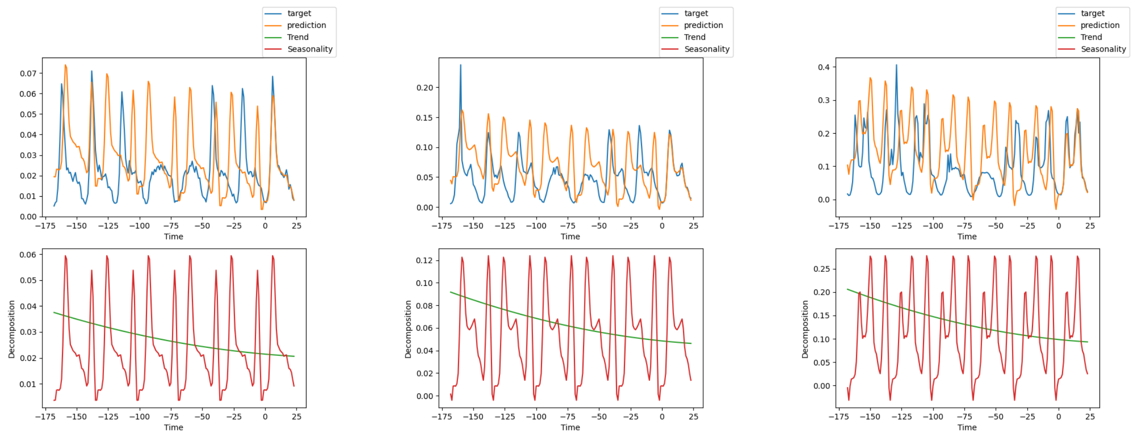
(a) *Parallel Coordinates* **(b)** *Hyperparameter Importances*

Figure 5.13. *Optuna results for the N-BEATS model on the Traffic dataset.*



(a) *400015* **(b)** *400071* **(c)** *400041*

Figure 5.14. *Predictions of N-BEATS on the Traffic dataset.*



(a) *400015* **(b)** *400071* **(c)** *400041*

Figure 5.15. *Trend-Seasonality interpretation of N-BEATS on the Traffic dataset.*

5.3 DeepAR

The performance of DeepAR on the electricity dataset was notably superior in comparison to alternative methodologies, demonstrating commendable results also on the traffic dataset. It was observed that despite certain predictions not achieving exactitude, the provision of confidence intervals effectively compensates by encompassing the target values.

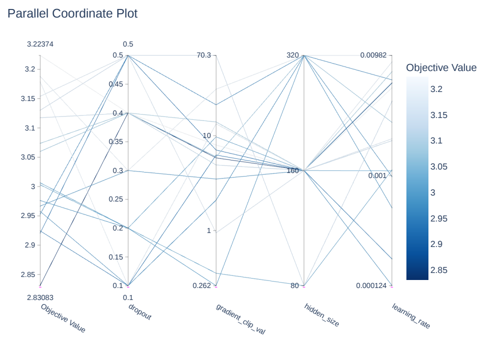
The exploration of hyperparameters yielded an optimization towards an intermediate size for the Long Short-Term Memory (LSTM) architecture across both datasets, alongside comparable learning rates and an allowance for increased gradient norms within the traffic dataset.

Regarding the electricity dataset, the capacity of DeepAR to apprehend and replicate the dynamics across various time-series magnitudes was affirmed, aligning with the assertions made within the foundational paper. Specifically, within the context of the MT003 series, DeepAR excelled in accurately discerning the series' structural nuances and generating robust forecasts. The model's prediction of a drop to zero was precisely anticipated within the confidence interval. Given that DeepAR's target distribution for this scenario is modeled on a Normal distribution, which is symmetric, the derived confidence intervals are inherently symmetrical, despite the practical applicability of a skewed interval. Consequently, this introduces a measure of uncertainty devoid of skewness indication, notwithstanding the practical relevance of anticipating a downward skew.

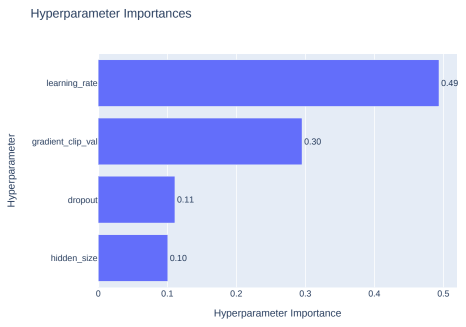
In the analysis of the traffic dataset, the accuracy of DeepAR's predictions was similarly noteworthy. The departure from symmetrical distributions is attributed to the usage of the Beta distribution. Instances such as 400071 and 40041 exhibited relatively broad confidence intervals at the peaks, congruent with the magnitudes of historical peaks, thereby reflecting the model's adeptness at capturing the dataset's variability.

	LSTM Size	Grad Clip	Dropout	Learning rate
electricity	160	5.83	0.4	0.006
traffic	160	44.44	0.4	0.004

Table 5.3. *Optuna-found hyperparameters for DeepAR.*

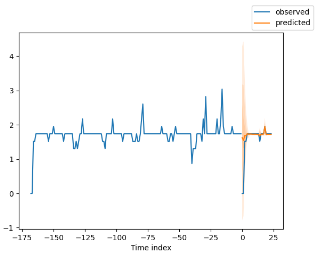


(a) *Parallel Coordinates*

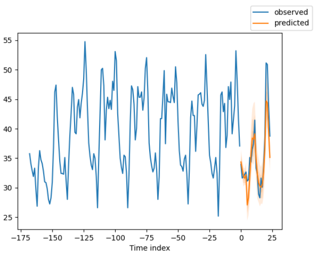


(b) *Hyperparameter Importances*

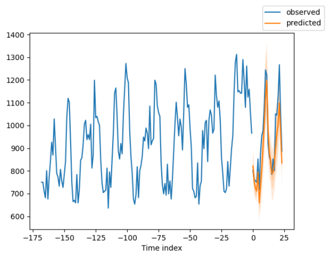
Figure 5.16. *Optuna results for the DeepAR model on the Electricity dataset.*



(a) *MT003*

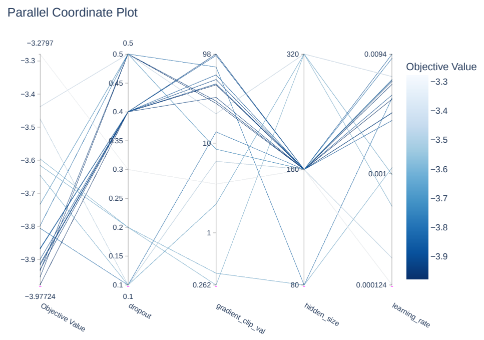


(b) *MT015*

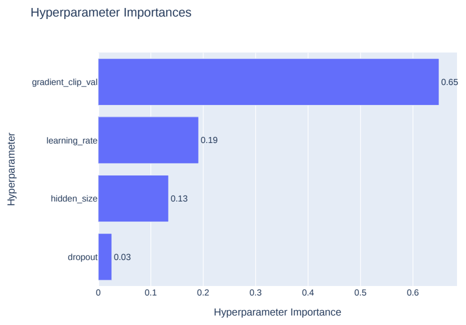


(c) *MT043*

Figure 5.17. *Probabilistic predictions of DeepAR on the Electricity dataset. Monte Carlo samples: 100, Quantiles: 0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98.*

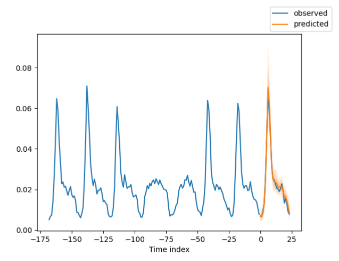


(a) *Parallel Coordinates*

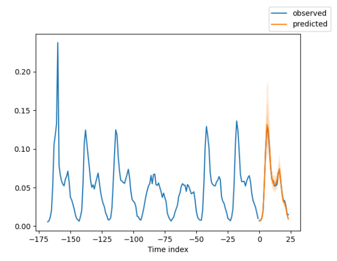


(b) *Hyperparameter Importances*

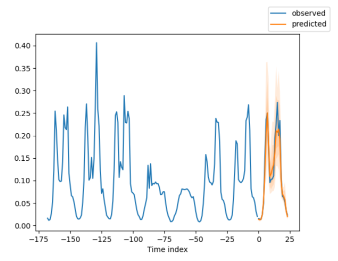
Figure 5.18. *Optuna results for the DeepAR model on the Traffic dataset.*



(a) *400015*



(b) *400071*



(c) *400041*

Figure 5.19. *Probabilistic predictions of DeepAR on the Traffic dataset. Monte Carlo samples: 100, Quantiles: 0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98.*

5.4 DeepVAR

DeepVAR exhibited commendable performance on the electricity dataset, akin to that of DeepAR; however, its efficacy on the traffic dataset was found lacking.

While hyperparameter optimization was conducted for DeepVAR (Table 5.4), subsequent experimentation revealed that adopting the optimized hyperparameters from DeepAR yielded superior performance. Therefore, the final results presented in this thesis utilize DeepAR's hyperparameter configuration for DeepVAR as well.

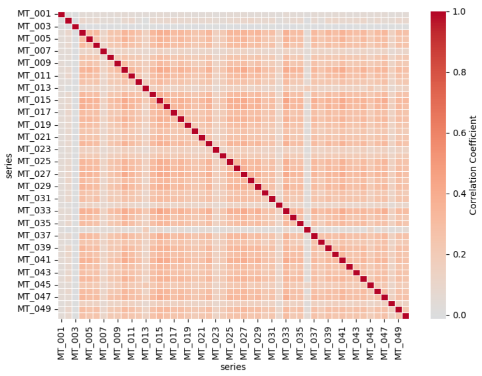
Within the context of the electricity dataset, DeepVAR's forecasted outcomes bore a striking resemblance to those of DeepAR, making it, alongside DeepAR, uniquely capable of rendering precise predictions for the MT003 series.

It is to be noted that the confidence intervals are symmetric, which is due to the series vector being modeled to have a multivariate normal distribution, by utilizing a Gaussian copula, which renders the marginal distributions normal, like in the DeepAR case. Comparative analysis of the averaged learned covariance matrices, as depicted in Figure 5.22, with those in Figure 3.1, reveals a parallel in the overall structure. However, discrepancies in the distribution and mean values of these covariances are evident. A potential contributor to this variation is the model's diagonal-plus-low-rank approximation for the covariance, utilizing a rank of 10 as opposed to the full rank of 50.

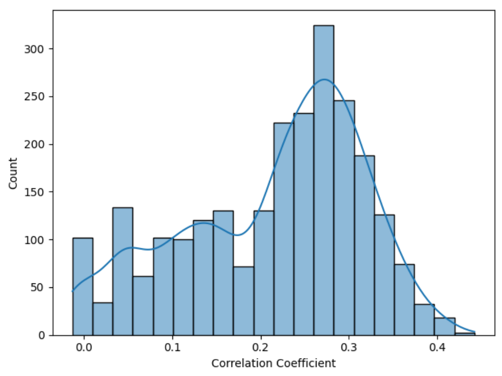
Concerning the traffic dataset, the model's estimations of distribution means were approximately accurate, yet the quantiles displayed significant variance, a scenario not justified by the dataset's noise levels. This issue is most likely attributed to an inadequate estimation of the covariance matrix, as illustrated in Figure 5.24, particularly when juxtaposed against Figure 3.4.

	LSTM Size	Grad Clip	Dropout	Learning rate
electricity	160	1.23	0.3	0.001
traffic	80	96.73	0.5	0.006

Table 5.4. *Optuna-found hyperparameters for DeepVAR. They were replaced with DeepAR's hyperparameters for the final results.*

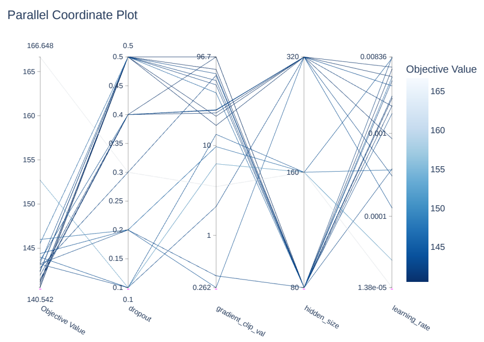


(a) *Correlation Heatmap*

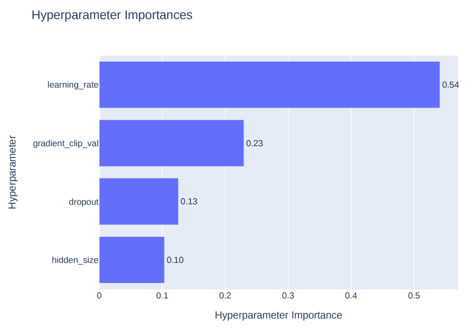


(b) *Correlation Histogram (1.0 excluded)*

Figure 5.22. *Average Correlation of the DeepVAR model on the Electricity dataset. Result from the average of the covariance matrices across Monte Carlo samples and forecast indices. See also Figure 3.1*

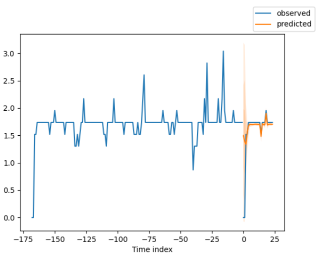


(a) *Parallel Coordinates*

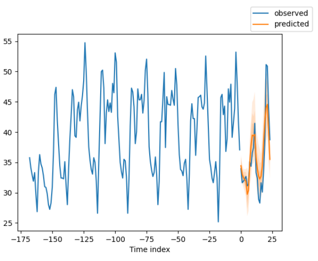


(b) *Hyperparameter Importances*

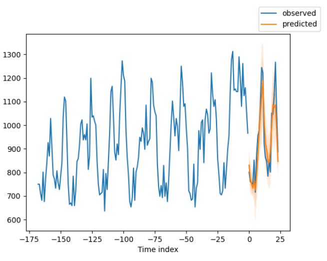
Figure 5.20. *Optuna results for the DeepVAR model on the Electricity dataset. They were replaced with DeepAR's hyperparameters for the final results.*



(a) *MT003*

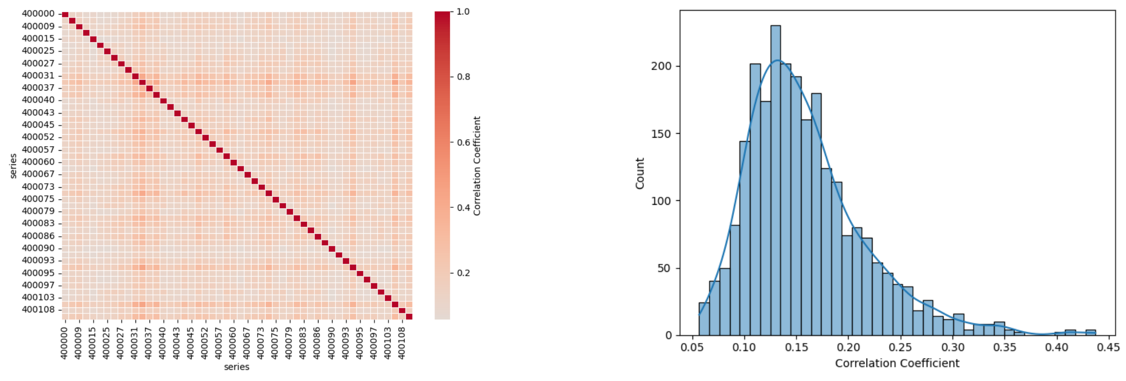


(b) *MT015*



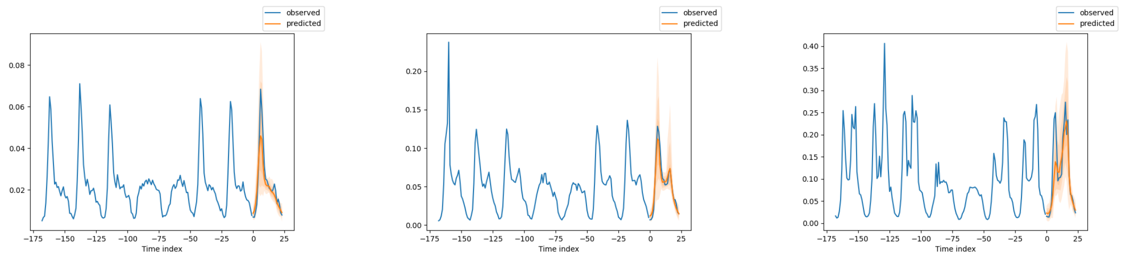
(c) *MT043*

Figure 5.21. *Probabilistic predictions of DeepVAR on the Electricity dataset. Monte Carlo samples: 100, Quantiles: 0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98.*



(a) *Correlation Heatmap* **(b)** *Correlation Histogram (1.0 excluded)*

Figure 5.24. *Average Correlation of the DeepVAR model on the Traffic dataset. Result from the average of the covariance matrices across Monte Carlo samples and forecast indices. See also Figure 3.4*



(a) *400015* **(b)** *400071* **(c)** *400041*

Figure 5.23. *Probabilistic predictions of DeepVAR on the Traffic dataset. Monte Carlo samples: 100, Quantiles: 0.02, 0.1, 0.25, 0.5, 0.75, 0.9, 0.98.*

5.5 Temporal Fusion Transformer

The Temporal Fusion Transformer (TFT) demonstrated superior performance on the traffic dataset while exhibiting suboptimal results on the electricity dataset.

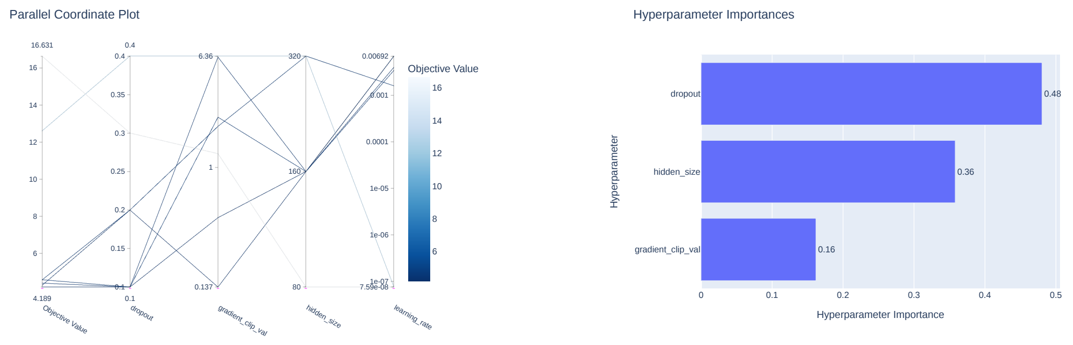
The hyperparameter optimization process advocated for a configuration of medium complexity for the electricity dataset and a more complex configuration for the traffic dataset. The optimization imposed constraints on the norms, maintaining them at lower values for the electricity dataset and permitting significantly higher values for the traffic dataset. Recommendations for dropout rates and learning rates remained consistent across both datasets.

In the context of the electricity dataset, while the TFT was capable of accurately capturing the temporal patterns of the time series, it encountered difficulties in accurately learning the magnitudes of the series, in contrast to the DeepAR model. The model's attention in Figure 5.27 peaks at two days in the past instead of one, which could be related to the model's underperformance on the dataset. The encoder's variable selection network assigned greater importance to the hour covariate over historical time series data, as evidenced in Figure 5.28a, suggesting a relative underperformance in learning from the time series data —a conclusion further supported by the superior efficacy of alternative models. The emphasis on the hour of the day by both the encoder and decoder's variable selection networks, as illustrated in Figure 5.28, aligns with expectations considering the significant influence of time of day on electricity consumption patterns.

Conversely, on the traffic dataset, characterized by less variation in time series magnitudes, TFT achieved remarkable predictive accuracy. The model's attention exhibited peaks at intervals corresponding to multiples of 24 hours in the past, with a pronounced emphasis on data from the preceding day, followed by data from five and six days prior, as shown in Figure 5.31. This attention distribution could be indicative of the model leveraging information from the previous day for local trends, and information from five and six days prior for capturing weekly patterns since data from seven days ago are not in the lookback window. In this dataset, the decoder's variable selection network prioritized historical time series data significantly, as shown in Figure 5.32a, marking a distinct contrast to the electricity dataset scenario. The importance assigned to the day of the week and hour of the day by the variable selection network was approximately equal, as indicated in Figure 5.28, suggesting a balanced consideration of these factors in the model's predictive process.

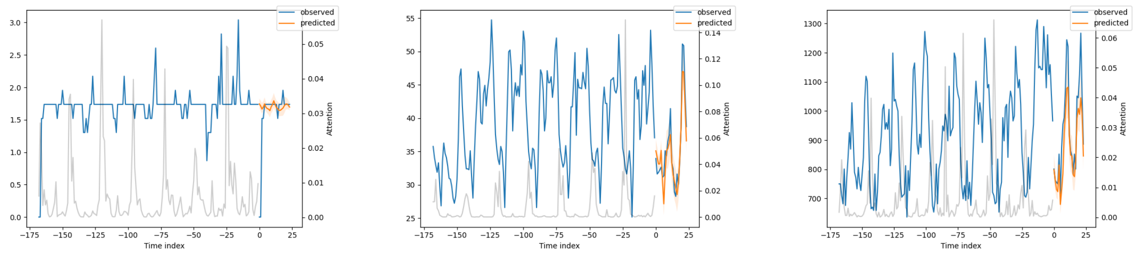
	Model Size	Grad Clip	Dropout	Learning Rate
electricity	160	0.43	0.1	0.007
traffic	320	20.69	0.2	0.003

Table 5.5. *Optuna-found hyperparameters for TFT.*



(a) *Parallel Coordinates* **(b)** *Hyperparameter Importances*

Figure 5.25. *Optuna results for the TFT model on the Electricity dataset.*



(a) *MT003* **(b)** *MT015* **(c)** *MT043*

Figure 5.26. *Quantile predictions of TFT on the Electricity dataset. The attention weights for each time index are also shown. Predictions at quantiles 0.1, 0.5 and 0.9*

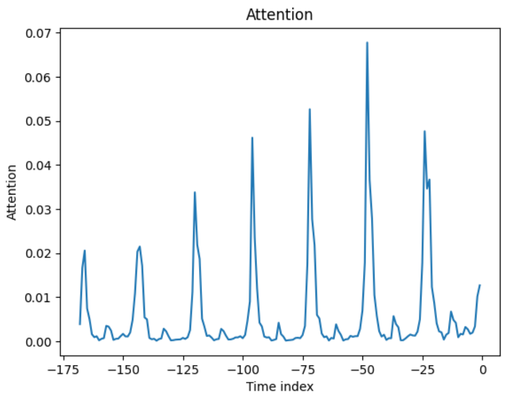
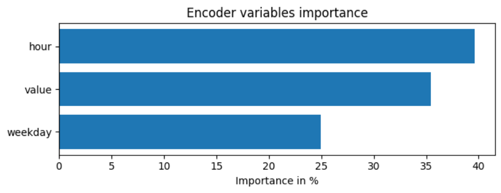
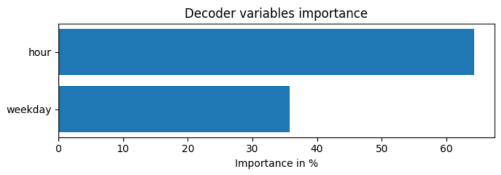


Figure 5.27. *Average attention of TFT on the conditioning range on the Electricity dataset.*

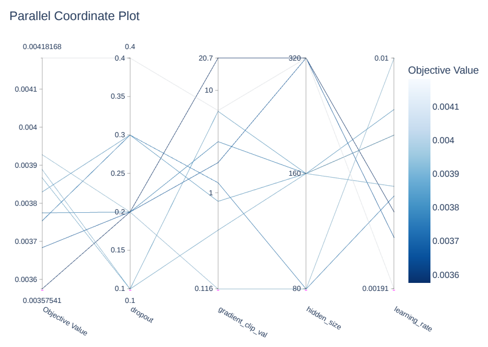


(a) *Encoder Variables Importance*

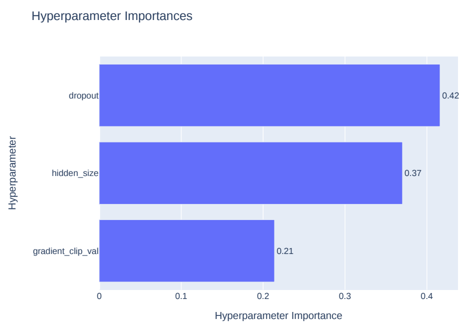


(b) *Decoder Variables Importance*

Figure 5.28. *Variables importance of the encoder and decoder of the TFT model on the Electricity dataset. Computed using the softmax outputs of the Variable Selection Networks across time.*

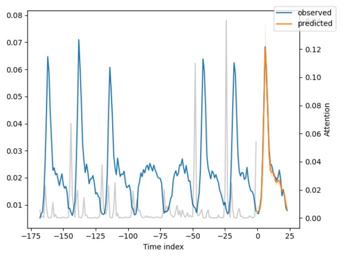


(a) *Parallel Coordinates*

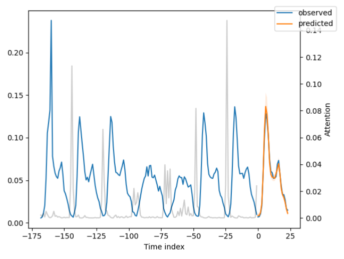


(b) *Hyperparameter Importances*

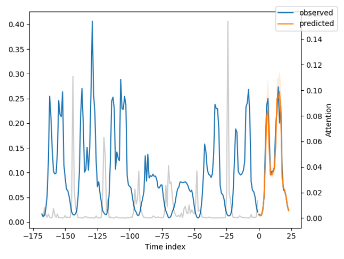
Figure 5.29. *Optuna results for the TFT model on the Traffic dataset.*



(a) *400015*



(b) *400071*



(c) *400041*

Figure 5.30. *Quantile predictions of TFT on the Traffic dataset. The attention weights for each time index are also shown. Predictions at quantiles 0.1, 0.5 and 0.9*

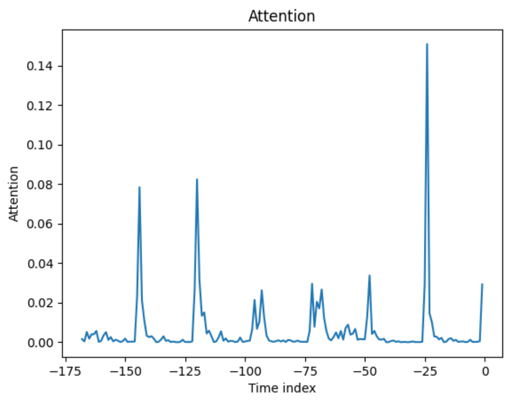
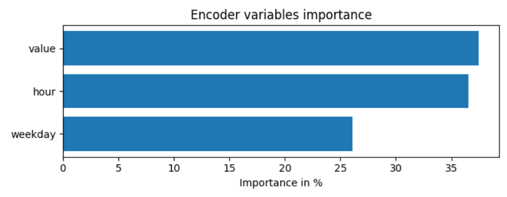
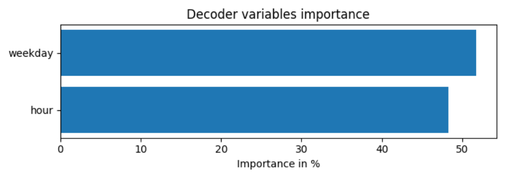


Figure 5.31. *Average attention of TFT on the conditioning range on the Traffic dataset.*



(a) *Encoder Variables Importance*



(b) *Decoder Variables Importance*

Figure 5.32. *Variables importance of the encoder and decoder of the TFT model on the Traffic dataset. Computed using the softmax outputs of the Variable Selection Networks across time.*

Chapter 6

Conclusion and Future Work

6.1 Summary of Findings

This thesis embarked on a comprehensive investigation of machine learning models for time series forecasting, with a focus on their applicability to real-world electricity consumption and traffic flow datasets. The selected models, spanning from the established Prophet framework to cutting-edge deep learning architectures, were meticulously evaluated using a rigorous experimental methodology.

Key takeaways from this research include:

- *Deep Learning Supremacy*: Deep learning models (N-BEATS, DeepAR, DeepVAR, TFT) consistently outperformed Prophet on both datasets, particularly in accurately capturing complex patterns and nonlinearities. This underscores the power of DL for sophisticated time series analysis when the volume of the data allows it.
- *Dataset Nuances*: Model performance varied significantly based on the dataset. N-BEATS excelled on the more regular traffic patterns but struggled with the electricity dataset's variability. Similarly, TFT's accuracy was influenced by the dataset's scale and complexity.
- *Probabilistic Advantage*: DeepAR and DeepVAR's ability to provide probabilistic forecasts, TFT's confidence intervals, as well as Prophet's uncertainty intervals, is a significant advantage, offering valuable insight into potential uncertainty, that can heavily impact decision-making. That said, such intervals should be used with caution since they can often be misleading.
- *The Importance of Manual Tuning*: In certain cases, manual hyperparameter tuning outperforms fully automated approaches for deep learning models. This underscores the value of domain knowledge and experimentation within the optimization process.
- *Interpretability Considerations*: While N-BEATS offers a theoretically interpretable architecture, the results suggest that practical interpretability benefits might be limited. TFT, on the other hand, provided valuable insights through its attention mechanism and variable selection. Finally, the interpretability of the Prophet model comes

with model simplicity.

6.2 Implications and Recommendations

The findings of this thesis have practical implications for real-world forecasting applications:

- *Electricity Demand Management*: DL models like DeepAR, due to their precision and uncertainty quantification, offer significant potential for optimizing energy distribution and planning.
- *Traffic Congestion Mitigation*: TFT's strong performance on the traffic dataset suggests its suitability for real-time traffic prediction, aiding in traffic management and infrastructure planning decisions.
- *Model Selection Guidance*: This research provides a data-driven basis for model selection. Practitioners can consider the nature of their datasets (regularity, scale, covariates) and the need for uncertainty quantification when choosing forecasting tools.
- *Practitioner Guidance*: This study informs practitioners about model strengths and weaknesses under varying scenarios, and highlights the need to consider dataset characteristics, interpretability requirements, and the importance of uncertainty when selecting and fine-tuning forecasting solutions.

6.3 Future Work

This thesis opens up several avenues for future research:

- *Hybrid Approaches*: Explore combining the strengths of Prophet's interpretability with the power of DL models to create hybrid forecasting solutions.
- *Additional Datasets*: Evaluate these models on datasets from other domains (e.g., finance, weather) to investigate their generalizability and performance in different contexts.
- *Expanding Scope*: Incorporate external factors like weather conditions or special events into models to enhance forecasting accuracy and inform proactive decision-making.
- *Real-World Deployment*: Investigate the challenges of deploying forecasting models in real-world operational settings, considering aspects like model maintenance, scalability, and integration with existing systems.
- *Limitations of Interpretability*: Investigate the factors hindering interpretability in theoretically interpretable architectures like N-BEATS.

- *Power of Attention Mechanisms*: Further exploration of attention-based models like TFT could yield insights into the key drivers and dependencies within complex time-series data.

6.4 Closing Statement

This thesis underscores the importance of understanding both models and data characteristics for effective forecasting. While deep learning offers notable advantages, approaches like Prophet or ARIMA retain value for simpler scenarios. Practitioners should carefully consider trade-offs between accuracy, interpretability, and uncertainty quantification. By continuing to explore the nuances of model performance and developing new techniques, the field of time series forecasting holds immense promise for data-driven decision-making across diverse domains.

Bibliography

[1] Sean J Taylor and Benjamin Letham. *Forecasting at scale*. *PeerJ Preprints*, 5:e3190v2, 2017.

[2] Rob J. Hyndman and Yeasmin Khandakar. *Automatic Time Series Forecasting: The forecast Package for R*. *Journal of Statistical Software*, 27(3):1–22, 2008.

[3] Vinod Nair and Geoffrey E. Hinton. *Rectified Linear Units Improve Restricted Boltzmann Machines*. *International Conference on Machine Learning*, 2010.

[4] Boris N. Oreshkin, Dmitri Carpov, Nicolas Chapados and Yoshua Bengio. *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*, 2020.

[5] Valentin Flunkert, David Salinas and Jan Gasthaus. *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks*. *CoRR*, abs/1704.04110, 2017.

[6] Bryan Lim, Sercan O. Arik, Nicolas Loeff and Tomas Pfister. *Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting*, 2020.

[7] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, Melbourne, Australia, 2nd edition, 2018. Accessed on 2023-01-01.

[8] David Salinas, Michael Bohlke-Schneider, Laurent Callot, Roberto Medico and Jan Gasthaus. *High-Dimensional Multivariate Forecasting with Low-Rank Gaussian Copula Processes*, 2019.

[9] Artur Trindade. *ElectricityLoadDiagrams20112014*. UCI Machine Learning Repository, 2015. DOI: https://doi.org/10.24432/C58C86.

[10] Marco Cuturi. *PEMS-SF*. UCI Machine Learning Repository, 2011. DOI: https://doi.org/10.24432/C52G70.

[11] Dimitrios Kollias, Anastasios Arsenos and Stefanos Kollias. *A deep neural architecture for harmonizing 3-D input data analysis and decision making in medical imaging*. *Neurocomputing*, 542:126244, 2023.

[12] Natalia Salpea, Paraskevi Tzouveli and Dimitrios Kollias. *Medical image segmentation: A review of modern architectures*. *European Conference on Computer Vision*, pages 691–708. Springer, 2022.

[13] Deema Abdal Hafeth, Stefanos Kollias and Mubeen Ghafoor. *Semantic Representations with Attention Networks for Boosting Image Captioning*. *IEEE Access*, 2023.

[14] Antonios Papaoikonomou, James Wingate, Vasudha Verma, Aiden Durrant, George Ioannou, Tasos Papagiannis, Miao Yu, Georgios Alexandridis, Abdelhamid Dokhane, Georgios Leontidis and others. *Deep learning techniques for in-core perturbation identification and localization of time-series nuclear plant measurements.* *Annals of Nuclear Energy*, 178:109373, 2022.

[15] Stefanos Kollias, Miao Yu, James Wingate, Aiden Durrant, Georgios Leontidis, Georgios Alexandridis, Andreas Stafylopatis, Antonios Mylonakis, Paolo Vinai and Christophe Demaziere. *Machine learning for analysis of real nuclear plant data in the frequency domain.* *Annals of Nuclear Energy*, 177:109293, 2022.

[16] Mamatha Thota, Stefanos Kollias, Mark Swainson and Georgios Leontidis. *Multi-source domain adaptation for quality control in retail food packaging.* *Computers in Industry*, 123:103293, 2020.

[17] Liyun Gong, Miao Yu and Stefanos Kollias. *Optimizing Crop Yield and Reducing Energy Consumption in Greenhouse Control Using PSO-MPC Algorithm.* *Algorithms*, 16(5):243, 2023.

[18] Bashar Alhnaity, Stefanos Kollias, Georgios Leontidis, Shouyong Jiang, Bert Schamp and Simon Pearson. *An autoencoder wavelet based deep neural network with attention mechanism for multi-step prediction of plant growth.* *Information Sciences*, 560:35–50, 2021.

[19] Ilianna Kollia and Stefanos Kollias. *A deep learning approach for load demand forecasting of power systems.* *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 912–919. IEEE, 2018.

[20] Dimitrios Kollias and Stefanos Zafeiriou. *Exploiting multi-cnn features in cnn-rnn based dimensional emotion recognition on the omg in-the-wild dataset.* *IEEE Transactions on Affective Computing*, 12(3):595–606, 2020.

[21] Andreas Psaroudakis and Dimitrios Kollias. *MixAugment & Mixup: Augmentation Methods for Facial Expression Recognition.* *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2367–2375, 2022.

[22] Fabio De Sousa Ribeiro, Francesco Calivá, Mark Swainson, Kjartan Gudmundsson, Georgios Leontidis and Stefanos Kollias. *Deep bayesian self-training.* *Neural Computing and Applications*, 32(9):4275–4291, 2020.

[23] Fabio De Sousa Ribeiro, Georgios Leontidis and Stefanos Kollias. *Capsule routing via variational bayes.* *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3749–3756, 2020.

[24] Fabio De Sousa Ribeiro, Georgios Leontidis and Stefanos Kollias. *Introducing routing uncertainty in capsule networks.* *Advances in Neural Information Processing Systems*, 33:6490–6502, 2020.

[25] Dimitrios Kollias, Miao Yu, Athanasios Tagaris, Georgios Leontidis, Andreas Stafylopatis and Stefanos Kollias. *Adaptation and contextualization of deep neural network models*. *2017 IEEE symposium series on computational intelligence (SSCI)*, pages 1–8. IEEE, 2017.

[26] N Bouas, Y Vlaxos, V Brillakis, M Seferis and S Kollias. *Deep transparent prediction through latent representation analysis*. *arXiv preprint arXiv:2009.07044*, 2020.

[27] Rob J Hyndman, Anne B Koehler, Ralph D Snyder and Simone Grose. *A state space framework for automatic forecasting using exponential smoothing methods*. *International Journal of Forecasting*, 18(3):439–454, 2002.

[28] Rob Hyndman, George Athanasopoulos, Christoph Bergmeir, Gabriel Caceres, Leanne Chhay, Mitchell O'Hara-Wild, Fotios Petropoulos, Slava Razbash, Earo Wang and Farah Yasmeen. *forecast: Forecasting functions for time series and linear models*, 2023. R package version 8.21.1.

[29] Andrew C. Harvey and Neil Shephard. *10 Structural time series models*. *Econometrics*, volume 11 in *Handbook of Statistics*, pages 261–302. Elsevier, 1993.

[30] Trevor Hastie and Robert Tibshirani. *Generalized Additive Models: Some Applications*. *Journal of the American Statistical Association*, 82(398):371–386, 1987.

[31] Everette S. Gardner Jr. *Exponential smoothing: The state of the art*. *Journal of Forecasting*, 4(1):1–28, 1985.

[32] G. E. P. Box and G. M. Jenkins. *Some Recent Advances in Forecasting and Control*. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 17(2):91–109, 2018.

[33] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li and Allen Riddell. *Stan: A Probabilistic Programming Language*. *Journal of Statistical Software*, 76(1):1–32, 2017.

[34] Richard H. Byrd, Peihuang Lu, Jorge Nocedal and Ciyou Zhu. *A Limited Memory Algorithm for Bound Constrained Optimization*. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[35] Spyros Makridakis and Michèle Hibon. *The M3-Competition: results, conclusions and implications*. *International Journal of Forecasting*, 16(4):451–476, 2000. The M3-Competition.

[36] Spyros Makridakis, Evangelos Spiliotis and Vassilios Assimakopoulos. *The M4 Competition: 100,000 time series and 61 forecasting methods*. *International Journal of Forecasting*, 36(1):54–74, 2020. M4 Competition.

[37] George Athanasopoulos, Rob J. Hyndman, Haiyan Song and Doris C. Wu. *The tourism forecasting competition*. *International Journal of Forecasting*, 27(3):822–844, 2011.

Special Section 1: Forecasting with Artificial Neural Networks and Computational Intelligence Special Section 2: Tourism Forecasting.

[38] Y. Bengio, S. Bengio and J. Cloutier. *Learning a synaptic learning rule*. *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 969 vol.2–, 1991.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *Deep Residual Learning for Image Recognition*, 2015.

[40] Gao Huang, Zhuang Liu, Laurensvan der Maaten and Kilian Q. Weinberger. *Densely Connected Convolutional Networks*, 2018.

[41] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-Term Memory*. *Neural Computation*, 9(8):1735–1780, 1997.

[42] Kyunghyun Cho, Bartvan Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014.

[43] Alex Graves. *Generating Sequences With Recurrent Neural Networks*, 2014.

[44] Andrew Gordon Wilson and Zoubin Ghahramani. *Copula Processes*, 2010.

[45] Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer Berlin Heidelberg, 1 edition, 2005.

[46] James Durbin and Siem Jan Koopman. *Time Series Analysis by State Space Methods*. Oxford University Press, 2012.

[47] Luc Bauwens, Sébastien Laurent and Jeroen V. K. Rombouts. *Multivariate GARCH models: a survey*. *Journal of Applied Econometrics*, 21(1):79–109, 2006.

[48] Ben S. Bernanke, Jean Boivin and Piotr Eliasz. *Measuring the Effects of Monetary Policy: A Factor-Augmented Vector Autoregressive (FAVAR) Approach**. *The Quarterly Journal of Economics*, 120(1):387–422, 2005.

[49] Laurent A.F. Callot and Anders Bredahl Kock. *238Oracle Efficient Estimation and Forecasting With the Adaptive Lasso and the Adaptive Group Lasso in Vector Autoregressions*. *Essays in Nonlinear Time Series Econometrics*. Oxford University Press, 2014.

[50] Royvan der Weide. *GO-GARCH: A Multivariate Generalized Orthogonal GARCH Model*. *Journal of Applied Econometrics*, 17(5):549–564, 2002.

[51] Robert Engle. *Dynamic Conditional Correlation*. *Journal of Business & Economic Statistics*, 20(3):339–350, 2002.

[52] Andrew J. Patton. *A review of copula models for economic time series*. *Journal of Multivariate Analysis*, 110:4–18, 2012. Special Issue on Copula Modeling and Dependence.

[53] C. Spearman. *"General Intelligence," Objectively Determined and Measured*. *The American Journal of Psychology*, 15(2):201–292, 1904.

[54] Donald Rubin and Dorothy Thayer. *EM algorithms for ML factor analysis*. *Psychometrika*, 47(1):69–76, 1982.

[55] H. Toutenburg. *Everitt, B. S.: Introduction to Latent Variable Models. Chapman and Hall, London 1984. 107 pp., £ 9.50*. *Biometrical Journal*, 27(6):706–706, 1985.

[56] Sam Roweis and Zoubin Ghahramani. *A Unifying Review of Linear Gaussian Models*. *Neural Computation*, 11(2):305–345, 1999.

[57] M. Sklar. *Fonctions de répartition à N dimensions et leurs marges*. *Annales de l'ISUP*, VIII(3):229–231, 1959.

[58] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, 2016.

[59] Scott Lundberg and Su In Lee. *A Unified Approach to Interpreting Model Predictions*, 2017.

[60] Djork Arné Clevert, Thomas Unterthiner and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, 2016.

[61] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. *Layer Normalization*, 2016.

[62] Yann N. Dauphin, Angela Fan, Michael Auli and David Grangier. *Language Modeling with Gated Convolutional Networks*, 2017.

[63] Yarin Gal and Zoubin Ghahramani. *A Theoretically Grounded Application of Dropout in Recurrent Neural Networks*, 2016.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin. *Attention Is All You Need*, 2023.

[65] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu Xiang Wang and Xifeng Yan. *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting*, 2020.

[66] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu Xiang Wang and Xifeng Yan. *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting*, 2020.

[67] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy and Dhruv Madeka. *A Multi-Horizon Quantile Recurrent Forecaster*, 2018.

[68] Markelle Kelly, Rachel Longjohn and Kolby Nottingham. *The UCI Machine Learning Repository*. `https://archive.ics.uci.edu`.

[69] Leonard J. Tashman. *Out-of-sample tests of forecasting accuracy: an analysis and review*. *International Journal of Forecasting*, 16(4):437–450, 2000. The M3- Competition.

[70] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta and Masanori Koyama. *Optuna: A Next-generation Hyperparameter Optimization Framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[71] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*, 2023.

[72] Leslie N. Smith. *Cyclical Learning Rates for Training Neural Networks*, 2017.