Master's thesis

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

Håkon Slåtten Kjærnli

# Improving the robustness of neural networks through augmentations with specific characteristics

Master's thesis in Informatics
Supervisor: Odd Erik Gundersen
June 2021

NTNU
Norwegian University of
Science and Technology

Håkon Slåtten Kjærnli

# Improving the robustness of neural networks through augmentations with specific characteristics

Master's thesis in Informatics
Supervisor: Odd Erik Gundersen
June 2021

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

NTNU
Kunnskap for ei betre verd

# Abstract

Neural networks are considered state of the art in many pattern recognition tasks like image classification and machine translation. However, it has been shown that when neural networks encounter data differing from what seen during training, referred to as out-of-distribution data, they often output a highly confident prediction. In a classification setting, the out-of-distribution data might not resemble any of the classes the network is trained to classify, yet the predictions remain confident. This thesis investigates the phenomenon in the domain of time series forecasting, which has not been considered by previous research on the topic.

An important prerequisite for research on out-of-distribution data is the acquisition of such data. By linking work related to generating time series with specific characteristics and work related to out-of-distribution data, a method for generating time series data is devised. The data generating method is then used to augment datasets with time series exhibiting different characteristics than those already present in a dataset. The results show that augmenting datasets with time series displaying specific characteristics allows model robustness to be increased in a highly controlled manner.

# Preface

The thesis is a conclusion of a Masters program in Informatics at the Norwegian University of Science and Technology in Trondheim, and was done in collaboration with TrønderEnergi AS. The supervisor was Odd Erik Gundersen, who I would like to thank both for helpful discussions and for making the project an enjoyable experience.

# Table of Contents

# List of Tables

# List of Figures

Chapter 1

# Introduction

This chapter introduces the context of the thesis, and describes the research approach, the research objectives and the results.

## 1.1 Background and Motivation

Time series and forecasting have been studied and applied in a wide range of settings like climate modeling, biology and medicine (Mudelsee, 2019; Stoffer and Ombao, 2012; Topol, 2019). Forecasting is applied to time series in order to gain information about the future, which can the be used to guide planning and decision making (Hyndman and Athanasopoulos, 2014). Traditional methods used to forecast time series are based on parametric models and relies on manual labor to find a model capable of representing the patterns characterizing the time series to be forecast. With the ever-increasing amount of available data, the manual labor associated with traditional methods have become a major hurdle. This hurdle, combined with the fact that the traditional methods are not developed to forecast large groups of time series exhibiting similar behavior, has inspired the development of methods that can be efficiently applied to large datasets without manual labor to produce accurate forecasts. Machine learning methods are in many ways a natural solution to this issue, seeing that the field is focused on making computer programs able to learn from data (Mitchell, 1997). While the application of machine learning methods to time series forecasting in many ways might seem straight forward, they have regularly been found inferior to traditional methods (Makridakis et al., 2018).

Neural networks, or deep learning, is a machine learning method that has become increasingly popular the last ten years. After it was shown that large neural networks could be applied to image classification and trained using (Cireşan et al., 2010; Krizhevsky et al., 2012), the field has experienced a surge of interest that in turn has produced breakthroughs in other domains where large amounts of data is available . Models driven by various forms of neural networks are now considered state of the art in fields like speech recognition, machine translation, object detection and image classification, in some cases reaching super-human performance (Chiu et al., 2018; Vaswani et al., 2017; Redmon et al., 2016;

Huang et al., 2017). As the size of datasets for time series forecasting increases, and given the success of neural networks when applied to large datasets, it seems logical to apply deep learning to time series forecasting with several examples of successful applications appearing recently (Salinas et al., 2020; Oreshkin et al., 2020; Zhou et al., 2021).

Having vast amounts of data available is usually seen as a necessity for deep learning based methods. In fact, even when there are already large amounts of data available, the size of the datasets are further increased by adding small changes to the each available data point through a process called augmentation. The data added by augmentation is often designed to increase the amount of available data and force the neural network to learn patterns that are more robust and general. As the use of deep learning models have become more prevalent, people have come to question if dataset augmentation is enough to ensure model robustness when encountering strange or unusual data, often refereed to as out-of-distribution data. The work of Szegedy et al. (2014) showed how images could be changed in imperceptible ways to make a neural network output a different label. Nguyen et al. (2015) showed a related result; it is possible to produce images completely unrecognizable to humans that a neural network will classify with 99% confidence. Research like this, combined with the fact that neural networks are highly complex and considered black box models, have made deep learning based methods somewhat infamous for its lack of robustness. In an attempt to rectify these issues an incredible amount of work has been publicized, yet definitive answers still remain illusive.

## 1.2 Problem outline

This thesis focuses on the robustness of time series forecasting models based on neural networks when encountering out-of-distribution data. The development of new forecasting models that are more robust is not considered. Instead, the work here focuses on increasing the robustness of already existing models.

There is a large amount of research both documenting and attempting to alleviate issues related to the robustness of neural networks when faced with out-of-distribution data. However, the work primarily focuses on classification tasks, with little work explicitly documenting the issue in the domain of time series forecasting. Nevertheless, it seems naive to assume that the method would not encounter similar issues when applied to time series forecasting. Because forecasting is commonly used to guide decision making and planning, it is of upmost importance to be able to understand and recognize conditions where a forecast can not be trusted. Without such an understanding the decisions being made on the basis of a forecast could lack important context or even be harmful.

This thesis documents research investigating how the problems outlined here can be solved, with both the acquisition of out-of-distribution data for time series forecasting and improving robustness of neural networks to such data being the central themes of the thesis.

## 1.3 Hypothesis, Objective and Research Questions

The hypothesis underlying this thesis is the following:

**HYP:** *Neural networks applied to time series forecasting can be made more robust by augmenting the dataset with out-of-distribution data.*

In many domains, datasets consists of a set of samples coming from some underlying distribution. This thesis investigates how data from different distributions affect the performance of neural networks in time series forecasting and if adding out-of-distribution data to a dataset increases robustness to such data.

When used in relation with neural networks, robustness is a term used to describe how resilient a network input of different types. Examples of different input types are noisy data, data coming from domains differing from the original training data and data specifically crafted to fool a network. Most work concerning the robustness of neural networks has been made in the context of classification, while the research made here only focusing on the time series forecasting task.

The objective of the thesis directly ties in with the hypothesis and is formulated as:

**O:** *Find out how out-of-distribution time series data can be generated and how it can be leveraged to increase the robustness of neural networks applied to time series forecasting.*

The main idea behind the objective was that to augment datasets with out-of-distribution data, the first step would be to understand how such data could be generated. Being in possession of such a method, out-of-distribution data could then be utilized to increase robustness in models. Seen through this perspective, the objective naturally divides itself into two parts, one concerning the generation of out-of-distribution data and the other concerning model robustness. Each of these two parts were investigated through its own research question:

**RQ1:** *How can out-of-distribution data for time series forecasting be generated?*

**RQ2:** *How can datasets for time series forecasting be augmented to improve the robustness of neural networks*?

**RQ1** was answered in partly through a literature review where a candidate method for generating out-of-distribution data was chosen. A set of experiments were then done to confirm that the method was able to produce out-of-distribution data. With a method for generating out-of-distribution data available, focus could be directed towards the second part of the objective and the second research question, **RQ2**. This research question was answered by testing multiple ways of augmenting datasets with out-of-distribution data across several datasets and models.

## 1.4 Research Approach

The research presented in this thesis was part of an exploratory study on the robustness of neural networks applied to time series forecasting. During the research, several different hypotheses were investigated, eventually leading to one of them being confirmed. Only the confirmed hypothesis, and the work related to it, is presented in this thesis. The research approach related to the last hypothesis can be divided into the following phases:

1. **Background study:** This phase consisted of getting acquainted with relevant theory and research in out-of-distribution data and time series augmentation. Having done so, the next part of the phase consisted of identifying open problems where making contributions were realistic, before formulating the research questions of the thesis.

2. **Development:** Having determined the research questions, development started. During this phase, the experiments of the thesis were designed and a runtime environment capable of executing the designed experiments was developed.

3. **Analysis:** The last phase consisted of analyzing the results from the experiments. The phase did not only consist of summarizing the results, but also entailed comparing the results with relevant research identified during phase 1.

## 1.5 Research Contributions

The research in this thesis has two contributions which are quickly described in this section. A more in depth discussion can be found in section 6.3.

**C1:** *Demonstrate an approach to generate out-of-distribution data for time series forecasting.*

An important part of this thesis is the method used to generate out-of-distribution data. As little research had studied which methods that were suitable to generate out-of-distribution data in a time series forecasting setting previously, analyzing suitable methods and devising an approach for generation was essential for further research into the topic.

**C2:** *Demonstrate an approach to improve the robustness of neural networks for time series forecasting when applied to out-of-distribution data.*

The main result of the thesis shows by using the method demonstrated in **C1** to generate out-of-distribution data, augmenting datasets in simple fashion produces a large increase in model robustness.

## 1.6 Thesis structure

This thesis is divided into the seven chapters, the first one being the introduction. The second chapter introduces background theory and the central topics of the thesis. In the third chapter, the state of the art of the two main research topics in the thesis is presented. The fourth chapter describes details related to the datasets, models, and experiments of the thesis. The results are presented in chapter five, before they are evaluated in chapter six. Finally, chapter seven concludes the thesis and suggests areas for future research.

Chapter 2

# Background

This chapter introduces topics and theory discussed throughout the thesis. The chapter starts by introducing theory about time series before moving on to Principal component analysis. Lastly, neural networks and out-of-distribution data are introduced. For a introduction into these topics the reader is referred to Hyndman and Athanasopoulos (2014); Gonzalez and Woods (2018); Goodfellow et al. (2016).

## 2.1 Time series forecasting

Forecasting is a task that consists of predicting the future as accurately as possible, given any available information. Available information can be both historic information and knowledge about future events. Methods for forecasting usually assume that patterns in the historic data will be repeated in the future, and attempts to capture these patterns to provide forecasts about the future development of a specific phenomenon.

A time series can be defined as a collection of data points $x = \{x_1, x_2, ..., x_n\}$ observed at the time points $1, 2, ..., n$. The time series forecasting task can be defined as using a vector of historic values $x \in \mathbb{R}^T = [x_1, ..., x_T]$ to produce a vector of forecasts $\hat{y} \in \mathbb{R}^H = [\hat{y}_{T+1}, ..., \hat{y}_{T+H}]$. The vector $\hat{y}$ is used to approximate the true future values of the time series $y \in \mathbb{R}^H = [y_{T+1}, ..., y_{T+H}]$, and the goal is for $\hat{y}$ to be as similar as possible to $y$. Here $T$ represents the length of the historic data, also called the lock-back window, and $H$ represents the number of time steps to forecast, also called the forecast horizon.

There are a multitude of methods that have been considered for time series forecasting. The simplest method for forecasting produces a forecast that is equal to the last observed value in the look-back window. Another example of a simple method is outputting the mean of the values in the look-back window. Simple methods such as these are referred to as naive. More advanced methods for time series forecasting include exponential smoothing, ARIMA and state space models. These models are called traditional, or statistical models, and are based on finding a set of parameters to model a time series as well as possible. For example, exponential smoothing attempts to find the parameters of a weighted

moving average to produce a forecast based on previous observed values.

It is possible to extend methods for time series forecasting so they do not only take historic values of the time series into account when forecasting, but also related variables. The related variables are called covariates or features. Covariates help forecasting methods learn patterns in the data that is the result of some external signal. For example, adding a covariate representing the day of the week could make it easier to forecast daily driving patterns on roads, as one can expect there to be less rush hour traffic during weekends.

When the forecast horizon $H$ consists of more than one time step, one has to consider how the forecasts for $H > 1$ are to be created. The two main approaches are called iterative and direct approaches. Iterative approaches consists of repeatedly forecasting one step ahead, and then feeding the one step ahead forecast back to the model to predict the subsequent step. Thus the forecasts from the model does not only depend on the look-back window, but also the forecasts of the same model from earlier time steps. The recursion can potentially lead to error accumulation in the forecasts of later horizons. The direct approach consists of simply forecasting all $H$ values in the forecast at once. Directly forecasting require $H$ to be specified but avoids the recursion used in iterative methods.

The performance of a forecasting method is usually quantified using a metric. The metrics that will be used in this thesis are the mean squared error (MSE), mean absolute error (MAE) and mean average scaled error (MASE) which are defined in the following way:

$$MSE = \frac{1}{H}\sum_{t=1}^{H}(y_t - \hat{y_t})^2 \tag{2.1}$$

$$MAE = \frac{1}{H}\sum_{t=1}^{H}|y_t - \hat{y_t}| \tag{2.2}$$

$$MASE = \frac{1}{H}\sum_{i=1}^{H}\frac{|y_i - \hat{y_i}|}{\frac{1}{T-m}\sum_{j=m+1}^{T}|y_j - y_{j-m}|} \tag{2.3}$$

### 2.1.1 Time series features and decompositions

It is possible to describe time series through a set of features designed to describe certain characteristics. Examples of features are simple statistics like the mean and variance of a time series, but more advanced features based on attempting to describe properties unique to time series like autocorrelations and decompositions are also available. Using features allows time series to be described with a feature vector $F = [f_1, f_2, ..., f_m]$ where each feature $f_i$ is calculated with some function applied to a time series $x$. This section presents the features used throughout the thesis, which are based on time series decompositions.

Time series are can be decomposed into three types of patterns called the trend, seasonal, and cyclic component. A trend is used to the describe the long term direction of growth in a time series, i.e. a long term increase or decrease. Seasonality describes the fluctuations happening at regular frequencies, and cycles describes fluctuations happening at irregular frequencies. The trend and cyclic components are often combined into one

component called the trend-cycle. The process of splitting a time series into these different patterns is a process called time series decomposition, and results in a trend-cycle, seasonal and remainder component. The remainder represents the values of a time series not explained by the trend-cycle or the seasonal component. Multiple seasonalities can be present in a time series and it is hence possible to decompose a time series into multiple seasonal components. A time series decomposition can be either additive or multiplicative. An additive decomposition can be defined as

$$x_t = T_t + S_t + R_t \tag{2.4}$$

where $x_t$ is the observed value of the time series at time $t$ and $T_t$, $S_t$ and $R_t$ correspond to the trend-cycle component, seasonal component and remainder component at the same time step. A multiplicative decomposition can be defined as

$$x_t = T_t \times S_t \times R_t \tag{2.5}$$

Additive decompositions are most suitable when the variance, both in terms of seasonal fluctuations and noise, of a time series is independent of both its level and the time step. That is, the variance is constant no matter the value of the time series at the point and does not increase or decrease as the length of observed time points increases. If variance is not constant, a multiplicative decomposition is better suited.

Sometimes one wants to analyze time series using only two of the three decomposed components, and especially one might want to analyze the detrended or seasonally adjusted series. These series are simply the time series without either the trend or seasonal component. The detrended series is then

$$S_t + R_t = x_t - T_t \tag{2.6}$$

and the seasonally adjusted series

$$T_t + R_t = x_t - S_t \tag{2.7}$$

The trend-cycle can be extracted using a moving averages. Extracting the trend-cycle using a moving average can be described as:

$$T_t = \frac{1}{m} \sum_{j=-k}^{k} y_{y+j} \tag{2.8}$$

where $m = 2k + 1$ and represents the order of the moving average. Using a moving average allows for the value of the trend-cycle to be determined by multiple values close in time, which eliminates some noise and results in a smoother trend-cycle component. Moving averages can be extended, for example by using centered or weighted moving averages. Time series are commonly decomposed with more advanced methods than moving averages, like STL, X-11 and SEATS.

As mentioned, features based on time series decompositions are important in this thesis. These features use a combination of the components trend, seasonality and remainder to describe the strength of the trend and seasonality. The strength of the trend component can be defined as:

$$T_{str} = max(0, \frac{Var(R_t)}{Var(T_t + R_t)}) \tag{2.9}$$

The intuition behind the definition is that for data with strong trends, the seasonally adjusted data will have a larger variance than the remainder component causing the denominator to be larger than the numerator. Thus, the fraction in the equation will be close to 0 for series with a strong trend and the final feature value close to 1. Data with a small trend component will have denominator with a smaller value, yielding a reduced feature value.

The seasonal strength can be defined similarly:

$$S_{str} = max(0, \frac{Var(R_t)}{Var(S_t + R_t)}) \tag{2.10}$$

and is similar to Equation 2.9, but with the variance of the detrended data is used in the denominator.

Two more features will be used throughout this thesis: the trend slope and the trend linearity (Kegel et al., 2017). Both of these require first fitting a linear regression model to the trend component of a time series:

$$T_t = \theta_1 + \theta_2 \cdot t + \delta_t \tag{2.11}$$

where $T_t$ is the trend components, and $\theta_1$ and $\theta_2$ correspond to the intercept and slope of the linear regression model respectively. $t$ is a vector representing each time step in the trend component and $\delta_t$ is the residuals of the linear regression model. The trend slope can then be defined as $\theta_2$. $\theta_2$ defines the change in the absolute value of the trend component of a time series, and is thus not independent of scale. To alleviate this issue, it is simply divided by the mean value of the trend component:

$$T_{slope} = \frac{\theta_2}{\frac{1}{T}\sum_{i=1}^{T} T_i} \tag{2.12}$$

making the feature represent the percentage increase or decrease of the mean value per time step.

The final feature presented here is the trend linearity. It is defined in a similar way as the trend strength and seasonal strength (Equation 2.9 and Equation 2.10):

$$T_{lin} = max(0, 1 - \frac{Var(\delta_t)}{Var(T_t)}) \tag{2.13}$$

The feature can be explained by seeing that when the variation in residuals is small, the trend component can be fairly well approximated with a linear regression model. That indicates that the trend is linear and results in a feature value close to 1. If the variance in the residuals is large compared to the variance in the trend component, it indicates that the linear regression model was unable to approximate the trend well and that the trend component is nonlinear.

## 2.2 Principal Component Analysis

Principal component analysis, PCA, is a method used for dimensionality reduction. PCA attempts to reduce the dimensionality in high dimensional data while keeping as much variance as possible. Given a set of features $F = [f_1, f_2, ..., f_m]$, PCA finds a set of $n$ components that are both uncorrelated with each other and maintains the largest degree of variance.

Let $\mathbf{X}$ be a $n \times m$ matrix with each row corresponding to a time series and each column corresponding to a feature value. The columns of the matrix are all centered to have a mean of zero.

$$\mathbf{X} = \begin{bmatrix} x_{0,0}, x_{0,1}, \ldots, bx_{0,m-1}, bx_{0,m} \\ x_{1,0}, x_{1,1}, \ldots, bx_{1,m-1}, bx_{0,m} \\ \vdots \\ x_{n,0}, x_{n,1}, \ldots, bx_{n,m-1} bx_{n,m} \end{bmatrix} \tag{2.14}$$

The covariance matrix of $\mathbf{C}$ can then be computed with the equation

$$\mathbf{C_X} = \mathbf{X}^T\mathbf{X} \tag{2.15}$$

$\mathbf{C_X}$ is then a $m \times m$ matrix where each element $c_{i,j}$ correspond to the covariance between column $i$ and $j$ in $\mathbf{X}$. Since $\mathbf{C_X}$ is real and symmetric, it is always possible to find $m$ orthonormal eigenvectors for the matrix.

Let $\mathbf{A}$ be a $m \times m$ matrix where the columns are formed with the eigenvectors of $\mathbf{C_X}$. The matrix A can then be used to map $\mathbf{X}$ into another matrix $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{XA} \tag{2.16}$$

It can then be shown that the covariance matrix of $\mathbf{Y}$, $\mathbf{C_Y}$, is a diagonal matrix with elements on the diagonal equal to the eigenvalues of $\mathbf{C_X}$. The off-diagonal elements of $\mathbf{C_Y}$ being all zero means that the elements of $\mathbf{Y}$ are uncorrelated, thus the matrix $\mathbf{A}$ has been used to transform $\mathbf{X}$ to $\mathbf{Y}$, and to a space where each column in $\mathbf{X}$ is uncorrelated. Additionally, the eigenvectors can be sorted based on the eigenvalues, with a larger eigenvalues meaning a larger degree of variance is explained.

PCA can then be used for dimensionality reduction by selecting the $k$ columns of $\mathbf{A}$ with the highest eigenvalues, forming the matrix $\mathbf{A}_k$. $\mathbf{A}_k$ can then be used to transform $\mathbf{X}$ to a $k$ dimensional space with the following equation:

$$\mathbf{Y}_k = \mathbf{XA}_k \tag{2.17}$$

The number $k$ controls the degree of dimensionality reduction in PCA. This thesis uses PCA to reduce the amount of dimensions in a matrix consisting of four features down to two, as a tool to enable visualization of time series based on features.

## 2.3 Neural networks

Following the great success of neural networks in image classification, researcher are attempting to apply neural networks to several other domains. Time series forecasting is one of those domains. Neural networks are a simplified model of how neurons in the brain are connected and how these neurons receive, process and send signals to other neurons. There are several different ways to organize neurons, and the organization of neurons in a network is usually referred to as the network architecture. The most commonly used architectures applied to forecasting are called recurrent neural networks, convolutional neural networks and transformers.

To apply neural networks to time series forecasting, it first has to be formulated as a supervised learning task. Given a training dataset $\mathcal{D}_{train} : \{\mathcal{X}_{train}, \mathcal{Y}_{train}\}$ sampled from a unknown distribution $P(\mathcal{X}, \mathcal{Y})$. $\mathcal{D}_{train}$ consists of pairs $(x, y)$ of observed values $x \in \mathcal{X}$ and target values $y \in \mathcal{Y}$. We are usually interested in the true distribution $P(\mathcal{X}, \mathcal{Y})$, but because of large state spaces or a high amount of complexity, the calculation of the distribution in often intractable. Instead, machine learning is used to approximate $P(\mathcal{X}, \mathcal{Y})$ through $\mathcal{D}_{train}$.

A basic neuron in a neural network receive some input $x$ and produces some output $\hat{y}$. A weighted sum is created with the inputs and the sum is transformed with what is called an activation function to create the final output. The weighted sum can be expressed as $z = f(\sum_{i=1}^{n} w_i x_i + b)$, where $w_i$ is the weight associated with the i-th input $x_i$, $b$ is the bias term of the neuron and $f$ is the activation function. Because the weighted sum can be described as a standard linear transformation, the activation function $f$ need to be non-linear for the network to learn non-linear functions.

The simplest architecture for a neural network are called fully-connected, or dense, networks. These networks organize their neurons in several layers of different sizes and the neurons in each layer are only allowed to send information "forwards", i.e. to the subsequent layer. The number of neurons in each layer is called the network width, and the number of layers is called the network depth. Layers not receiving input from a previous layer in the network while also not being responsible for the final output of the network are called hidden layers. The layer receiving the initial input of the network is called the input layer and the layer producing the final output of the network is referred to as the output layer. The fully-connected architecture is called dense because the output of every neuron in one layer is fed to every neuron in the next layer. This means that to compute the output of a layer, the output of all neurons in the previous layer has to be computed. The computation of the output of the final layer in a network is called a forward propagation, or forward pass, of the network. A fully-connected network can be seen in Figure 2.1

Formally, a neural network can be seen as a function $f$ parameterized by the parameters $\theta$:

$$\hat{y} = f_\theta(x) \tag{2.18}$$

Neural networks are trained to optimize a task formalized by a loss function $\mathcal{L}$ given the set of training data $\mathcal{D}_{train}$. The training goal can then be defined as finding a set of parameters $\theta$ that optimizes $\mathcal{L}(y, f_\theta(x))$. In the time series forecasting setting, the MSE (Equation 2.1) is often used as a loss function.

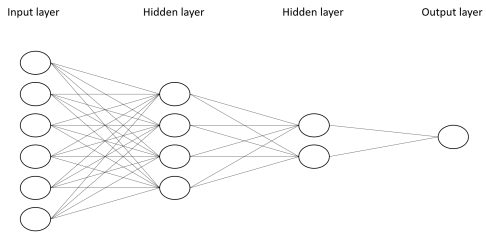



**Figure 2.1:** A fully connected neural network.

$\mathcal{L}$ is optimized by adjusting the weights of the neural network to find the best possible approximation. This is done by calculating the derivatives of each parameter in the network with the respect to the loss. The derivatives of each parameter in the network is calculated with the backpropagation algorithm, while the weights are adjusted with optimization algorithms like stochastic gradient descent.

### 2.3.1 Recurrent neural networks

Recurrent neural networks, or RNNs, are neural networks specialized for processing sequential data and have been utilized in domains like time series forecasting, language modeling and speech recognition. RNNs consists of a specialized memory structure that allows the output at time $t$ to not only depend on the input the network receives at that time, but also all previous input.

The RNN's memory structure is called the hidden state and the update of a hidden state $h$ at time $t$ given some input $x$ can be written as

$$h_t = f(h_{t-1}, x_t) \tag{2.19}$$

Thus, the hidden state at time $t$ is not only dependent on the input at time $t$ but also the hidden state at time $t-1$, which was dependent on the input at time $t-2$ and so on. This allows the hidden state to contain a memory of all previous inputs to the network. Figure 2.2 shows the structure of a simple RNN.

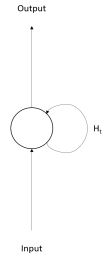



**Figure 2.2:** A recurrent neural network.

The hidden state of the network can be seen as a memory of what has previously happened. This memory is of a constant size and as a consequence it will, for long sequences,

start to forget old information. The limited memory causes RNNs to have problems learning long-term dependencies in the input.

Another issue that arises with long sequences is what is called exploding and vanishing gradients. During the computation of the gradients in the network, the repeated multiplication of a weight matrix to the hidden state will cause the gradients to either vanish or explode, making the network hard to train.

The two aforementioned issues are the motivation behind the a more advanced type of RNN called the Long Short Term Memory networks, or LSTM (Hochreiter and Schmidhuber, 1997). LSTMs has the same overall structure as a standard RNN, but introduces a cell state in addition to the hidden state. A single LSTM cell contains several gates, each with a specific purpose.

The first thing that happens in a LSTM cell at time $t$ is the concatenation of the previous hidden state $h_{t-1}$ and the input $x_t$. These to vectors are used as the input to all other gates in the cell at that time step.

The forget gate decides which parts of the cell state $c_{t-1}$ that should be forgotten at time $t$. It can be defined as

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{2.20}$$

where $\sigma$ is the sigmoid activation function $w_f$ is the weight matrix of the gate and $b_f$ is the bias of the gate. The output of this gate is then used in an element-wise multiplication. Since the sigmoid function outputs values between 0 and 1, the individual output values of the forget gate can be seen as a quantification of how much we want to forget of each element in the cell state $c_{t-1}$. A value of 0 indicates that the element should be completely forgotten, and a value of 1 indicates that the element should be unmodified.

The second gate is called the input gate, and decides how the values of the cell state should be updated. This layer is identical to the forget gate and defined similarly:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{2.21}$$

The output of the input gate is combined with a set of modifications defined as:

$$\tilde{C}_t = \sigma(w_{\tilde{c}}[h_{t-1}, x_t] + b_{\tilde{c}}) \tag{2.22}$$

$i_t$ and $\tilde{C}_t$ are combined through a element-wise multiplication and added to the cell state $C_t$. The intuition behind $i_t$ and $\tilde{C}_t$ and how they are combined is that $i_t$ decides which elements of the cell state we will update while $\tilde{c}_t$ decides how selected elements will be modified.

When $f_t$, $i_t$ and $\tilde{C}_t$ have been computed the cell state at time $t$ is given by the equation

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.23}$$

After the cell state has been updated it is used together with the output of another gate, the output gate. The output gate is also layer with a sigmoid activation function, this time deciding the parts of the cell state that should be used as outputs. It is defined similarly to the other gates:

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{2.24}$$

Finally, $o_t$ is combined with the hyperbolic tangent of the cell state to produce the next hidden state:

$$h_t = o_t * tanh(C_t) \tag{2.25}$$

$h_t$ is the output of the LSTM at time $t$ and is also passed on to the cell at time $t + 1$ together with $C_t$.

When applied to sequence modeling, RNNs are typically built with a structure called a sequence to sequence model (Sutskever Google et al., 2014). These models are divided into an encoder and a decoder. The encoder takes the input sequence and encodes the information to a hidden state $h_t$. The hidden state of the encoder is then used as input to the decoder, which then produces the the output of the model. Due to the importance of mapping long input sequences to a hidden state which is later passed to the decoder, RNNs like LSTMs or GRUs are generally preferred in these types of architectures.

### 2.3.2 Convolutional neural networks

Convolutional neural networks, or CNNs, are neural networks based on convolutions. CNNs was the driving force behind the success of neural networks in the domain of image classification and has also been applied to time series data. Standard convolutions are usually not applied to time series forecasting, with dilated causal convolutions being used instead. CNNs based on dilated causal convolutions are commonly called temporal convolutional neural networks, or TCNs.

The convolutional operation can be defined as

$$s(t) = (x * w)(t) = \sum_{m=-M}^{M} x(t-m)w(m) \tag{2.26}$$

where $t$ is the time index, $x$ is the input to the convolution, $w$ is the kernel and $2M + 1$ is the size of the kernel. The output of a convolutional operation in a neural network is calculated by sliding the kernel over all values of the input $x(t)$.

The problem with applying convolutions to one dimensional data like time series is that one needs many layers or large kernel sizes to increase the amount of historic data used to predict a single point. Furthermore, it is not obvious how a kernel should be applied to avoid using future information when making predictions. WaveNet (van den Oord et al., 2016) proposed to solve these problems by stacking dilated causal convolutions, which has become the standard way of applying convolutions to time series data.

Causal convolutions are simply convolutions where the output of a convolution is only dependent on past time points, i.e. $f(x_1, x_2, ..., x_{t-1}) = \hat{y_t}$ where $f$ is a CNN. This is usually achieved by padding the input of the convolution on the left side and then applying the convolutional operator as usual.

Dilated convolutions are convolutions where the kernel is applied to a larger number of values than the length of the kernel. This is done by skipping a set of values in the input for each value in the kernel. For example, a kernel with size 2 and a dilation factor of 2 will apply the kernel over an area of 3 input values at the time, skipping the middle value. A convolution with a dilation factor of 1 results in a standard convolutional filter.

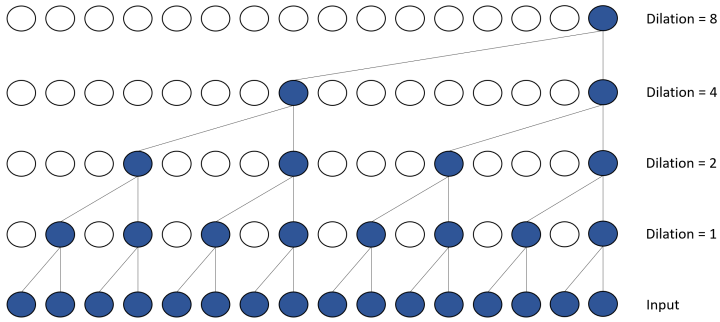



**Figure 2.3:** A stack of dilated causal covolutions. Only the blue nodes contribute to the final output.

WaveNet proposed to use multiple layers of dilated causal convolutions, with the dilation factor being doubled every layer. This greatly increases the number of historic values a single prediction is conditioned on, while keeping the total number of computations necessary at a manageable level. A stack of causal and dilated convolutions can be seen in Figure 2.3. It should also be noted that the number of layers then directly decides the length of the input to the model. In TCNs, the length of the input is often called a receptive field.

Because the kernel in a CNN is reused at all positions in the input, and the kernel is typically much smaller than the number of inputs. The total number of parameters in CNNs are greatly reduced when compared to fully connected networks. Thus, CNNs resemble RNNs in the way that both architectures reuse parameters. However, RNNs reuse parameters at different time points in the input while CNNs reuse parameters at different locations. For sequential data like time series, reusing parameters at different time points and at different locations is equivalent.

### 2.3.3 Transformers

Transformers are a fairly new architecture that was first applied to machine translation (Vaswani et al., 2017). Since then they have been applied to several other problems, and recently they have started being used in time series forecasting (Wu et al., 2020; Li et al., 2019; Lim et al., 2019; Zhou et al., 2021). These models do not implement any type of recurrence or convolutions and instead rely solely on a mechanism called attention. Transformers typically consist of an encoder and a decoder where the key part is the use of the multi-head attention mechanism.

Attention, or self-attention, is calculated using three matrices called the Query, Key and Value, or just $Q$, $K$ and $V$, which are learned during the training of the neural network. An input $x$ is multiplied with three different weight matrices $W_Q$, $W_K$ and $W_V$, producing $Q$, $K$ and $V$. The output of the attention layer can be defined as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}}V) \tag{2.27}$$

where $d_k$ is the dimensions of the Query and Key matrices.

Transformers are said to use multi-head attention because they have multiple sets of $Q$, $K$ and $V$ matrices. Using multiple "heads" of attention allows the transformer the attend to the input using multiple different representations. The output of each different attention head is concatenated, and then multiplied with a matrix $W^O$ projecting it back to the same number of dimensions as the input to the multi-head attention layer. The final projection is necessary because of a residual connection over the multi-head attention layer.

The encoder in a transformer contains several identical blocks containing two main layers. The first layer in a block is a multi-head attention layer followed by a residual connection. The output of the residual connection is then passed through a LayerNorm (Ba et al., 2016), normalizing the input, before arriving at a position-wise fully connected layer. This layer is applied to all the positions in the output from the multi-head attention individually. That is, for each output of the self-attention layer describing how a single input position is related to all other position, the position-wise layer is applied, and this is done for all positions. As with the multi-head attention layer, the position-wise fully connected layer is followed by a residual connection and a LayerNorm.

The transformer decoder has a similar structure to the encoder, but with one additional sub-layer for each block. The first sub-layer is a multi-head attention layer. Masked self-attention can be used in this layer to prevent the decoder from attending to subsequent positions in the input. The output of the layer is a matrix used as the query $Q$ in the next layer in the decoder, the encoder-decoder attention layer.

The encoder-decoder attention layer is the layer where the decoder integrates the output from the encoder into its own state. The $K$ and $V$ matrices in this layer come from the final output of the encoder, and the $Q$ matrix is the output of the previous layer in the decoder. One could say that the encoder-decoder attention layer allows the encoder to present interesting information about the input sequence to the decoder through $K$ and $V$, while the decoder queries for specific information through $Q$.

The final layer in a decoder block is a position-wise fully connected layer. As with the encoder, all sub-layers in the decoder are followed by a residual connection and a LayerNorm.

In a transformer the encoder and the decoder will consist of multiple blocks identical to the ones described above. The final output of the decoder is passed through a fully connected layer to project the output to the desired number of dimensions.

The attention mechanism is permutation invariant meaning that ordering of the inputs does not affect the output of the mechanism. To add information about absolute and relative positions in the input, transformers adds a positional encoding. The positional encoding takes the form of:

$$PE(pos, 2i) = sin(\frac{pos}{1000^{\frac{2i}{d_{model}}}}) \tag{2.28}$$

$$PE(pos, 2i+1) = cos(\frac{pos}{1000^{\frac{2i}{d_{model}}}}) \tag{2.29}$$

where $pos$ and $i$ correspond to the position and the dimension of the input. $d_{model}$ represents the number of dimensions in the input and output of the model. The positional encoding thus inserts positional information in the input by adding a unique sinusoidal wave to each of the input dimensions.

A key advantage of transformers when compared to CNNs and RNNs is the number of computations between long-range dependencies in the network. In RNNs the full hidden state is computed for each element in the input sequence. Thus, for an input sequence of length $t$, there are $t - 1$ calculations of the hidden sate between the first and the last state. In a similar fashion, the first and last input in a dilated causal CNN will have to pass through all but the last filter before they are combined. The large number of computations between inputs far apart in the input sequence increases the probability of information being lost, thus making it harder to learn long-range dependencies. On the other hand, a transformer's use of self-attention computes the dependencies between every combination of elements in the input sequence, reducing the number of computations to one for every pair of elements in the input sequence.

### 2.3.4 Out-of-distribution data

Neural networks are often trained using highly restricted datasets. The data encountered during training is commonly referred to as in-distribution data, while data outside of the training distribution is referred to as out-of-distribution data. A network's performance might degrade considerably when applied to out-of-distribution data, and it is thus important to be able to distinguish between in-distribution and out-of-distribution data.

When neural networks are asked to predict data that is not similar to what they have seen during training, one would like them to express some sort of uncertainty. In classification, the output of the neural network is commonly passed through the softmax function to create a probability distribution. For data that does not resemble the data encountered during training, it would then be appropriate if the probability distribution became more uniform to reflect some notion of uncertainty. Unfortunately, neural networks have been shown to produce overconfident probability scores for such data (Goodfellow et al., 2015; Nguyen et al., 2015).

Out-if-distribution, OOD, data can be defined as a set of data $\mathcal{D}_{OOD} \sim \hat{P}(\mathcal{X}, \mathcal{Y})$ that is sampled from a different underlying distribution than the training data $\mathcal{D}_{train} \sim P(\mathcal{X}, \mathcal{Y})$. OOD data is usually studied through data that is either statistically out-of-distribution (Nguyen et al., 2015), or through adversarial attacks where so called adversarial examples are crafted to fool the network (Goodfellow et al., 2015). This thesis will only consider the former, where out-of-distribution data chosen statistically.

### 2.3.5 Data Augmentation

While data augmentation is a staple when applying neural networks to domains like image classification, it has not seen the same amount of use in time series forecasting. Still, data augmentation is a key concept in this thesis and this will subsection introduce the very basic concepts. The reader is referred to Shorten and Khoshgoftaar (2019) for a comprehensive review of augmentation techniques.

Data augmentation increases the amount of available data by artificially adding new data points, either through transformations adding small changes to already existing points or by generating entirely new points. The process does not only increase the overall size of the dataset, but also ensures that the dataset represents a more complete set of possible

data points. Making the dataset more complete is essential, as it forces the model to learn more robust and general patterns.

The simplest forms of augmentation rely on transformations like flipping, cropping and translations applied to data in the input space. Augmentations such as these rely on the corresponding task being invariant to them, meaning that the output of the neural network should not change when they are applied. Invariant transformations are called label preserving, and the set of such transformations vary depending on the domain. There are several more advanced methods for data augmentation, including transformations in feature space, adversarial training and synthesis of data using generative models.

Chapter 3

# State of the Art

This chapter reviews the state of the art of research in the fields time series augmentation and out-of-distribution data. The goal of the chapter is to present research in the topics and explain how the work in this thesis relates to it.

## 3.1 Out-of-distribution data

This section reviews the state of the art in research made on the behavior of neural networks when faced with out-of-distribution, OOD, data. The approaches to increase robustness to OOD data can roughly be divided into three main categories: by leveraging a classifier's uncertainty, by using a binary classifier or by using a generative model.

When neural networks are applied to classification, a common approach to detect OOD data is to attempt leverage the classifier's uncertainty (Hendrycks and Gimpel, 2017; Liang et al., 2018). Hendrycks and Gimpel (2017) found that the prediction probability of incorrect and OOD points where usually lower than for points predicted correctly. By collecting statistics of prediction probabilities for correctly classified points the authors made a simple baseline for detecting OOD points by comparing the classifier's prediction probability with the collected statistics. Liang et al. (2018) increased the difference in prediction probability further by adding a temperature scaling to the softmax function and small perturbations to the input.

Lee et al. (2018) adds an additional term to the loss function that is designed to minimize the Kullback-Leibler divergence from the predicted distributions on OOD data to the uniform distribution. The extra term causes OOD data to produce more uniform probability distributions, but does require OOD data for training. The authors solve this by generating OOD data with a Generative Adversarial Network, or GAN (Goodfellow et al., 2014). The work in Hendrycks et al. (2019) is similar in the way that they add an additional term to the loss function, but by using auxiliary datasets as OOD data they avoid the need for a generative model.

Several works train a separate model to check if a data point is in- or out-of-distribution. This can be done by using a binary classifier, by adding an additional class to a classifier

or by using a generative model (DeVries and Taylor, 2018; Vernekar et al., 2019; Wang et al., 2017). By adding a confidence branch to a classifier, DeVries and Taylor (2018) makes a neural network able to output both the prediction probabilities and the confidence in its own prediction. The confidence can then be used to detect OOD data by classifying all points with a confidence less than some threshold as OOD.

Vernekar et al. (2019) takes another approach and argues that adding an additional class to explicitly predict OOD data restricts the decision boundaries of the classifier. The idea is that classifiers that only model the original classes are not forced to learn decision boundaries that are restricted to in-distribution data. Explicitly modeling OOD data as a separate class restricts the decision boundaries of the original classes to areas in the training distribution, and creates a new decision boundary for OOD data outside the training distribution.

When using generative models to detect OOD data, the most straight forward approach is to use the calculated the probability of a point being generated by the distribution seen during training. Pidhorskyi et al. (2018) propose to train an autoencoder to model the distribution of the training data. Data is then predicted as OOD if the probability of originating from the learned distribution is lower than some threshold. Wang et al. (2017) learns a generative model for each class in a dataset, and classifies data by finding the generative model with the lowest reconstruction loss for a given input. If the reconstruction loss can then be examined to determine if an input is likely to be OOD.

While these two preceding approaches seem reasonable, it has been shown that deep generative models also are susceptible to assigning high probability to OOD data (Hendrycks et al., 2019; Choi et al., 2018; Nalisnick et al., 2018). Ren et al. (2019) attempts to address the problem by using a likelihood ratio test while Morningstar et al. (2020) suggests a method called Density of States Estimation. Finally, Meinke and Hein (2019) suggest combining a Gaussian mixture model with a neural network and proves that low confidence predictions will be provided for data far away from the training distribution.

All the reported work summarized in this section deals primarily with classification. In fact it appears that no research has yet been made on how OOD data affects deep learning for time series forecasting. However, given the results from research done on classification it seems reasonable to assume that neural networks would suffer from the same issues when applied to time series forecasting. There are however some key differences between classification and forecasting that distinguishes the two domains and limits what research that can be directly applied to the forecasting setting. One of these differences is that when applied to forecasting, neural networks do not output a probability distribution over a set of classes and instead produce either a point forecast or a distribution. When producing a distribution one would like the forecast to display a higher degree of uncertainty for OOD data, but this seems unlikely given the behavior of the softmax distribution in classifiers. Another key difference is that the output of the forecasting task is continuous and the output is not separated into classes, making efforts related to restricting a classifiers decision boundary not immediately applicable. Lastly, in a classification setting robustness to OOD data can be reduced to detecting such data because it will not belong to any of the classes of the in-distribution data. In a forecasting setting, one might not want to abstain from forecasting OOD data and hence simply detecting if data is OOD does not provide a satisfactory solution.

## 3.2 Time series augmentation

Despite the importance of data augmentation in the fields like computer vision and speech recognition (Shorten and Khoshgoftaar, 2019; Cui et al., 2015), less work has been done attempting to find and improve augmentation techniques for time series data. This section reviews the state of the art in time series data augmentation.

Cui et al. (2016) uses CNNs for time series classification and suggests cropping time series and is similar to the cropping augmentation used in computer vision. A time series $Y = y_1, ..., y_n$ is cropped into a group of smaller series $S_{i:j} = y_i, ..., y_j$ for $1 \leq i \leq j \leq n$. Each cropped time series $S_{i:j}$ has the same label as the original time series $Y$ thus causing an increase in the amount of data available per class. Guennec et al. (2016) builds upon this work by adding a technique called window warping. Window warping selects an interval and either up samples or down samples the amount of observations within the interval. It is combined with cropping to ensure that individual time series were of equal length.

Um et al. (2017) applies a CNN to classification of wearable sensor data and suggests several fairly simple augmentation techniques like jittering, scaling, rotating, cropping and permuting time series. The authors also use a technique they called time warping, which consists of up sampling some intervals in a time series while down sampling others to make sure the length of the time series stays the same. The results show that adding cropping, scaling and jitter to the time series do not improve the performance of the CNN, but rotations, permutations and time warping do.

Forestier et al. (2017) suggests augmenting sparse datasets by using an extension of Dynamic Time Warping to average a subset of the training data to create new time series. The averaging is done using a weighted average, enabling several new time series to be created per set of time series by changing the weights. This technique was later used in Fawaz et al. (2018) to boost the performance of a neural network for time series classification.

Instead of applying trasformations in the input space, DeVries and Taylor (2019) proposes to transform data in the learned feature space of a neural network. The authors employ an autoencoder to learn a feature space. Once an autoencoder is trained, the dataset can be augmented by projecting each example into the feature space. The representations are modified by adding noise, or by either interpolating or extrapolating between examples close in the feature space. New data can then be generated by decoding the modified representations.

The work presented above has only been applied to time series classification. In work applying deep learning to the time series forecasting task it appears augmentation is not commonly used (Salinas et al., 2020; Oreshkin et al., 2020; Zhou et al., 2021). A possible reason behind this is the fact that traditional forecasting methods often provide a simpler and superior alternative to deep learning based methods when the size of a dataset is small. In fact, deep learning based methods was considered inferior to traditional methods for a long time (Makridakis et al., 2018), and the effectiveness of deep learning on large datasets was not demonstrated before the work of Salinas et al. (2020) learned a single global model to forecast datasets with a large number of unique time series exhibiting similar patterns. In a work researching the effects of data agumentation in smaller datasets for time series forecasting, Bandara et al. (2020) augments data using the techniques presented

in Kang et al. (2020); Bergmeir et al. (2016); Forestier et al. (2017). The techniques are assessed through either pooling augmented and original data to one big dataset, or by transferring models trained on generated data to original data. The results indicate that it is advantageous to either use generated data that is similar to the training distribution, or generate highly diverse data for training before transferring the learned representations to the original dataset with some retraining of the model.

The methods used by Bandara et al. (2020) were not explicitly designed to augment time series data for the use of deep learning. Nevertheless, Bandara et al. (2020) showed that they could be used in that context. There are a large amount of research concerning generation of time series data that is not directly linked with deep learning. Some of these methods are presented next.

Bergmeir et al. (2016) generates new time series by first decomposing them into trend, seasonal and residual components. The residual component is then bootstrapped before adding the components back together. This generates multiple new time series, although with limited diversity. The work of Iftikhar et al. (2017) is similar, but also clusters time series. The bootstrapped residuals are then exchanged between time series within the same cluster.

Kang et al. (2017) investigated how different forecasting methods perform on time series with specific characteristics, and if any systematic differences could be uncovered between methods through such an analysis. To be able to do so, they represent each time series in a dataset through a feature vector and projects it into a instance space by applying PCA before visualising the first two components. The instance space can then be inspected and the authors devise an algorithm which enables generating new time series at specific areas in the instance space not well covered by the datset. Specifically, a genetic algorithm is leveraged to evolve time series with characteristics placing them at targeted areas in the instance space. The evolution is guided by calculating the feature vectors for the generated time series and performing a selection based on the distance in instance space to the targeted area. In a later paper (Kang et al., 2020), the authors improve the method by simulating new time series using Gaussian mixture autoregressive models. The proposed solution avoids directly optimizing all values in a time series, and instead optimizes the parameters of Gaussian mixture autoregressive models, with the optimization still being done by a genetic algorithm.

In a paper related to Kang et al. (2017), Kegel et al. (2017) use STL to decompose time series into trend, seasonal and residual components. The trend and seasonal components are then transformed generating new time series. The procedure in Kegel et al. (2017) is much faster than the ones in Kang et al. (2017, 2020), but also less flexible. In Kegel et al. (2018) the authors build upon their own work and allow new time series to be generated by assembling trend and seasonal components from different time series and simulating a new residual component. The generated time series are constructed in such a way that they are close to some targeted feature vector. However, since it is based on a recombination of already existing time series generating time series with features not similar to already existing time series might prove problematic.

Deep generative models have also been used to generate time series data, most often through GANs. Esteban et al. (2017); Mogren (2016) uses a GAN where both the generator and discriminator are chosen to be LSTMs to generate new time series data. Ramponi

et al. (2018) also uses a GAN to generate time series data, but uses CNN instead of a RNN. Another example is Chen et al. (2018), which uses a bayesian GAN to generate time series for wind and solar energy. The main issue with these methods is that the generative models are not modified to take into account temporal dependencies. In Yoon et al. (2019) the authors argue that creating observations for time steps individually and summing the loss per time step, as is done in other GAN-based approaches, is insufficient to capture the temporal dependencies present in time series. The key difference from the other approaches is that the the noise input given to the generator per time step follows a Wiener process, making the input to the generator at time $t$ dependent on the input at time $t - 1$.

This section has reviewed state of the art in time series augmentation and uncovered that while there is an abundance of methods for generating new time series data, they are seldom applied to time series forecasting with deep learning. There is some research investigating the use of augmentation in time series classification, but these methods might be highly problem specific due to the fact that certain transformations might be label preserving in some domains but not in others. For example, Cui et al. (2016); Guennec et al. (2016) apply cropping successfully but Um et al. (2017) finds that it deteriorates performance. On the other hand, the amount of work being done on generating time series in a context not directly related to augmentation data for neural networks is large and as Bandara et al. (2020) shows, these methods enable interesting types of research into neural networks.

Lastly, the work of Yoon et al. (2019) separates it self from the other research presented here in that it explicitly considers temporal dependencies. While methods like Kang et al. (2017) are able to generate time series with specific characteristics, no focus is given to how similar temporal dependencies can be ensured. The simple methods also do not consider temporal dependencies, and could perhaps be part of an explanation of why certain augmentations improve performance in some cases but not in others. While not explicitly stated in Kang et al. (2020), simulating new time series ensures that the temporal dependencies of the generated time series are similar as to the original data. The methods of Kegel et al. (2017, 2018); Bergmeir et al. (2016); Iftikhar et al. (2017) are also more likely to keep temporal dependencies intact, as generated data is based on already existing time series.

## 3.3 Summary

Both the state of the art concerning neural network's robustness to out-of-distribution and augmentation for time series data rarely considers time series forecasting. With one key challenge in research into robustness being the acquisition of data, some of the generative methods for time series data could potentially be leveraged, linking the two research fields reviewed in this chapter. This is especially true for the generative methods that allows for the creation of time series with specific characteristics, like Kang et al. (2017, 2020); Kegel et al. (2017, 2018). These methods could be used to generate out-of-distribution data in a controlled manner generating time series exhibiting characteristics differing from the in-distribution data.

Chapter 4

# Experiment details

This section describes details related the datasets, models, features, data generation and experiments of the thesis. First, the chapter provides an overview of the datasets, models, features and transformations used, then the details of each experiment conducted is given.

## 4.1 Datasets

The datasets used in this thesis are electricity, traffic and M4. The electricity and traffic are often used for testing forecasting methods based on deep learning while the M4 datasets have frequently been used to benchmark forecasting methods in general. This selection enables analysis of datasets where it well documented that deep learning based methods preform well, and on datasets with larger variety making learning a single global model more challenging.

Electricity contains time series representing the hourly electricity consumption of 370 clients. Most time series in the dataset display yearly, weekly and daily patterns. However, most research only use a single week as input to their model and a horizon of 24 hours making the yearly patterns irrelevant. The result is that the input time series are dominated by the daily seasonality and has little or no trend.

Traffic is a dataset where the time series are collected from the hourly occupancy rates of 963 different car lanes. The time series has strong daily and weekly patterns, and similarly to electricity, most research use a look-back window of a single week and a horizon of 24 hours. Both the Electricity and Traffic datasets use the last 7 horizons of all time series as test data and the 7 horizons preceding the test data was as validation data.

The M4 datasets are divided into hourly, daily, weekly, monthly, quarterly and yearly subsets. These subsets are highly diverse, both when compared to each other and when compared internally. The internal diversity makes it harder to apply single global model for all time series, making them more challenging to forecast than electricity and traffic for deep learning based methods. The time series in each subset vary in length and can exhibit different types of seasonality and trends. For all subsets, the last horizon of each time series in the training data was used as validation data, omitting time series that were

to short to still have a full look-back window and horizon after the removal of a horizon for validation purposes. See Table 4.1 for the size of the look-back window and horizon of all datasets.

| Dataset | Look-back window | Horizon |
|---|---|---|
| Electricity | 168 | 24 |
| Traffic | 168 | 24 |
| M4 yearly | 12 | 6 |
| M4 quarterly | 16 | 8 |
| M4 monthly | 36 | 18 |
| M4 weekly | 26 | 13 |
| M4 daily | 28 | 14 |
| M4 hourly | 96 | 48 |

**Table 4.1:** The size of the look-back window and horizon for all datasets.

## 4.2 Models

The models used in the thesis were chosen to represent some of the architectures presented in section 2.3. Five different models were used and these were a fully-connected network, a LSTM based sequence to sequence model, a TCN, a transformer and N-BEATS. The models varied in terms of number of parameters, with N-BEATS being the biggest followed by the LSTM. The TCN was the smallest model, with either the fully-connected or transformer model being the second smallest depending on the dataset.

The fully-connected neural network was a fully connected neural network similar to the ones described in section 2.3. It had two hidden layers, each with 100 neurons using the ReLU activation function. It directly produced forecasts for all time steps in the horizon.

The LSTM was a sequence to sequence architecture that works as described in subsection 2.3.1. The model was autoregressive and was unrolled during both training and testing to produce the forecast for the horizon. Both the encoder and decoder had 2 layers, and the size of the hidden state was 128. Figure 4.1 shows the LSTM as a sequence to sequence model.

The TCN was based on the model proposed by Borovykh et al. (2017). Borovykh et al. (2017) suggests a simplified WaveNet architecture mainly consisting of blocks of dilated causal convolutions and parameterized residual connections in the form of a 1x1 convolution. To allow the output of the model to be conditioned on covariates, the authors suggests combining the output of two separate blocks, receiveing the observed value of the time series and covariates, respectively. The output of the two blocks is then combined through a element-wise summation before being passed on to the next block in the network. The final layer of the network passes the output of the previous block through a layer of 1x1 convolutions. Figure 4.2 shows the structure of the model.

All blocks in the network use ReLU activation functions and the convolutional filters used 16 channels. During training the model produced a one-step prediction for each time

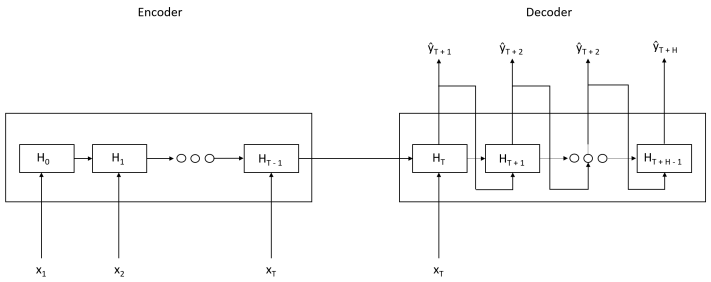



**Figure 4.1:** A sequence to sequence model.

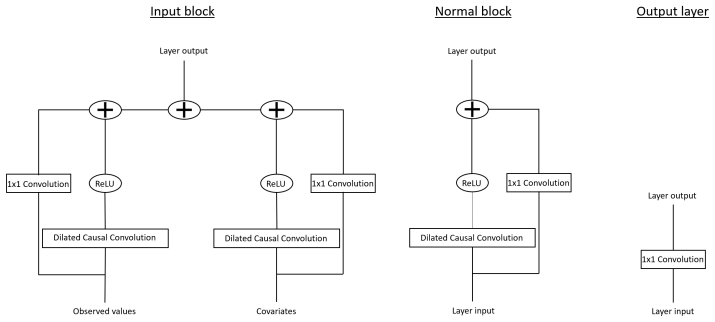



**Figure 4.2:** The different blocks of the TCN model.

step in the horizon, while during testing the model was unrolled by feeding the one-step forecast back into the model repeatedly, similarly to the LSTM. The dilation factor per block was $d = 2^l$ where $l$ is the layer, as described in subsection 2.3.2. Because the number of layers in a TCN directly decides the size of the receptive field, the amount of layers changes with the size of the look-back window of the dataset. For a given look-back window, the maximum number of layers was chosen such that the size of the receptive field was less than or equal to the size of the look-back window. Note that this might have given the model a disadvantage on some datasets, as the size of the receptive field might be significantly lower that the size of the look-back window.

The transformer had a architecture as described in subsection 2.3.3, with one important difference from Vaswani et al. (2017). In the original tranformer, the model is repeatedly unrolled during testing, similarly to the LSTM and TCN decribed above. In contrast, the implementation used here directly forecasts the entire horizon as is done in Zhou et al. (2021). The model had 4 layers in both the encoder and decoder, and used 4 attention heads. The size of the input dimension of the model was 12 and the size of the fully-connected layers layers were 64.

These four models were all trained by minimizing the MSE loss (Equation 2.1) and with the time series being standardized to a mean of zero and unit variances. Additionally, the LSTM, TCN and transformer models used covariates. For simplicity the calendar features hour of day, day of week, day of month, day of year and month of year were used as input for all datasets, regardless of seasonality. The calendar features were also

standardized. Lastly, each time series was given an unique ID, which was embedded in an embedding layer before passing it to the models.

The N-BEATS model was implemented as the generic block described in Oreshkin et al. (2020). The architecture is based on several blocks of fully-connected layers with residual connections. Each block produces both a backcast and a forecast. The backcast is added to the input to the block through a residual connection and passed as input to the next block. The forecast of each block is eventually summed to produce the forecast of the model. All experiments used 30 blocks with a layers size of 512. In Oreshkin et al. (2020) an ensemble of N-BEATS instances is used, and different instances uses different look-back windows and optimizes different loss functions. In contrast, this thesis only trains a single instance of the model by minimizing the MASE loss (Equation 2.3). The time series were not standardized before being passed to the N-BEATS model, and it did not use covariates as input.

The models where trained by sampling batches of windows at random positions from the original, full length time series in a dataset. All models used a batch size of $512$, $50$ batches were sampled per epoch and the Adam optimizer with a learning rate of $0.001$. During training, the models where tested on the validation data every fifth epoch. A learning rate scheduler decayed the learning rate by a factor of $0.5$ whenever the validation loss did not decrease for 10 evaluations in a row. If the learning rate dropped lower than $0.00005$ the training stopped. The models where trained for a maximum of 1000 epochs with early stopping.

## 4.3 Data generating method

The method used to generate out-of-distribution data was based on combining the work of Kegel et al. (2017) and Kang et al. (2017). The features used to describe the characteristics of time series and the transformations used to generate new data was the same as what is proposed in Kegel et al. (2017). This method for data generation was chosen primarily because of its simplicity while it is still able to generate time series with specific characteristics and possibly generating time series with similar temporal dependencies. To generate out-of-distribution data, the transformations and features was combined with the idea of an instance space from Kang et al. (2017). Embedding the data in an instance space allowed for the characteristics of in-distribution data to be identified. Out-of-distribution data could then generated by transforming the already existing time series in the dataset in such a way that the generated data possessed characteristics not present, or uncommon, in the dataset. This section describes the method for data generation in detail, starting with the set of features and instance space.

### 4.3.1 Features and visualization

The features used to describe time series in this thesis were based on the work of Kegel et al. (2017). The features used were $T_{str}$, $S_{str}$, $T_{slope}$ and $T_{lin}$ defined in Equations 2.9, 2.10, 2.12 and 2.13. After calculating the features of all time series, PCA was used to project the features to a two dimensional instance space, similar to what was done in Kang et al. (2017).

| Dataset | Seasonal period |
|---|---|
| Electricity | 24 |
| Traffic | 24 |
| M4 hourly | 24 |
| M4 daily | 7 |
| M4 weekly | 1 |
| M4 monthly | 6 |
| M4 quarterly | 4 |
| M4 yearly | 1 |

**Table 4.2:** The seasonal period used for each dataset. A seasonal component was not extracted for datasets with a seasonal period of 1.

To calculate the features of a time series, the time series was first decomposed. STL decomposition was used for all datasets except M4 yearly and M4 weekly. The full length time series in M4 weekly exhibit some seasonal patters, but due to the fact that the length of the lookback-window and horizon were less than the seasonal period of 52, it was omitted. M4 yearly displays no seasonal patterns at all. For these two datasets, the trend component was extracted using a moving average (Equation 2.8) with a order of 7. The length of the seasonal periods in each dataset can be seen in Table 4.2. Having decomposed the time series, the feature values could be calculated using the Equations 2.9, 2.10, 2.12 and 2.13. The feature $S_{str}$ was ignored in datasets lacking a seasonal component.

To create the instance space, 50 batches of data was sampled from the training data and the features of the sampled time series were calculated. These features were used to determine the two first PCA components, which then defined the instance space of the dataset. Figure 4.3 shows the test data of each dataset embedded in the instance space.

### 4.3.2 Data generation

The method for data generation was simple and relied on using a set of transformations on already existing time series to generate new ones. These transformations generate new trend components or seasonal components with the trend component being transformed with the following equation:

$$s = t \cdot \frac{m}{n} \cdot \theta_2 \tag{4.1}$$

$$\hat{T}_t = \theta_1 + f \cdot (m \cdot \theta_2 \cdot t + \frac{1}{h} \cdot \delta_t) + s \tag{4.2}$$

where $\theta_1$, $\theta_2$ and $\delta_t$ correspond to the intercept, slope and residuals of the linear regression model in Equation 2.11. $n$ is the length of the time series to be transformed and $t$ is a vector representing each time step. The factors $f$, $h$, and $m$ transform specific features of the trend component which correspond to $T_{str}$, $T_{slope}$ and $T_{lin}$ defined in Equations 2.9, 2.12 and 2.13. Note that the added slope to the time series $s$, is dependent on the time series length.

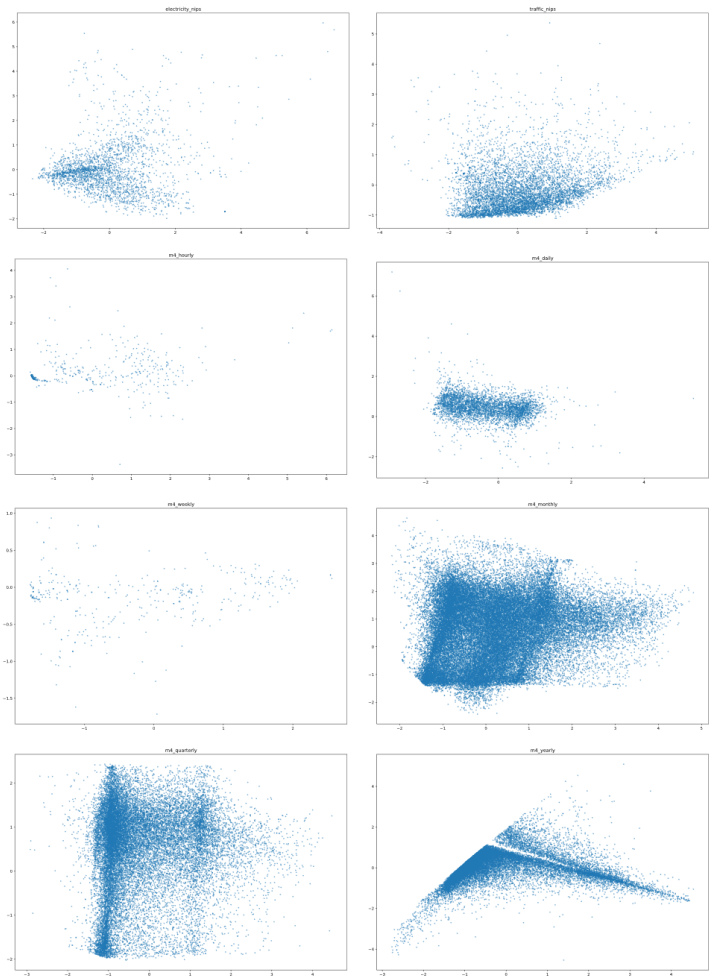



**Figure 4.3:** The test data of each dataset embedded in the instance space.

The seasonal component was transformed with the equation

$$\hat{S}_t = k \cdot S_t \tag{4.3}$$

where $k$ is a factor transforming the feature $S_{str}$ defined in Equation 2.10. The effects of different transformation factors can be seen in Figure 4.4.

## 4.4 Runtime environment

All experiments were run on the could computing platform Microsoft Azure using a virtual machine with Ubuntu 18.04.4 LTS as the operating system and the following hardware:

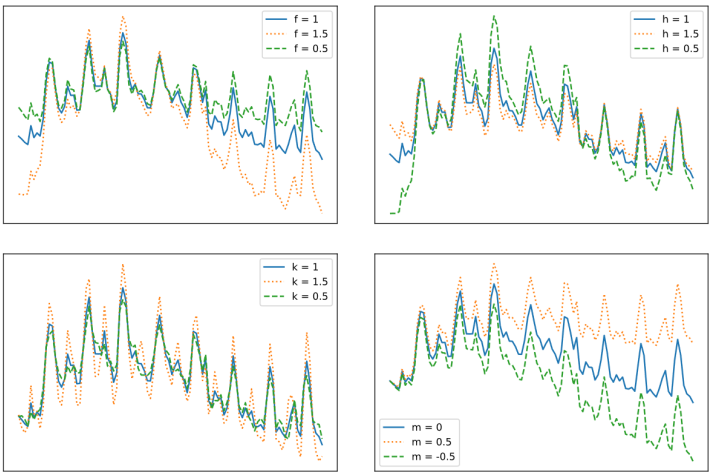



**Figure 4.4:** The effect of different transformation factors.

**GPU:** NVIDIA Tesla K80

**CPU:** Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz

**Number of Cores:** 6

**RAM:** 65 GiB

The models were implemented in Python 3.8 using the library Pytorch (Paszke et al., 2019). The datasets were downloaded using the library GluonTS (Alexandrov et al., 2020)[1].

## 4.5 Experiments

There where four experiments conducted in this thesis. Later in the thesis these will be referred to as **E1**, **E2**, **E3** and **E4**.

### 4.5.1 Experiment 1

The first experiment, **E1**, was designed to verify that the features and instance space described in subsection 4.3.1 could be used to identify out-of-distribution data. Because outliers in the instance space already represented unusual time series in terms of features, the experiment sought to visually confirm that such time series were unusual and different compared to in-distribution data from the same dataset. To do this, the test data was

[1]Note that the Electricity and Traffic datasets used here correspond to electricity_nips and traffic_nips in the library.

visualized using the instance space, with the instance space defined as described in subsection 4.3.1. Time series that could be identified as an outlier were visually inspected and compared to typical in-distribution data. The process was repeated for all datasets.

### 4.5.2 Experiment 2

The objective of the second experiment, **E2**, was both to investigate if the transformations defined in subsection 4.3.2 could be used to generate data at specific areas of the instance space, and to confirm that the generated out-of-distribution data was visually similar to existing out-of-distribution data.

To simplify the procedure, transformations only modified the value of a single transformation factor at a time and only values using some minimum or maximum value were tested. The different values of each factor can be seen in Table 4.3. With the time series of datasets often exhibiting different characteristics, the factors that were suitable to generate out-of-distribution data varied. For example, increasing the factor $k$ was unable to generate out-of-distribution data in the Electricity dataset as most time series already contained a strong seasonal component. Furthermore, certain transformation factors did not cause the generated data to be concentrated at one specific area of the instance space. Instead, the data was scattered, covering different areas containing time series exhibiting vastly different characteristics. Such transformations were also discarded as the objective of the experiment was to generate data at targeted areas of the instance space with specific characteristics and to visually compare generated and real data. Figure 4.5 shows the three transformations used to generate out-of-distribution data on the Electricity dataset, and one transformation found unsuitable for the experiment.

**Figure 4.5:** The left figure shows the three transformations used to generated out-of-distribution data for the Electricity dataset. The right figure shows a transformation found unsuitable for the experiment.

The two aforementioned details resulted in only some values of transformation factors being used to generate out-of-distribution data per dataset. Table 4.4 shows the values of different transformation factors applied to each dataset.

| | min | max |
|---|---|---|
| $f$ | 0.01 | 100 |
| $h$ | 0.01 | 100 |
| $k$ | 0.01 | 100 |
| $m$ | -1 | 1 |

**Table 4.3:** The two different values of each transformation factor used to generate data.

| **Dataset** | $f$ | $h$ | $k$ | $m$ |
|---|---|---|---|---|
| Electricity | - | - | 0.01 | -1, 1 |
| Traffic | 0.01 | - | 0.01 | -1, 1 |
| M4 hourly | - | - | 0.01 | -1, 1 |
| M4 daily | - | - | 0.01, 100 | -1, 1 |
| M4 weekly | 0.01 | - | - | -1, 1 |
| M4 monthly | 0.01, 100 | 0.01, 100 | 0.01, 100 | -1, 1 |
| M4 quarterly | 0.01, 100 | 0.01, 100 | 0.01, 100 | -1 |
| M4 yearly | - | 0.01 | - | -1, 1 |

**Table 4.4:** The transformation factors applied to each dataset to generate out-of-distribution data.

As with **E1**, the instance space already determined if time series were similar in terms of features. Thus, to confirm that the generated out-of-distribution data was similar to already existing out-of-distribution data, only visual comparisons were made. The time series generated with the transformations in Table 4.4 were compared to outliers in the original test data which were positioned similarly in the instance space. Having identified nearby outliers, the closest generated time series, based on the Euclidian distance in the instance space, was selected for a visual comparison.

### 4.5.3 Experiment 3

**E3**, attempted to use generated data to increase the robustness of neural networks applied to time series forecasting. The objective of the experiment was to investigate if the robustness of various models to out-of-distribution data at specific areas of the instance space could be improved by augmenting the training data. The experiment required training data to be augmented in such a way that sampled time series were positioned at a similar area as the generated out-of-distribution data. As in **E1**, due to differences in characteristics of the datasets the transformations applied to each dataset varied. Additionally, because training data was sampled from time series of longer lengths, finding transformations for the full length time series that yielded samples at the same area of the instance space as the generated test data proved to be difficult in some cases. The result were that some of the transformations in Table 4.4 were removed from the experiment. The transformations applied to each dataset can be seen in Table 4.5.

| Dataset | $f$ | $h$ | $k$ | $m$ |
|---|---|---|---|---|
| Electricity | - | - | 0.01 | -1, 1 |
| Traffic | 0.01 | - | - | 1 |
| M4 hourly | - | - | - | -1, 1 |
| M4 daily | - | - | 100 | - |
| M4 weekly | 0.01 | - | - | - |
| M4 monthly | 0.01 | 0.01 | - | 1 |
| M4 quarterly | - | 0.01 | - | -1 |
| M4 yearly | - | 0.01 | - | - |

**Table 4.5:** The transformation factors used in **E3**.

To augment the training data, each full length time series in the training data was transformed with the same transformation as listed in Table 4.4, with one exception. Because the new slope in Equation 4.2 is dependent on the time series length, using the same transformation factor for both the test data and the full length training time series resulted in large changes for former and small changes for the later. The issue was simply resolved by using $m = 10$ when for transformations of the full length training time series.

Having transformed the time series, they were then added to the original training data, doubling the size of the dataset. The transformations resulted in the sampled time series displaying similar characteristics as generated out-of-distribution data. All models were trained on both the original dataset and the augmented dataset, then tested on both generated out-of-distribution data and the original test data.

### 4.5.4 Experiment 4

**E4** investigated if models trained with augmented datasets would perform well on not just similar generated data, but similar real data. This experiment used the M4 monthly dataset, the largest and most diverse dataset used in this thesis, and removed a large part of the training data, creating a dataset with reduced size. The reduced size dataset was then used to do a similar experiment as in **E3**, but with models tested using real data.

To be able to do so, each full length time series from the training data was embedded in the instance space, and any time series with a first PCA component with a value larger than 2 was removed. The results were that sampled time series from the reduced training set did not cover the entire distribution of test data in the instance space. The reduced training set was then augmented, making it cover a larger area of the instance space. The dataset was augmented by transforming all time series with two factors, $k = 1.5$ and $m = 1.5$. These transformation factors were chosen manually by observing how the transformation factors affected the distribution of training data. Figure 4.6 shows the distribution of the reduced size training data before and after augmentation compared to the full test dataset. Models were trained on the reduced training set both with and without augmentation and the then compared.

When comparing the performance of the models, the test data was partitioned into two parts. The first part consisted of test data in the area where augmented training data was distributed, and one part consisted of the remaining time series. Specifically, the first part

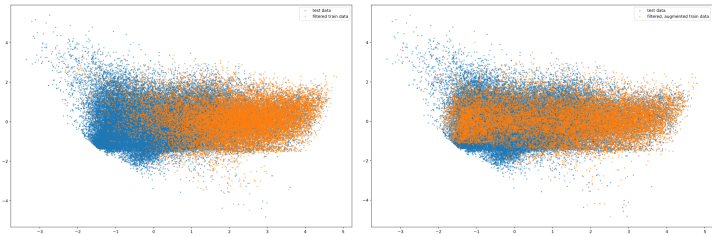



**Figure 4.6:** The reduced training data of M4 monthly compared to the full test dataset. The left figure shows time series sampled from the reduced training set compared to all test data in M4 monthly. The right figure shows time series sampled from the reduced training after augmentation.

was defined as test data where the first PCA component had a value less than 0, and the second PCA component had a value less than 2. The first part was in this experiment defined as the out-of-distribution data test data, and the second part as in-distribution test data.

Chapter 5

# Results

This chapter presents the results of the experiments.

## 5.1 Experiment 1

Examples of visual comparisons made between outliers and inliers in the Electricity dataset during **E1** can be seen in Figure 5.1. Appendix A contains similar figures for the other datasets. In the case of the Electricity dataset, the figure show that the inliers in the dataset consists of a strong seasonal pattern without any significant trend. The visualized outliers in this dataset appear to exhibit a combination of more noise, a larger trend component and a smaller seasonal component.

In general, the visual inspection showed that the feature space was able to separate in-distribution and out-of-distribution data and that time series with similar characteristics were placed close in the instance space. The datasets M4 daily, weekly and yearly had instance spaces were outliers were hard to visually separate from inliers. Otherwise two notable exceptions were found: time series displaying sudden drops or jumps and time series with a seasonal period different from the ones defined in Table 4.2. Time series with sudden jumps or drops in value were often placed next to time series with strong slopes, while time series with a different seasonal period were positioned next to time series with displaying much smaller amounts of seasonality.

## 5.2 Experiment 2

For **E2**, a visual comparisons between generated time series and outliers can be seen in Figure 5.2. The generated data usually resembles already existing test data at similar positions in the instance space, but exceptions are encountered for similar cases as in **E1**.

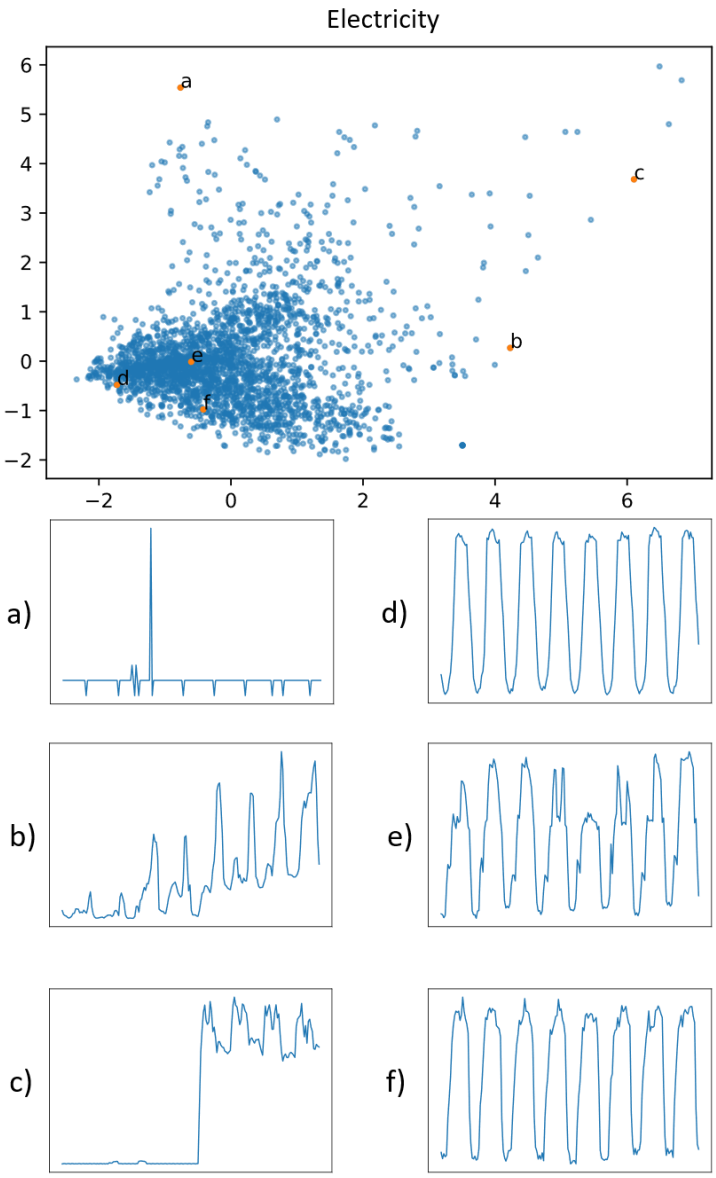



**Figure 5.1:** Outliers and inliers in the Electricity dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.

## 5.3 Experiment 3

The results of **E3** can be seen in Table 5.1 and are given in terms of MAE (Equation 2.2). The average percentage change of each augmentation scheme can be seen in Table 5.2


**Figure 5.2:** Outlier time series compared with transformed time series close in instance space. Each column consists of either ”target” time series, time series already present in the dataset, or ”transformed” time series, time series that have been transformed to be as close as possible to the corresponding target in instance space. Transformed time series in column 2 are close in instance space with targets in column 1, and the other columns follow a similar pattern.

and the average percentage change per model can be seen in Table 5.3. On average, the augmented models had a $12.755\%$ increase in MAE on the original test data and a $40.143\%$ decrease in MAE on the generated test data.

| | | Fully-connected | | LSTM | | N-BEATS | | TCN | | Transformer | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | old | augmented | old | augmented | old | augmented | old | augmented | old | augmented |
| Electricity ($k = 0.01$) | original test set | 58.41 | 63.663 (+8.994%) | 53.361 | 59.56 (+11.618%) | 44.002 | 44.284 (+0.642%) | 53.067 | 74.053 (+39.547%) | 51.421 | 53.983 (+4.982%) |
| | generated test set | 39.696 | 30.907 (-22.141%) | 43.864 | 42.126 (-3.962%) | 38.066 | 25.911 (-31.933%) | 39.43 | 35.993 (-8.715%) | 40.159 | 32.635 (-18.736%) |
| Electricity ($m = -1$) | original test set | 58.41 | 48.889 (-16.299%) | 53.361 | 56.733 (+6.32%) | 44.002 | 44.581 (+1.316%) | 53.067 | 52.674 (-0.74%) | 51.421 | 50.602 (-1.593%) |
| | generated test set | 315.842 | 52.343 (-83.427%) | 309.964 | 174.621 (-43.664%) | 111.604 | 55.5 (-50.27%) | 244.093 | 72.32 (-70.372%) | 244.993 | 62.651 (-74.428%) |
| Electricity ($m = 1$) | original test set | 58.41 | 49.359 (-15.495%) | 53.361 | 61.706 (+15.64%) | 44.002 | 44.079 (+0.176%) | 53.067 | 55.908 (+5.354%) | 51.421 | 54.598 (+6.178%) |
| | generated test set | 246.497 | 53.815 (-78.168%) | 335.354 | 147.311 (-56.073%) | 162.909 | 60.974 (-62.572%) | 203.868 | 75.164 (-63.131%) | 201.305 | 82.859 (-58.839%) |
| Traffic ($m = 1$) | original test set | 0.007 | 0.008 (+5.663%) | 0.009 | 0.009 (+7.231%) | 0.005 | 0.005 (+2.507%) | 0.008 | 0.01 (+35.766%) | 0.009 | 0.028 (+205.099%) |
| | generated test set | 0.015 | 0.008 (-46.047%) | 0.027 | 0.023 (-16.634%) | 0.014 | 0.007 (-48.256%) | 0.031 | 0.017 (-47.355%) | 0.04 | 0.031 (-23.248%) |
| Traffic ($f = 0.01$) | original test set | 0.007 | 0.008 (+0.394%) | 0.009 | 0.008 (-4.143%) | 0.005 | 0.006 (+3.841%) | 0.008 | 0.012 (+61.554%) | 0.009 | 0.026 (+191.278%) |
| | generated test set | 0.009 | 0.009 (-0.429%) | 0.024 | 0.019 (-22.352%) | 0.008 | 0.008 (-0.766%) | 0.014 | 0.013 (-2.98%) | 0.031 | 0.026 (-16.51%) |
| M4 hourly ($m = -1$) | original test set | 285.752 | 332.324 (+16.298%) | 530.073 | 359.344 (-32.209%) | 387.313 | 300.203 (-22.491%) | 310.46 | 577.916 (+86.148%) | 363.743 | 438.258 (+20.485%) |
| | generated test set | 4044.765 | 629.797 (-84.429%) | 4095.786 | 718.622 (-82.455%) | 4977.778 | 657.757 (-86.786%) | 4442.087 | 840.151 (-81.087%) | 4435.814 | 693.364 (-84.364%) |
| M4 hourly ($m = 1$) | original test set | 285.752 | 504.937 (+76.705%) | 530.073 | 316.841 (-40.227%) | 387.313 | 373.598 (-3.541%) | 310.46 | 396.511 (+27.717%) | 363.743 | 352.61 (-3.061%) |
| | generated test set | 3156.815 | 686.334 (-78.259%) | 3579.044 | 585.751 (-83.634%) | 3587.283 | 439.313 (-87.754%) | 4792.464 | 2306.926 (-51.863%) | 3939.253 | 988.161 (-74.915%) |
| M4 daily ($k = 100$) | original test set | 180.823 | 184.145 (+1.837%) | 180.538 | 179.869 (-0.371%) | 176.184 | 176.107 (-0.044%) | 186.772 | 179.482 (-3.903%) | 180.101 | 178.252 (-1.027%) |
| | generated test set | 5097.152 | 2459.157 (-51.754%) | 4832.058 | 2282.26 (-52.768%) | 4838.892 | 2314.794 (-52.163%) | 4878.213 | 2466.513 (-49.438%) | 4787.081 | 2723.599 (-43.105%) |
| M4 weekly ($f = 0.01$) | original test set | 305.795 | 317.963 (+3.979%) | 330.376 | 342.834 (+3.771%) | 342.279 | 356.011 (+4.012%) | 343.894 | 371.897 (+8.143%) | 330.943 | 349.955 (+5.745%) |
| | generated test set | 1370.924 | 545.287 (-60.225%) | 1072.014 | 521.377 (-51.365%) | 236.57 | 162.882 (-31.149%) | 992.923 | 712.984 (-28.193%) | 214.704 | 181.355 (-15.532%) |
| M4 monthly ($m = 1$) | original test set | 599.508 | 614.42 (+2.487%) | 578.395 | 598.135 (+3.413%) | 543.125 | 557.961 (+2.731%) | 607.586 | 681.44 (+12.155%) | 593.16 | 630.708 (+6.33%) |
| | generated test set | 730.01 | 614.547 (-15.817%) | 686.014 | 600.295 (-12.495%) | 856.948 | 580.417 (-32.269%) | 773.443 | 659.565 (-14.724%) | 802.441 | 630.376 (-21.443%) |
| M4 monthly ($h = 0.01$) | original test set | 599.508 | 675.136 (+12.615%) | 578.395 | 610.934 (+5.626%) | 543.125 | 553.251 (+1.864%) | 607.586 | 980.701 (+61.409%) | 593.16 | 662.76 (+11.734%) |
| | generated test set | 28597.189 | 17923.248 (-37.325%) | 25665.182 | 14539.581 (-43.349%) | 24330.667 | 15835.171 (-34.917%) | 29048.515 | 17295.919 (-40.459%) | 28854.079 | 22039.445 (-23.618%) |
| M4 monthly ($f = 0.01$) | original test set | 599.508 | 583.48 (-2.673%) | 578.395 | 577.47 (-0.16%) | 543.125 | 543.73 (+0.111%) | 607.586 | 612.072 (+0.738%) | 593.16 | 583.062 (-1.702%) |
| | generated test set | 379.357 | 304.517 (-19.728%) | 369.606 | 300.902 (-18.589%) | 332.974 | 293.062 (-11.987%) | 395.106 | 321.762 (-18.563%) | 390.73 | 315.97 (-19.133%) |
| M4 quarterly ($h = 0.01$) | original test set | 601.418 | 662.194 (+10.106%) | 593.796 | 645.512 (+8.709%) | 549.069 | 562.453 (+2.438%) | 586.797 | 659.522 (+12.394%) | 619.836 | 693.36 (+11.862%) |
| | generated test set | 37976.754 | 27645.278 (-27.205%) | 33722.531 | 24824.17 (-26.387%) | 33184.391 | 27129.669 (-18.246%) | 34644.927 | 21912.904 (-36.75%) | 37737.281 | 31564.946 (-16.356%) |
| M4 quarterly ($m = -1$) | original test set | 601.418 | 612.979 (+1.922%) | 593.796 | 610.975 (+2.893%) | 549.069 | 586.319 (+6.784%) | 586.797 | 618.783 (+5.451%) | 619.836 | 693.07 (+11.815%) |
| | generated test set | 1365.815 | 677.806 (-50.374%) | 1296.877 | 656.002 (-49.417%) | 1496.897 | 637.207 (-57.431%) | 1556.312 | 707.03 (-54.57%) | 1356.687 | 740.236 (-45.438%) |
| M4 yearly ($h = 0.01$) | original test set | 856.154 | 941.741 (+9.997%) | 909.995 | 1082.577 (+18.965%) | 836.153 | 860.36 (+2.895%) | 882.914 | 969.431 (+9.799%) | 888.936 | 890.305 (+0.154%) |
| | generated test set | 24146.394 | 20045.627 (-16.983%) | 26061.999 | 26058.511 (-0.013%) | 22951.076 | 14019.047 (-38.918%) | 24956.773 | 22856.016 (-8.418%) | 34480.49 | 28419.593 (-17.578%) |

**Table 5.1:** The results of the old and new models on the original and generated test sets in terms of MAE. Both the generated test data and the augmented training data where generated by applying the transformation in the parenthesis behind the dataset name.

| | Original test set | Generated test set |
|---|---|---|
| Electricity ($k = 0.01$) | $13.157 \pm 13.708$ | $-17.097 \pm 9.91$ |
| Electricity ($m = -1$) | $-2.199 \pm 7.566$ | $-64.432 \pm 15.019$ |
| Electricity ($m = 1$) | $2.371 \pm 10.232$ | $-63.757 \pm 7.651$ |
| Traffic ($m = 1$) | $51.253 \pm 77.847$ | $-36.308 \pm 13.544$ |
| Traffic ($f = 0.01$) | $50.585 \pm 74.316$ | $-8.607 \pm 9.071$ |
| M4 hourly ($m = -1$) | $13.646 \pm 41.759$ | $-83.824 \pm 1.939$ |
| M4 hourly ($m = 1$) | $11.519 \pm 39.06$ | $-75.285 \pm 12.512$ |
| M4 daily ($k = 100$) | $-0.702 \pm 1.862$ | $-49.846 \pm 3.554$ |
| M4 weekly ($f = 0.01$) | $5.130 \pm 1.666$ | $-37.293 \pm 16.236$ |
| M4 monthly ($m = 1$) | $5.423 \pm 3.635$ | $-19.350 \pm 7.102$ |
| M4 monthly ($h = 0.01$) | $18.650 \pm 21.744$ | $-35.933 \pm 6.784$ |
| M4 monthly ($f = 0.01$) | $-0.737 \pm 1.258$ | $-17.600 \pm 2.839$ |
| M4 quarterly ($h = 0.01$) | $9.102 \pm 3.578$ | $-24.989 \pm 7.283$ |
| M4 quarterly ($m = -1$) | $5.773 \pm 3.486$ | $-51.446 \pm 4.170$ |
| M4 yearly ($h = 0.01$) | $8.362 \pm 6.549$ | $-16.382 \pm 12.969$ |

**Table 5.2:** The average percentage change $\pm$ on standard deviation in MAE for models trained on augmented data, both on the original and generated test data for each augmentation scheme. Negative numbers signify a reduction in average MAE.

| | Original test set | Generated test set |
|---|---|---|
| Fully-connected | $7.769 \pm 20.429$ | $-44.821 \pm 26.714$ |
| LSTM | $0.4728 \pm 15.584$ | $-37.544 \pm 25.08$ |
| N-BEATS | $0.216 \pm 6.472$ | $-43.028 \pm 23.797$ |
| TCN | $24.102 \pm 26.354$ | $-38.441 \pm 23.382$ |
| Transformer | $31.219 \pm 65.84$ | $-36.883 \pm 23.964$ |

**Table 5.3:** The average percentage change $\pm$ on standard deviation in MAE for models trained on augmented data, both on the original and generated test data per model. Negative numbers signify a reduction in average MAE.

## 5.4 Experiment 4

The results of **E4** can be seen in Table 5.4 and is given in terms of MAE.

| | | In-distribution | Out-of-distribution | Full test set |
|---|---|---|---|---|
| Fully-connected | Original model | 713.781 | 528.009 | 656.486 |
| | Augmented model | 776.41 (+8.774%) | 456.101 (-13.619%) | 650.551 (-0.904%) |
| N-BEATS | Original model | 662.597 | 490.612 | 615.116 |
| | Augmented model | 703.695 (+6.203%) | 425.279 (-13.317%) | 609.246 (-0.954%) |
| LSTM | Original model | 718.874 | 535.679 | 665.311 |
| | Augmented model | 771.928 (+7.38%) | 469.092 (-12.43%) | 656.704 (-1.294%) |
| TCN | Original model | 693.521 | 543.963 | 653.459 |
| | Augmented model | 432.441 (+5.211%) | 729.662 (-20.502%) | 615.377 (-5.828%) |
| Transformer | Original model | 773.311 | 650.618 | 754.088 |
| | Augmented model | 760.909 (-1.604%) | 468.098 (-28.053%) | 648.496 (-14.003%) |

**Table 5.4:** The average MAE for models trained on the reduced M4 dataset. The percentage increase or decrease compared to the original model is shown in the parenthesis.

Chapter 6

# Evaluation

This chapter starts off by evaluating each research question based on the results presented in the previous chapter, discussing each research question in turn. The results are then evaluated in light of the related work presented in chapter 3, before the chapter discusses the contributions listed in section 1.5. Finally, the chapter evaluates the hypothesis and objective of the thesis.

## 6.1 Evaluation of Research Questions

The first research question in this thesis was concerned with how out-of-distribution data could be generated and was answered through a review of the state of the art in time series augmentation methods, **E1** and **E2**. Specifically, it was formulated as:

**RQ1:** Which methods are suitable for generating out-of-distribution data for time series forecasting?

The review uncovered many different possible methods, with some already having been applied in Bandara et al. (2020). In this work, the approach of Kegel et al. (2017) was chosen primarily due to its simplicity while it is still able to generate time series with specific characteristics and possibly generating time series with similar temporal dependencies. To generate out-of-distribution data with the method, it was necessary to understand the characteristics of in-distribution data. By utilizing the idea of an instance space from Kang et al. (2017), time series could be summarized with a set of features and visualized in a two dimensional space. The instance space was then identify the characteristics of in- and out-of-distribution data.

Given that the instance space already visualized time series that were statistically dissimilar far away from in-distribution data, **E1** sought to confirm that such time series were different visually as well. The results showed that out-of-distribution data were different than in-distribution time series, with the time series in M4 weekly and yearly being datasets were it was harder to visually separate inliers from outliers. These datasets did

not contain any seasonality, and hence $S_{str}$ was ignored. Lack of separation between inliers and outliers in these datasets could be related to time series only being described through three features, and it also points to remaining features being unsuited for these datasets in general.

There were also exceptions related to specific characteristics of time series, the prime example being found in the M4 hourly dataset and can be seen in Appendix A. M4 hourly contains several periodic time series, but the length of the period is not necessarily the same as the seasonality used when decomposing the time series (see Table 4.2). The mismatch between period and seasonality caused the seasonal component of such time series to be small, and the value of $S_{str}$ (Equation 2.10) to be small. It is possible to argue that this is expected behavior, however it caused the instance space to be unable to separate time series consisting of mostly noise from simple periodic time series. Another exception was encountered for time series displaying sudden jumps or drops in value. These time series also had trend components with strong slopes, leading the instance space to place time series with these two characteristics together.

This experiment uncovered limitations in the features used, as they are unable to represent such sudden jumps or drops of value in a time series. These two exceptions, added with the difficulties encountered for M4 weekly and yearly, points towards the feature set being unfit in some cases, and that it should be adapted on a dataset by dataset basis. Still, the feature space could, in most cases, be used to separate inliers from outliers.

**E2** investigated if generated out-of-distribution data was visually similar outliers already present in the dataset. The experiment showed that the data generating method created time series that were visually similar to time series close in the instance space with exceptions again being time series displaying sudden jumps or drops and periodic time series with a period different that the seasonality. This limitation stems from the simplicity of the data generating method used in this thesis, and the method is simply not able to generate such series. The bottom left comparison of Figure 5.2 shows an example of a periodic time series and an aperiodic time series that were placed close in the instance space of M4 hourly.

Looking at the augmentation schemes used **E3**, another possible limitation of the data generating method appears. Table 4.5 shows that increasing one of the transformation factors to generate out-of-distribution data and augment the training data was only possible in one case (M4 daily $k = 100$). The source of the apparent difficulty in using increasing values of $f$, $h$ and $k$ to create both out-of-distribution test data and training data could stem from both a limitation of the transformations themselves, or the transformation factors not being able to sufficiently change local properties of the full length time series in the training data. The later of these possibilities could easily would easily be solved by augmenting the training data with shorter time series transformed to exhibit the desired characteristics.

While the results related to **RQ1** showed that the chosen method for generating out-of-distribution data had its limitations, it was able to produce out-of-distribution data. Combined with section 3.2 uncovering several other suitable methods for generation of out-of-distribution data, the research question can be regarded as answered.

The second research question of the thesis was formulated as:

**RQ2:** How can neural networks be made more robust to out-of-distribution data in the

context of time series forecasting?

The question was answered through **E3** and **E3**, with the results of **E4** being shown in Table 5.1, 5.2 and 5.3. The experiment showed that on average models trained with augmented data improved the performance on generated test data by $40.143\%$, but at the same time performance on the original test data decreased by $12.755\%$. The results also show that the performance of the original models in most cases degenerated when applied to out-of-distribution data. In cases were original model performance did not degenerate on out-of-distribution data, the models trained with augmented data still performed better.

When looking at the individual augmentation schemes applied to each dataset in Table 5.2, it can be seen that augmenting datasets with out-of-distribution data increases the robustness to similar data. However, the effects on performance on the original test set display a large amount of variance. Electricity $(m = -1)$ reduced the average MAE of models with 2.199% but on the other hand, Traffic $(m = 1)$ increased the average MAE by 51.253%. The table also shows that the standard deviation is large for some of the augmentation schemes, and especially Traffic $(m = 1)$ and $(f = 0.01)$, M4 hourly $(m = -1)$ and $(m = 1)$, and M4 monthly $(h = 0.01)$. Table 5.1 shows that for Traffic $(m = 1)$ and $(f = 0.01)$, the large standard deviation mostly comes from the fact that the transformer's performance is reduced by around 200% with the TCN also experiencing a large reduction. The large standard deviation in M4 monthly $(h = 0.01)$ mostly stems from the TCN, with the other models exhibiting a more moderate change in performance. The M4 hourly $(m = -1)$ and $(m = 1)$ schemes have a much smaller average performance reduction than Traffic $(m = 1)$, $(f = 0.01)$ and M4 monthly $(h = 0.01)$, but still display a large standard deviation. For these to schemes, the models are affected in very different ways. In the case of M4 hourly $(m = -1)$, the performance of the TCN and transformer again deteriorate significantly with the performance of the Fully-connected model also dropping. N-BEATS and the LSTM models do in contrast show a large increase in performance. For M4 hourly $(m = 1)$, the TCN and Fully-connected models again have a large decrease in performance and the LSTM again having a large increase. The transformer and N-BEATS do in this case only experience a smaller change in performance.

Table 5.3 shows the average percentage change per augmented model compared to the original models, and it shows that all models experience gains in robustness. Similarly to Table 5.2, there are fairly big differences between the models when comparing performance on the original test data. This suggests that not all models are equally able to represent both the original in-distribution data and out-of-distribution data. The augmented LSTM and N-BEATS models has on average a small decrease in performance on the original test data, but it should be noted that N-BEATS has a much smaller standard deviation. While the N-BEATS and LSTM models appear to be able to retain similar performance on the original test data when retrained, the TCN and transformer suffer a significant reduction in performance. While the reduction in performance is large for these models, it is also heavily skewed by the large reductions seen in Traffic $(m = 1)$ and $(f = 0.01)$, M4 hourly $(m = -1)$ and $(m = 1)$, and M4 monthly $(h = 0.01)$. The Fully-connected model also suffer a reduction in performance on average, but to a lesser extent than the TCN and transformer.

These results make it seem like there might be some intrinsic properties of the LSTM and N-BEATS models that make them retain the performance on the original dataset to a

larger degree. While an interesting possibility, this pattern is more likely to be the result of the number of parameters in each model, with N-BEATS having by far the highest and the LSTM having the second highest number of parameters. Interestingly, TCN has the smallest number of parameters, yet still retain performance to a higher degree than the transformer. Taking a deeper look into the underlying causes of the differences seen between the models could be an interesting area of future research.

The last experiment, **E4**, was designed to test if the augmentation method would increase model robustness in a real setting. The M4 dataset was used in this experiment, and the training data was reduced to time series only containing a small amount of characteristics compared to the original dataset. By augmenting the training data, the results in Table 5.4 shows that the model robustness increased to out-of-distribution data. Both the increase in performance on out-of-distribution data and decrease in performance on in-distribution data was smaller than what was seen in Table 5.3. The TCN and transformer models that experienced the larges decrease in performance on out-of-distribution data in **E3** here had the smallest decrease in performance. However, these models also had the largest variance in terms of results (Table 5.3), and more similar experiments would have to be conducted before reaching any definitive conclusions related to this observation. The results of **E4** can be summarized as confirming that models trained on datasets augmented with generated out-of-distribution data would gain an increased robustness when faced with real out-of-distribution data.

In general, the results empirically show the efficacy of augmenting datasets with out-of-distribution data to boost the robustness of a model to time series with specific characteristics. Combining the data generating method with an instance space allowed outliers in the dataset to be identified and out-of-distribution data with similar characteristics to be generated. Therefore the combination of the instance space and method for data generation allows for identification of data that are plausible, but not well represented in the dataset, and in turn augmenting the dataset with such data makes the model more robust with a cost of reduced performance on the original data. A drawback of the proposed method is the amount of manual labor required to find transformations that generate time series at specific areas of the feature space. This limitation was discussed in subsection 4.5.1, but it should be noted that the need for the generated test and training data to overlap in the instance space further complicates the issue. Additionally, with the decline in performance on the original datasets in some cases being substantial, the trade off between performance and robustness needs to be considered before applying the method.

From the results it is evident that augmenting the training data with out-of-distribution data increases the robustness of neural networks to out-of-distribution data. The result have been demonstrated and repeated across several datasets and models for out-of-distribution data with varying characteristics and were eventually tested on a real dataset. Thus, **RQ2** can be regarded as answered.

## 6.2 Evaluation in Light of Related Work

Compared to related work concerning the generation of time series data, the method used here is simple and fast, and in combination with instance spaces it can be used to generated new data with targeted characteristics. The ability to transform specific features of a time

series separates it from the simplest approaches to data generation, like those proposed in Um et al. (2017); Cui et al. (2016); Guennec et al. (2016). In combination with an instance space, the transformations enables generating out-of-distribution data with targeted characteristics, something the simplest approaches are unsuited for. The main drawback of the method is the amount of manual work required to find transformation factors that transform a time series to the target area. This issue was largely ignored in this work, and instead crude transformations moving data to some extreme was chosen to demonstrate the approach. The work in (Kang et al., 2017, 2020) is more flexible in that regard, allowing time series to be generated at specific areas of the instance space but the increased flexibility comes with a higher computational cost.

Some methods for generating time series data presented in section 3.2 do not ensure the validity of the temporal dynamics of a generated time series. Simple techniques like rotating and permuting time series are examples of such methods, but more advanced methods also suffer from the same shortcomings (Kang et al., 2017; Esteban et al., 2017; Mogren, 2016; Ramponi et al., 2018; Chen et al., 2018). Other methods are more likely to produce time series with valid temporal dynamics because the generated time series are either simulated from a temporal process (Kang et al., 2020; Yoon et al., 2019) or generated using an already existing time series as a prototype (Kegel et al., 2017, 2018; Bergmeir et al., 2016; Iftikhar et al., 2017). The approach of Kegel et al. (2017), which is used in this thesis, falls in the later of these two categories because already existing time series are used as a prototype to generate new ones. By decomposing the prototype and transforming its individual components, the generative method assures that new time series have temporal dynamics at least partly similar to the prototype. The method employed here is simpler than the methods like Kang et al. (2017, 2020); Yoon et al. (2019), while still facilitating the generation of out-of-distribution data with specific characteristics, something that methods like Bergmeir et al. (2016); Iftikhar et al. (2017) are unable to do.

The work presented in section 3.1 reviled that most work concerning out-of-distribution data in neural networks mostly focused on robustness in classification tasks. In these tasks, improving a models robustness can be reduced to detecting out-of-distribution data as such samples do not belong in any of the classes of the in-distribution data. The consequence is that approaches based on classifying time series as in- or out-of-distribution provide an unsatisfactory solution in the forecasting setting, as this would discard time series without providing a forecast. For probabilistic forecasting models it would be possible to investigate how the uncertainty of the forecast can be increased when applied to out-of-distribution data. That would be analogous to the research in Hendrycks and Gimpel (2017); Liang et al. (2018); Lee et al. (2018); Hendrycks et al. (2019), which attempts to increase the uncertainty of a classifier when faced with out-of-distribution data.

Instead of increasing the uncertainty of a forecast, this thesis focused on improving the forecasts on subsets of out-of-distribution data. As the instance spaces of datasets show, outliers tend to be spread in certain directions away from the main distribution of the data. These directions can be thought of as signifying areas of the instance space where data could possibly be generated, but where the dataset lacks coverage. This makes it possible to identify time series which one could expect to encounter in the future, and the model should hence be expected to provide satisfactory forecasts for. Identifying plausible types of out-of-distribution data is beneficial because it avoids making the model robust to

time series with characteristics which are not realistically encountered in a domain. For example, given a time series monitoring the number of cars passing a sign per hour over multiple days, the number of cars passing being strictly increasing is highly unrealistic and it is not necessary for a model to be able to accurately predict such time series.

Another benefit of restricting the characteristics of out-of-distribution data is a simplified data generation procedure. In the approaches to improving model robustness which require out-of-distribution data, effort has to be spent on either training a generative model to be able to produce such data (Lee et al., 2018) or to provide out-of-distribution data through auxiliary datasets (Hendrycks et al., 2019). Having a highly diverse set of out-of-distribution data is key in such methods since models have to be robust to all types of out-of-distribution data in a classification setting. Reducing the number of different characteristics of out-of-distribution data need to contain greatly simplifies the acquisition out-of-distribution data.

## 6.3 Contributions

This section evaluates the contributions of the thesis. The first contribution was formulated as:

**C1:** *Demonstrate an approach to generate out-of-distribution data for time series forecasting.*

Research studying the robustness of neural networks rely on access to out-of-distribution data. Such data can either be generated or provided by some auxiliary dataset. This thesis has linked research concerning methods for generating time series with specific characteristics with research concerning robustness of neural networks neural networks. The approach demonstrated here showed how a simple method for data generation could be used to generate time series with specific characteristics. Being able to generate specific characteristics was crucial, as it made it possible to generate time series different from those already present in a dataset. The controlled manner in which out-of-distribution data is generated enables models to be tested in highly specific ways, having the potential to reveal interesting properties of the models under consideration.

**C2:** *Demonstrate an approach to improve the robustness of neural networks for time series forecasting when applied to out-of-distribution data.*

The demonstrated data generating method was in this thesis applied as an augmentation strategy to improve the robustness of neural networks. By augmenting a dataset with time series displaying a given characteristic, the models robustness to such data was improved. The argument behind such a method is the observation that certain datasets are more liable to produce outliers with certain characteristics. Improving a model's robustness to such outliers critical, while time series with characteristics differing from both the in-distribution data and the outliers do not contribute to model robustness in any meaningful way.

## 6.4 Evaluation of Objective and Hypothesis

The underlying hypthesis of the thesis was defined as:

**HYP:** *Neural networks applied to time series forecasting can be made more robust by augmenting the dataset with out-of-distribution data.*

With the objective of the thesis being directly related to the hypothesis:

**O:** *Find out how out-of-distribution time series data can be generated and how it can be leveraged to increase the robustness of neural networks applied to time series forecasting.*

To reach the objective, the two research questions **RQ1** and **RQ2** where posed and answered. Each of the two research questions were answered through the contributions **C1** and **C2** respectively. With the contributions answering each of the two research questions **RQ1** and **RQ2**, the objective **O** of the thesis had been reached which also confirmed the hypothesis **HYP**.

Chapter 7

# Conclusion and Future Work

This chapter concludes the thesis and lists possible directions for future work.

## 7.1 Conclusion

This thesis documented research done to improve the robustness of neural networks when applied to time series forecasting. A simple method for generating out-of-distribution data was first demonstrated, then leveraged to augment several different datasets for time series forecasting. The data generating method was successfully able to generate out-of-distribution data for all datasets, but due to its simplicity generating some characteristics of time series proved to be difficult. Multiple models were then trained on both the original and augmented datasets, and comparisons between performance on in- and out-of-distribution data showed an increase in performance on out-of-distribution data for the models trained with augmented data. The increase in performance on out-of-distribution data came with a cost of reduced performance on in-distribution data, with certain models experiencing a larger drop in performance than others.

In conclusion, the thesis has shown that the robustness of neural networks can be increased by augmenting datasets with out-of-distribution data, thus confirming the hypothesis of the thesis.

## 7.2 Future work

This section lists topics for future research:

**The data generating method:** The data generating method demonstrated in this thesis is simple and future research could attempt to extend it to alleviate the weaknesses uncovered in section 6.1. Another possible direction is to use one of the more advanced methods discussed in section 3.2. An improved method for generating data

could generate time series with a larger variation in terms of characteristics, facilitating more research into the robustness of neural networks applied to time series forecasting.

**Augmenting with multiple characteristics:** The experiments done in this thesis only augments datasets to improve the robustness in regard to one specific characteristic. Augmenting datasets with multiple characteristics could provide further insight into how model robustness can be improved.

**Uncovering model properties:** Being able to generate data with specific characteristics enables extensive testing of models under various conditions. Such testing could uncover inherent strengths and weaknesses of different models which are currently poorly understood due to the opaque nature of neural networks.

**Performance trade-off:** Augmenting the datasets with out-of-distribution data caused both an increase in model robustness and a reduction in performance on in-distribution data. The cause of the interaction has not been properly addressed in the research made here, but uncovering the true cause of the phenomenon might provide an improved understanding of model robustness.

# Bibliography


Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D.C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A.C., Wang, Y., 2020. GluonTS: Probabilistic and Neural Time Series Modeling in Python. Journal of Machine Learning Research 21, 1–6. URL: http://jmlr.org/papers/v21/19-820.html.

Ba, J.L., Kiros, J.R., Hinton, G.E., 2016. Layer Normalization URL: http://arxiv.org/abs/1607.06450, arXiv:1607.06450.

Bandara, K., Hewamalage, H., Liu, Y.H., Kang, Y., Bergmeir, C., 2020. Improving the Accuracy of Global Forecasting Models using Time Series Data Augmentation URL: http://arxiv.org/abs/2008.02663, arXiv:2008.02663.

Bergmeir, C., Hyndman, R.J., Benítez, J.M., 2016. Bagging exponential smoothing methods using STL decomposition and Box-Cox transformation. International Journal of Forecasting 32, 303–312. doi:10.1016/j.ijforecast.2015.07.002.

Borovykh, A., Bohte, S., Oosterlee, C.W., 2017. Conditional time series forecasting with convolutional neural networks, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag. pp. 729–730. arXiv:1703.04691.

Chen, Y., Li, P., Zhang, B., 2018. Bayesian renewables scenario generation via deep generative networks, in: 2018 52nd Annual Conference on Information Sciences and Systems, CISS 2018, Institute of Electrical and Electronics Engineers Inc.. pp. 1–6. doi:10.1109/CISS.2018.8362314, arXiv:1802.00868.

Chiu, C.C., Sainath, T.N., Wu, Y., Prabhavalkar, R., Nguyen, P., Chen, Z., Kannan, A., Weiss, R.J., Rao, K., Gonina, E., Jaitly, N., Li, B., Chorowski, J., Bacchiani, M., 2018. State-of-the-Art Speech Recognition with Sequence-to-Sequence Models, in: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, Institute of Electrical and Electronics Engineers Inc.. pp. 4774–4778. doi:10.1109/ICASSP.2018.8462105, arXiv:1712.01769.

Choi, H., Jang, E., Alemi, A.A., 2018. WAIC, but Why? Generative Ensembles for Robust Anomaly Detection. arXiv URL: http://arxiv.org/abs/1810.01392, arXiv:1810.01392.

Cireşan, D.C., Meier, U., Gambardella, L.M., Schmidhuber, J., 2010. Deep, big, simple neural nets for handwritten digit recognition. doi:10.1162/NECO_a_00052.

Cui, X., Goel, V., Kingsbury, B., 2015. Data Augmentation for Deep Neural Network Acoustic Modeling. IEEE Transactions on Audio, Speech and Language Processing 23, 1469–1477. doi:10.1109/TASLP.2015.2438544.

Cui, Z., Chen, W., Chen, Y., 2016. Multi-Scale Convolutional Neural Networks for Time Series Classification. Multi-scale Convolutional Neural Networks for Time Series Classification URL: http://arxiv.org/abs/1603.06995, arXiv:1603.06995.

DeVries, T., Taylor, G.W., 2018. Learning Confidence for Out-of-Distribution Detection in Neural Networks. arXiv URL: http://arxiv.org/abs/1802.04865, arXiv:1802.04865.

DeVries, T., Taylor, G.W., 2019. Dataset augmentation in feature space, in: 5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings, International Conference on Learning Representations, ICLR. URL: https://arxiv.org/abs/1702.05538v1, arXiv:1702.05538.

Esteban, C., Hyland, S.L., Rätsch, G., 2017. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs URL: http://arxiv.org/abs/1706.02633, arXiv:1706.02633.

Fawaz, H.I., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A., 2018. Data augmentation using synthetic data for time series classification with deep residual networks URL: http://arxiv.org/abs/1808.02455, arXiv:1808.02455.

Forestier, G., Petitjean, F., Dau, H.A., Webb, G.I., Keogh, E., 2017. Generating synthetic time series to augment sparse datasets, in: Proceedings - IEEE International Conference on Data Mining, ICDM, Institute of Electrical and Electronics Engineers Inc.. pp. 865–870. doi:10.1109/ICDM.2017.106.

Gonzalez, R., Woods, R., 2018. Digital Image Processing. Pearson. URL: https://books.google.no/books?id=0F05vgAACAAJ.

Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. MIT Press. http://www.deeplearningbook.org.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets, in: Advances in Neural Information Processing Systems, Neural information processing systems foundation. pp. 2672–2680. doi:10.3156/jsoft.29.5_177_2.

Goodfellow, I.J., Shlens, J., Szegedy, C., 2015. Explaining and harnessing adversarial examples, in: 3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings, International Conference on Learning Representations, ICLR. arXiv:1412.6572.

Guennec, A.L., Malinowski, S., Tavenard, R., 2016. Data augmentation for time series classification using convolutional neural networks. Data Augmentation for Time Series Classification using Convolutional Neural Networks .

Hendrycks, D., Gimpel, K., 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks, in: 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings, International Conference on Learning Representations, ICLR. URL: https://arxiv.org/abs/1610.02136v3, arXiv:1610.02136.

Hendrycks, D., Mazeika, M., Dietterich, T., 2019. Deep anomaly detection with outlier exposure, in: 7th International Conference on Learning Representations, ICLR 2019, International Conference on Learning Representations, ICLR.

Hochreiter, S., Schmidhuber, J., 1997. Long Short-Term Memory. Neural Computation 9, 1735–1780. doi:10.1162/neco.1997.9.8.1735.

Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks, in: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Institute of Electrical and Electronics Engineers Inc.. pp. 2261–2269. doi:10.1109/CVPR.2017.243, arXiv:1608.06993.

Hyndman, R., Athanasopoulos, G., 2014. Forecasting: principles and practice. OTexts. URL: https://books.google.no/books?id=gDuRBAAAQBAJ.

Iftikhar, N., Liu, X., Danalachi, S., Nordbjerg, F.E., Vollesen, J.H., 2017. A scalable smart meter data generator using spark, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer Verlag. pp. 21–36. URL: https://doi.org/10.1007/978-3-319-69462-7_2, doi:10.1007/978-3-319-69462-7_2.

Kang, Y., Hyndman, R.J., Li, F., 2020. GRATIS: GeneRAting TIme Series with diverse and controllable characteristics. Statistical Analysis and Data Mining 13, 354–376. doi:10.1002/sam.11461, arXiv:1903.02787.

Kang, Y., Hyndman, R.J., Smith-Miles, K., 2017. Visualising forecasting algorithm performance using time series instance spaces. International Journal of Forecasting 33, 345–358. doi:10.1016/j.ijforecast.2016.09.004.

Kegel, L., Hahmann, M., Lehner, W., 2017. Generating what-if scenarios for time series data, in: ACM International Conference Proceeding Series, Association for Computing Machinery, New York, NY, USA. pp. 1–12. URL: https://dl.acm.org/doi/10.1145/3085504.3085507, doi:10.1145/3085504.3085507.

Kegel, L., Hahmann, M., Lehner, W., 2018. Feature-based comparison and generation of time series, in: ACM International Conference Proceeding Series, Association for Computing Machinery, New York, NY, USA. pp. 1–12. URL: https://dl.acm.org/doi/10.1145/3221269.3221293, doi:10.1145/3221269.3221293.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks. Technical Report. URL: http://code.google.com/p/cuda-convnet/.

Lee, K., Lee, H., Lee, K., Shin, J., 2018. Training confidence-calibrated classifiers for detecting out-of-distribution samples, in: 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, International Conference on Learning Representations, ICLR.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.X., Yan, X., 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, in: Advances in Neural Information Processing Systems, Neural information processing systems foundation.

Liang, S., Li, Y., Srikant, R., 2018. Enhancing the reliability of out-of-distribution image detection in neural networks, in: 6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings, International Conference on Learning Representations, ICLR.

Lim, B., Arik, S.O., Loeff, N., Pfister, T., 2019. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. arXiv URL: http://arxiv.org/abs/1912.09363, arXiv:1912.09363.

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2018. Statistical and Machine Learning forecasting methods: Concerns and ways forward. PLoS ONE 13. doi:10.1371/journal.pone.0194889.

Meinke, A., Hein, M., 2019. Towards neural networks that provably know when they don't know. arXiv URL: http://arxiv.org/abs/1909.12180, arXiv:1909.12180.

Mitchell, T.M., 1997. Machine Learning. McGraw-Hill, New York.

Mogren, O., 2016. C-RNN-GAN: Continuous recurrent neural networks with adversarial training URL: http://arxiv.org/abs/1611.09904, arXiv:1611.09904.

Morningstar, W.R., Ham, C., Gallagher, A.G., Lakshminarayanan, B., Alemi, A.A., Dillon, J.V., 2020. Density of States Estimation for Out-of-Distribution Detection. arXiv URL: http://arxiv.org/abs/2006.09273, arXiv:2006.09273.

Mudelsee, M., 2019. Trend analysis of climate time series: A review of methods. doi:10.1016/j.earscirev.2018.12.005.

Nalisnick, E., Matsukawa, A., Teh, Y.W., Gorur, D., Lakshminarayanan, B., 2018. Do Deep Generative Models Know What They Don't Know? arXiv URL: http://arxiv.org/abs/1810.09136, arXiv:1810.09136.

Nguyen, A., Yosinski, J., Clune, J., 2015. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society. pp. 427–436. doi:10.1109/CVPR.2015.7298640, arXiv:1412.1897.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., Kavukcuoglu, K., 2016. WaveNet: A Generative Model for Raw Audio URL: http://arxiv.org/abs/1609.03499, arXiv:1609.03499.

Oreshkin, B.N., Carpov, D., Chapados, N., Bengio, Y., 2020. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=r1ecqn4YwB.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library, in: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-libra
pdf.

Pidhorskyi, S., Almohsen, R., Adjeroh, D.A., Doretto, G., 2018. Generative Probabilistic Novelty Detection with Adversarial Autoencoders. Technical Report.

Ramponi, G., Protopapas, P., Brambilla, M., Janssen, R., 2018. T-CGAN: Conditional Generative Adversarial Network for Data Augmentation in Noisy Time Series with Irregular Sampling URL: http://arxiv.org/abs/1811.08295, arXiv:1811.08295.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society. pp. 779–788. doi:10.1109/CVPR.2016.91, arXiv:1506.02640.

Ren, J., Research, G., Liu, P.J., Fertig, E., Snoek Google Research, J., Poplin Google Research, R., DePristo Google Research, M.A., Dillon, J.V., Lakshminarayanan, B., 2019. Likelihood Ratios for Out-of-Distribution Detection. Technical Report.

Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T., 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting 36, 1181–1191. doi:10.1016/j.ijforecast.2019.07.001, arXiv:1704.04110.

Shorten, C., Khoshgoftaar, T.M., 2019. A survey on Image Data Augmentation for Deep Learning. Journal of Big Data 6, 1–48. URL: https://link.

springer.com/articles/10.1186/s40537-019-0197-0https://link.springer.com/article/10.1186/s40537-019-0197-0, doi:10.1186/s40537-019-0197-0.

Stoffer, D.S., Ombao, H., 2012. Editorial: Special issue on time series analysis in the biological sciences. doi:10.1111/j.1467-9892.2012.00805.x.

Sutskever Google, I., Vinyals Google, O., Le Google, Q.V., 2014. Sequence to Sequence Learning with Neural Networks. Technical Report.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R., 2014. Intriguing properties of neural networks, in: 2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings, International Conference on Learning Representations, ICLR. arXiv:1312.6199.

Topol, E.J., 2019. High-performance medicine: the convergence of human and artificial intelligence. URL: https://doi.org/10.1038/s41591-018-0300-7, doi:10.1038/s41591-018-0300-7.

Um, T.T., Pfister, F.M.J., Pichler, D., Endo, S., Lang, M., Hirche, S., Fietzek, U., Kulić, D., 2017. Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring using Convolutional Neural Networks. ICMI 2017 - Proceedings of the 19th ACM International Conference on Multimodal Interaction 2017-January, 216–220. URL: http://arxiv.org/abs/1706.00527http://dx.doi.org/10.1145/3136755.3136817, doi:10.1145/3136755.3136817, arXiv:1706.00527.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I., 2017. Attention is all you need, in: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vernekar, S., Gaurav, A., Denouden, T., Phan, B., Abdelzad, V., Salay, R., Czarnecki, K., 2019. Analysis of Confident-Classifiers for Out-of-distribution Detection. arXiv URL: http://arxiv.org/abs/1904.12220, arXiv:1904.12220.

Wang, W., Wang, A., Tamar, A., Chen, X., Abbeel, P., 2017. Safer Classification by Synthesis. arXiv URL: http://arxiv.org/abs/1711.08534, arXiv:1711.08534.

Wu, S., Xiao, X., Ding, Q., Zhao, P., Wei, Y., Huang, J., 2020. Adversarial sparse transformer for time series forecasting, in: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (Eds.), Advances in Neural Information Processing Systems, Curran Associates, Inc.. pp. 17105–17115. URL: https://proceedings.neurips.cc/paper/2020/file/c6b8c8d762da15fa8dbbdfb6baf9e260-Paper.pdf.

Yoon, J., Jarrett, D., Van Der Schaar, M., 2019. Time-series Generative Adversarial Networks. Technical Report.

Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W., 2021. Informer: Beyond efficient transformer for long sequence time-series forecasting, in: The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, AAAI Press. p. online.

# Appendix

# Inlier and Outlier comparisons

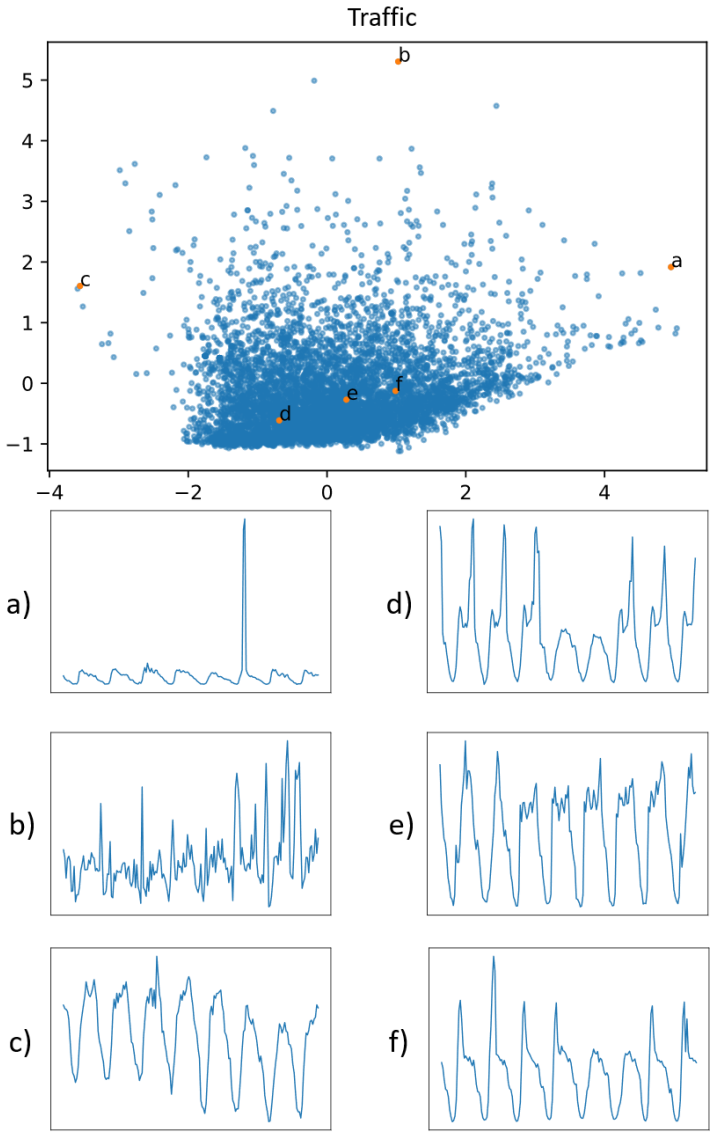



**Figure 7.1:** Outliers and inliers in the Traffic dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.

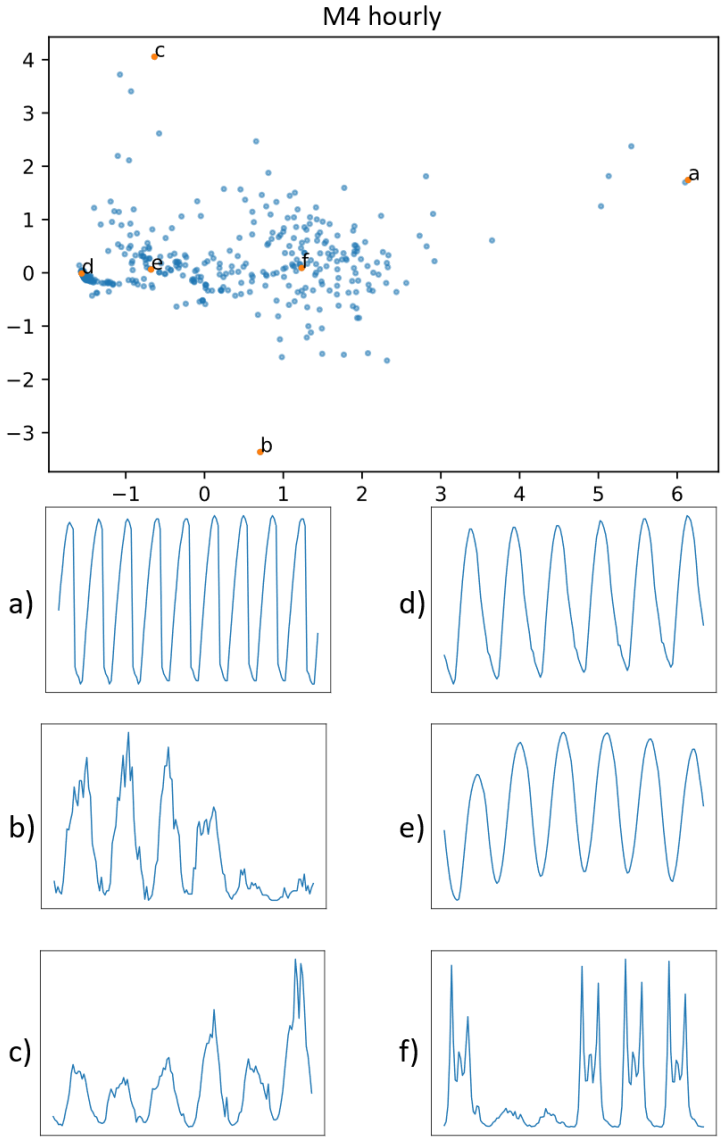



**Figure 7.2:** Outliers and inliers in the M4 hourl dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.

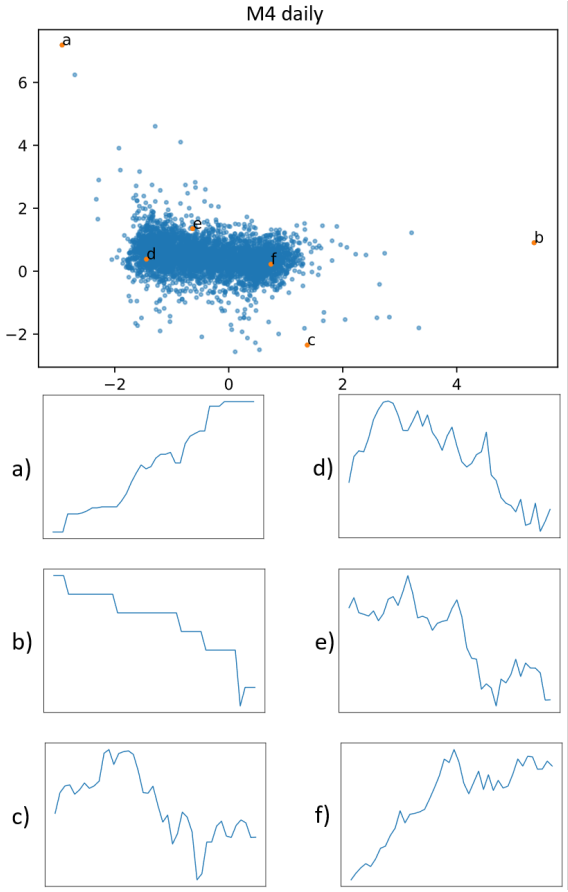



**Figure 7.3:** Outliers and inliers in the M4 daily dataset. The top plot shows the position of the time series in the instance space. The outliers are marked $a$, $b$, $c$, and the inliers are marked $d$, $e$, $f$.

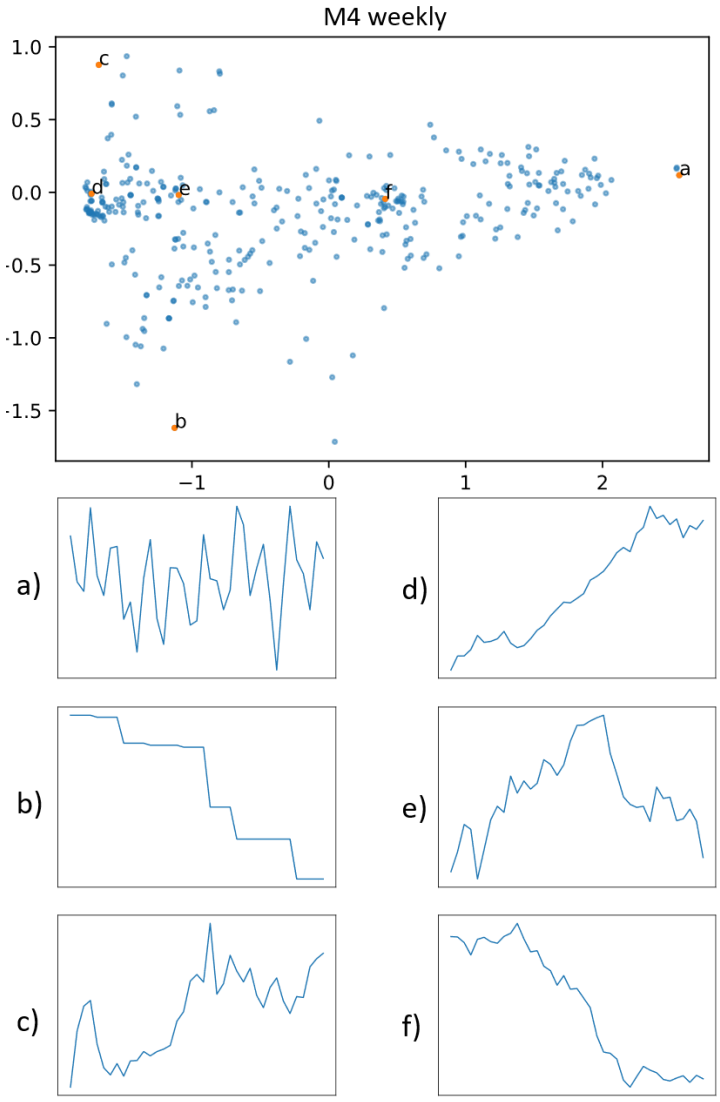



**Figure 7.4:** Outliers and inliers in the M4 weekly dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.

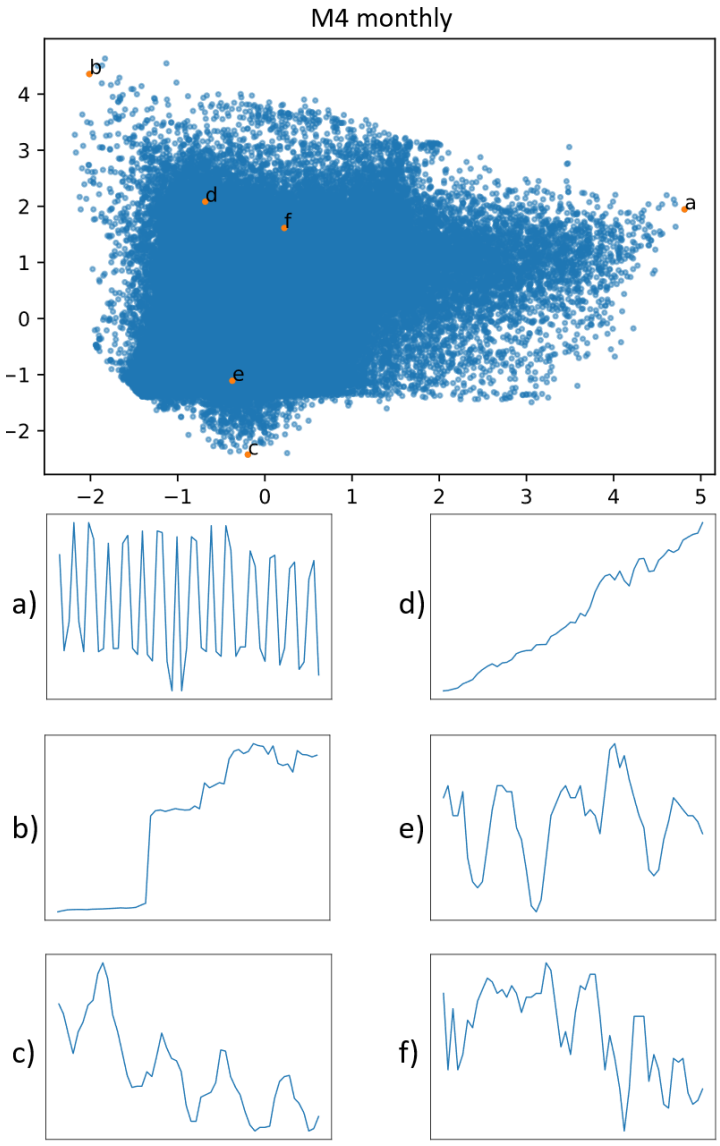



**Figure 7.5:** Outliers and inliers in the M4 monthly dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.

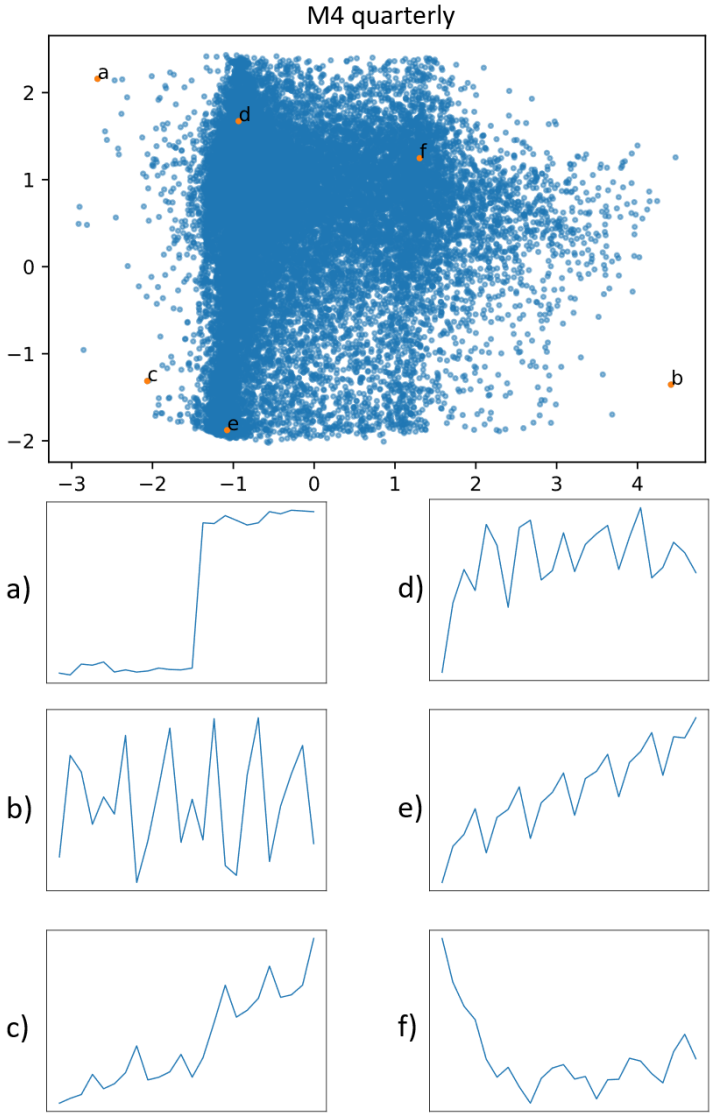



**Figure 7.6:** Outliers and inliers in the M4 quarterly dataset. The top plot shows the position of the time series in the instance space. The outliers are marked $a$, $b$, $c$, and the inliers are marked $d$, $e$, $f$.

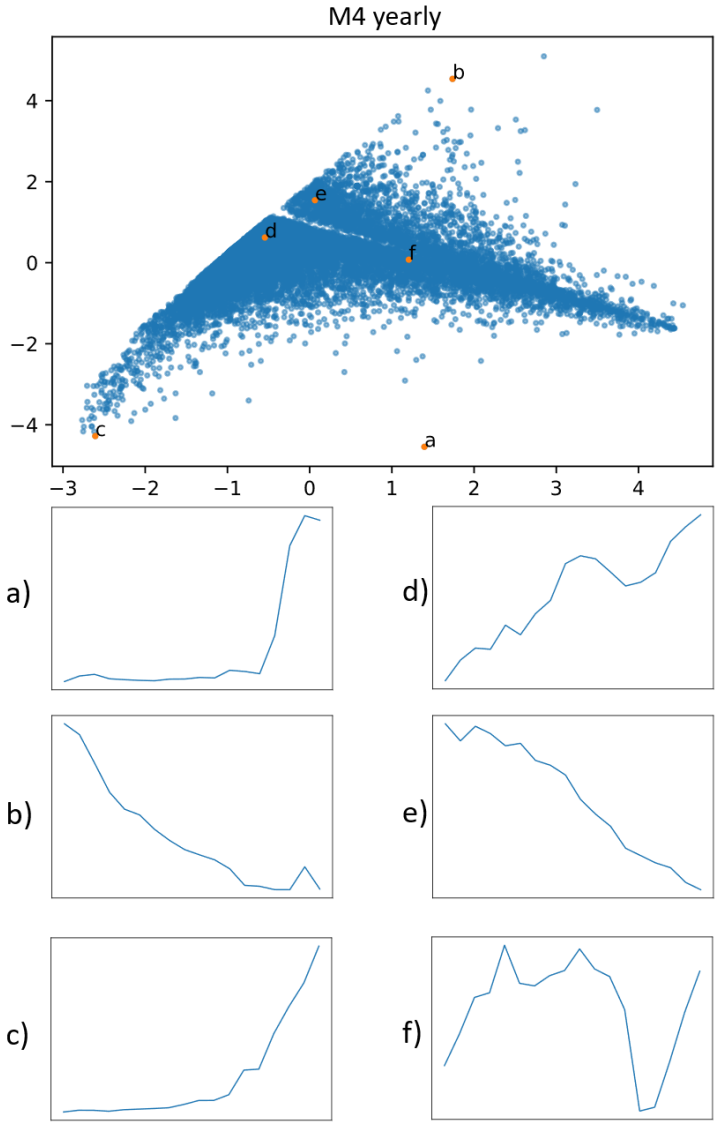
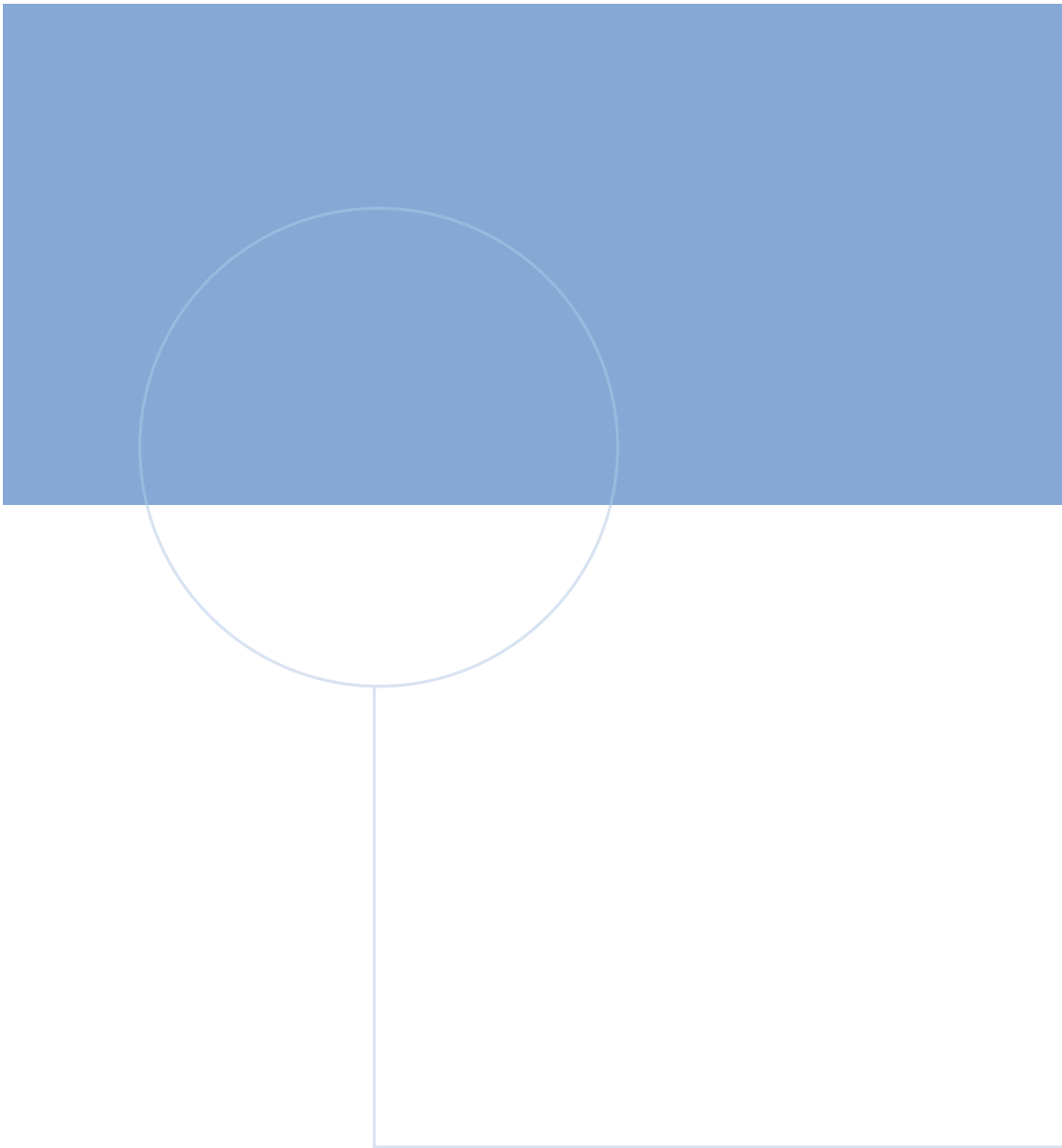



**Figure 7.7:** Outliers and inliers in the M4 yearly dataset. The top plot shows the position of the time series in the instance space. The outliers are marked *a*, *b*, *c*, and the inliers are marked *d*, *e*, *f*.



NTNU

Norwegian University of
Science and Technology