Master's thesis

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

Elen Ekeberg Klippen

Forecasting Univariate Time Series with Missing Data

Master's thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
July 2021

NTNU
Norwegian University of
Science and Technology

Elen Ekeberg Klippen

Forecasting Univariate Time Series with Missing Data

Master's thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
July 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

July 4, 2021

Summary

Forecasting of time series is to predict future values based on historical values using models somehow describing the time series data. Making informative estimates of the future are a useful tool used by our society every day. Businesses use it to plan productions based on estimated demands and investors to predict stock returns. Statistical forecasting methods have traditionally been giving the best estimates. However, Machine Learning approaches have recently given performances outperforming them. Long Short-Term Memory neural networks (LSTMs) have become popular for sequential data like time series and are used in this thesis.

Missing data are values not observed for a variable of interest and are often found in real-world time series. Forecasting time series with missing data is challenging since most forecasting techniques require the time series to be complete. Thus, the missing values must be handled properly or suffer from reduced statistical power and biased estimates. We want a forecasting model that handles missing values robustly. We add a Missing Value Indicator (MVI) to the model input representing where values are missing (1) and not (0), as this has shown to be a robust way of handle missing values in my specialization project. This approach handles missing data directly and incorporates the pattern of the missing values into the model.

The Makridakis competitions have recently shown promising results for global models. Global models are trained with numerous time series and can forecast all of them, in contrast to local models where every time series has individual forecasting models. Another learning method of increasing popularity is Multi-task Learning. A multi-task learning model is trained jointly on related tasks to improve the individual tasks by utilizing information from each other. We hypothesize forecasting and imputation to be related tasks and train them in a global multi-task model.

A global, multi-task bi-directional LSTM forecasting model with robust handling of missing values and the auxiliary task imputation is explored in this thesis. Through the experiments, the architecture of the bi-LSTM block, the weighting of the imputation loss, and the amount of missing data in the time series are varied. The main objective is to find out if imputation can improve forecasting.

We use two different data sets, a synthetic, simulated data set from SARIMA processes and a real-world electricity data set. We find the suggested multi-task

forecasting model to perform better than the single-task model. Especially for higher ratios of missing values in the test data. However, for some model architectures, the imputation task is helping the forecasting performance for all ratios of missing values. The model performs best on the real-world electricity data, where a Bi-LSTM model with three hidden layers gives the best utilization of the auxiliary task imputation. Another finding is that an additional ReLU layer makes the single-task forecasting model more robust.

Samandrag

Manglande data er ikkje-observerte verdiar for ein variabel, og er ofte å finne i tidsrekkjer frå den verkelege verda. Ved å lage modellar som beskriv tidsrekkjer kan ein lage estimat på framtidige verdiar av det tidsrekkja beskriv. Dette er eit nyttig verkty som blir brukt i samfunnet rundt oss på kvardagsleg basis. Verksemder estimerer etterspørsel og kan planleggje produksjon deretter. Investorar brukar det til å estimere aksjeavkastningar. Tradisjonelt er det statistiske metodar som har gjort det best på dette området, men i det siste har maskinlæringsmetodar blitt populære og både konkurrerer med og gjer det betre enn dei klassiske metodane. For data som tidsrekkjer har det nevrale nettverket LSTM gjort det bra på grunn av sitt minne, og er den type læringsmodell som blir brukt i denne oppgåva.

Oppgåva å estimere framtidige verdiar av tidsrekkjer når dei inneheld manglande data er utfordrande, då dei fleste metodane krev fullstendig data. Difor må dei handterast på best mogleg måte, om ikkje vil det kunne føre til redusert statistisk haldbarheit og innføring av bias. Me ynskjer at modellen som skal estimere framtidige verdiar også skal kunne handtere manglande data på ein robust måte. Måten det blir gjort på i denne oppgåva er å utvide tidsrekkjene med ein binær variabel som indikerer kvar i tidsrekkja verdiane manglar (1) og er observert (0). Dette er det nye dataformatet som læringsmodellen vil bli trena med, og slik kan modellen direkte handtere manglande data ved å inkorporere mønsteret av dei manglande verdiane i modellen. I prosjektoppgåva mi viste denne metoden seg for å vere robust mot aukande andelar av manglande data.

Makridakis-konkurransane har nyleg vist lovande resultat for globale modellar. Globale modellar er trena med mange tidsrekkjer og kan estimere framtidige verdiar for dei alle. Ein annan læringsmetode av aukande popularitet er noko kalla multitask læring. Då bli ein modell trena til å lære seg fleire relaterte oppgåver samtidig med håpet om at modellen kan dra utnytte av relasjonen mellom dei for å gjere dei individuelle oppgåvene betre. I denne oppgåva blir det utforska om det å estimere framtidige verdiar og estimere manglande verdiar er relaterte, og om det å estimere manglande data kan forbetre estimeringa av framtidige verdiar.

Det er altså ein global, multitask bi-LSTM modell som robust kan handtere manglande data som blir utforska i denne oppgåva. Hovudfokuset er å sjå om modellen trena for både estimering av framtidige og manglande verdiar vil gjere det betre enn modellen berre trena for estimering av framtidige verdiar. Gjennom eksperimenta er det fleire ting som blir varier for å sjå korleis det har innverknad

på resultatet. Dette er arkitekturen til det nevrale nettverket, vektinga av den ekstra oppgåva, samt kor mykje manglande data det er i tidsrekkjene. To ulike datasett blir brukt i ekperimenta, nemleg eit syntetisk og simulert SARIMA datasett, og eit elektrisitetsdatasett frå den verkelege verda. Me finn modellen til å oppføre seg robust mot manglande data, og den ekstra oppgåva med å estimere dei manglande verdiane forbetrer hovudoppgåva i mange av eksperimenta. Særleg er resultata signifikante for elektrisitetsdatasettet når det er mykje data som manglar. Den modellen som i størst grad får utbytte av hjelpeoppgåva er nettverket med tre gøymde lag på elektrisitetsdatasettet. Denne modellen forbetrar estimata av framtidige verdiar for alle gradar av manglande data. Me finn også at modellen berre trena for estimering av framtida blir meir robust når det blir lagt til eit ikkje-lineært ReLU lag i modellen.

Preface

This thesis marks the end of my master´s degree in Industrial Mathematics, a part of the M.Sc. program in Applied Physics and Mathematics at the Norwegian University of Science and Technology (NTNU) in Trondheim. The master thesis has been accomplished at the Department of Mathematical Sciences during spring 2021. I would like to thank my supervisor Erlend Aune for his knowledge, guidance, and support during the last year when working on the specialization project and master thesis. Much gratitude is also given to my family for support and patience, and to my friends for good memories during my time at NTNU.

Elen Ekeberg Klippen
July 2021
Trondheim

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

A time series is a time-oriented or chronological sequence of observations of a variable of interest [1]. Time series data appears everywhere since time is a component in everything observable. In addition, the digitalization of the world leads to sensors and systems constantly logging data in an unstoppable stream of time series data. This large amount of available data can potentially be of great value. A way of transforming the data into something useful is forecasting. A forecast is a prediction of some future event or trend based on historical observed data. With good estimates of the future, factories can estimate their product demands and plan their production. Investors can forecast the returns of their stocks or the risk associated with them. Governments use population growth forecasts to plan policy and social service actions like health resources, retirement programs, and kindergartens [1]. Hence, forecasting plays a role in our everyday lives.

In many time series, missing values appear. They occur due to human errors like manual data entry and incorrect measurements or defective sensors and equipment. Some standard ways of handling missing data are deleting and imputation, meaning that the missing values are either left out or replaced with a single substitute. In time series, the temporal dependency is essential, and deleting values may interrupt this continuity, and imputation methods may change the original times series, causing reduced statistical power and biased estimations [2].

A forecasting model robustly handling missing values is to desire. Developing a robust forecasting baseline that is simple, easy to implement, and naive against time-series-specific details is helpful for further understanding and investigation of the topic. My specialization project aimed to determine if training an LSTM neural network with input added a Missing Value Indicator (MVI) for handling missing data gave robust forecasts. The MVI is a binary variable indicating if a value is missing (1) or not (0). With this approach, the method directly handles the missing values without imputation or similar methods and incorporates the missing value pattern into the forecasting task. This study showed that the model behaved more robust than the LSTM model tested on imputed data when the

number of missing values increased.

With the result from the specialization project in mind, we want to explore if the model can be improved further by utilizing information from a similar task. Imputation predicts missing values while forecasting predicts future values. We want to explore if the tasks of forecasting and imputation are related and if the imputation task can improve the forecasting task. The approach of training a single model to learn several tasks is referred to as Multi-task learning. The tasks share representation during training, and in this manner, forecasting utilizes information from the imputation task that potentially leads to better forecasts.

This thesis investigates whether imputation and forecasting are related tasks and if the imputation task can help improve the robust forecasting model from my specialization project further. A bi-directional LSTM neural network is trained to learn the tasks of forecasting and imputation jointly, with forecasting as the main task. Both tasks have linear output layers for their individual tasks. The model is fed univariate time series data with a missing value indicator feature handling missing values. Additionally, the model is trained on numerous time series, making the model *global*, a promising approach from the M4 Makridakis competition. Analogous to Multi-task models, where the model utilizes information from different tasks, global models utilize information from numerous time series.

1.2 Research Questions

- Q1: Can forecasting of univariate time series with missing data benefit from its relation to imputation of missing values and perform better when the tasks are learned jointly in a multi-task model?
- Q3: Does the amount of missing data in the test data set affect the benefit of the multi-task model, and how robust is the model against an increase of missing data?
- Q3: How does the model architecture affect the performance of the multi-task model?

1.3 Contributions

A robust bi-LSTM forecasting model for time series containing missing data is explored. Imputation of the missing time series values is added as a task to the forecasting model to see if the robust model can be further improved by the additional information from the related task. We find that there is a relation between the tasks and that imputation can significantly improve forecasting in some cases, making the model more robust.

1.4 Thesis structure

The outline of the thesis is as follows. Chapter 2 introduces the theory and concepts behind time series, forecasting techniques, missing data, neural networks, global models, and multi-task learning. The theory chapter also presents the thesis model. A review of the literature in the fields of global forecasting, multi-task models, and missing data handling techniques is given in Chapter 3. Chapter 4 describes the data sets, the necessary preprocessing steps, and the experimental setup before results and discussions are given in Chapter 5. Lastly, the conclusion from the experiments and possible future work are presented in Chapter 6.

Chapter 2

Theory

2.1 Time Series

Time series can be defined as a collection of random variables indexed according to the order they are obtained in time. Generally, a collection of random variables indexed by t can be thought of as a stochastic process. The notation used for the stochastic process is $X = \{x_t\}$ with x_t being the observations/realizations at time t. The term *time series* is used to refer to both a general stochastic process and specific realizations. The temporal dependence is introducing a correlation between the data points. This correlation restricts the applicability of many statistical data analysis methods because they assume the data points to be independent and identically distributed [3]. This gives rise to the field of time series analysis. Time series data appears everywhere since time is a component of everything observable. Potential information of great value for many companies and similar is hidden in the great amount of data.

There is a distinction between *univariate* and *multivariate* time series. A time series with one single variable varying over time is called a univariate time series. Once the time series contains two or more variables, it is a multivariate time series. The type considered in this thesis is univariate time series. The time considered is *discrete-time*, meaning the observations are taken at equally spaced points in time, and the task we focus on is *forecasting* of time series.

2.2 Forecasting

Forecasting univariate time series is the process of making predictions of future values based on previous values using models describing the time series. Accurate forecasts are essential and helpful guidance for informative planning and sound decisions. For example, an application of importance from the real world this year is to forecast the trend of newly infected people of the coronavirus to plan the opening of the society and the distribution of vaccines.

The Makridakis Competitions allow researchers and practitioners to explore

and invent new time series forecasting models and approaches. The first competition resulted from statisticians criticizing Makridakis and Hibon's article [4] published in the Journal of The Royal Statistical Society in 1978. Their results showed that simple methods could perform well compared to more complex and sophisticated ones, and the statisticians meant their results were impossible [5]. There have been five Makridadis competitions so far, the last ones results were published in 2020 [6].

The literature mainly distinguishes between two different branches of forecasting, namely *statistical*- and *Machine Learning*(ML) approaches. The forecasting model proposed in this thesis is a machine learning model. The next sections introduce some of the most widely used techniques in both statistical and machine learning forecasting.

The notation used in the following sections is as follows: The subscript t indicates time, hence x_t is the observation at time t. We denote a forecast of x_t based on the history up to time $t-1$ with a hat, $\hat{x}_{t|t-1}$. If we want to forecast more than one step, we denote the forecast as $\hat{x}_{t+h|t}$ where h is the horizon of the forecast. If $h > 1$ it is referred to as *multi-step* forecasting. However the horizon in this thesis is *one-step-ahead* forecasting.

2.3 Statistical Forecasting Methods

In the classic statistical approach, the primary objective of time series analysis is to build a mathematical model to describe the data [3]. The desired forecast can be considered as a random variable because it is unknown. Some of the methods are very simple, but in many cases effective. The theory behind the statistical forecasting methods introduced below are inspired by [7].

2.3.1 Average Method

In the Average Method, the forecast is simply the average/mean of all the historical data,

$$\hat{x}_{t+h|t} = \frac{1}{t}\sum_{i=1}^{t} x_i. \tag{2.1}$$

This method can be efficient for time series with small variance and values close to the mean.

2.3.2 Naïve and SNaïve

The Naïve forecasting method assumes that the stochastic model generating the time series is the random walk, so that the forecast for every horizon, h, is the last observed value. This method is often used as a benchmark model. Mathematically it can be defined as,

$$\hat{x}_{t+h|t} = x_t.$$

An extension of this is SNaïve, where S stands for *seasonal*. In this model, it is assumed that the time series has a seasonal pattern. Seasonal patterns occur when seasonal factors influence the behavior of the time series. The seasonal period, m, is always fixed and known. For example, hourly electricity demand data is expected to have daily seasonality, thus the seasonal period becomes $m = 24$. A seasonal naive forecast can thus predict a value to be the same as the corresponding observation for the last seasonal period. The forecast for the horizon h is then

$$\hat{x}_{t+h|t} = x_{t+h-m}.$$

2.3.3 Exponential smoothing

Exponential smoothing is one of the most successful classical forecasting methods. The most basic form of exponential smoothing is called *simple exponential smoothing*. The forecasts are calculated using weighted averages of past observations. The weights decrease exponentially looking back in time such that the smallest weights are associated with the oldest observations. The one-step-ahead forecast at time t is given by

$$\hat{x}_{t+1|t} = \alpha x_t + (1-\alpha)\hat{x}_{t|t-1}. \tag{2.2}$$

$0 \leq \alpha \leq 1$ is called the smoothing parameter. Equation (2.2) is recursive and can be rewritten as

$$\hat{x}_{t+1|t} = \alpha x_t + \alpha(1-\alpha)x_{t-1} + \alpha(1-\alpha)^2 x_{t-2} + ... + (1-\alpha)^t l_0 \tag{2.3}$$

$$= \sum_{j=0}^{t-1} \alpha(1-\alpha)^j x_{t-j} + (1-\alpha)^t l_0, \tag{2.4}$$

where l_0 is the starting point of the process. This forecasting function is "flat", meaning all forecasts takes the same value, independent of the forecast horizon:

$$\hat{x}_{t+h|t} = \hat{x}_{t+1|t}, \quad h = 2, 3, ...$$

This is not suitable for time series with heavy trend and/or seasonality.

2.3.4 Holt-Winter's seasonal method

Holt-Winter's seasonal method is an extension of the simple exponential smoothing to capture trend and seasonality. This method includes the forecast equation in addition to three smoothing equations (level l_t, trend b_t and seasonal s_t), with corresponding smoothing parameters α, β^*andγ. m is the seasonal order and k is the integer of $(h-1)/m$. The Holt-Winters' additive method equations are

$$\begin{aligned}\hat{x}_{t+h|t} &= l_t + hb_t + s_{t+h-m(k+1)} \\ l_t &= \alpha(x_t - s_{t-m}) + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma(x_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m},\end{aligned}$$

and Holt-Winters' multiplicative method is

$$\begin{aligned}\hat{x}_{t+h|t} &= (l_t + hb_t)s_{t+h-m(k+1)} \\ l_t &= \alpha\frac{x_t}{s_{t-m}} + (1-\alpha)(l_{t-1} + b_{t-1}) \\ b_t &= \beta^*(l_t - l_{t-1}) + (1-\beta^*)b_{t-1} \\ s_t &= \gamma\frac{x_t}{(l_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m}.\end{aligned}$$

2.3.5 ARIMA

ARIMA is short for Auto Regressive Integrated Moving Average and is one of the most widely used forecasting techniques. It is a combination of one AutoRegressive (AR) and one Moving Average (MA) model. An AR(p) forecasting model is a linear combination of the p past values of a variable. A time series $\{x_t\}$ is an AR(p) model if

$$x_t = \alpha_1 x_{t-1} + \cdots + \alpha_p x_{t-p} + z_t \tag{2.5}$$

$$= \sum_{i=1}^{p} \alpha_i x_{t-i} + z_t, \quad z_t \sim WN(0, \sigma_z^2). \tag{2.6}$$

In contrast, a MA(q) forecasting model is a linear combination of the q past forecast errors,

$$x_t = \beta_1 z_{t-1} + \cdots + \beta_q x_{t-q} + z_t \tag{2.7}$$

$$= \sum_{i=1}^{q} \beta_i z_{t-i} + z_t, \quad z_t \sim WN(0, \sigma_z^2). \tag{2.8}$$

An ARIMA model requires the time series to be *stationary*, meaning that the statistical properties of the time series process do not change over time. Namely, the mean should be the same at all time points, in addition to the covariance between two time points t and $t-k$ is only dependent on the distance k between them, not on the specific time [8].

To make the time series stationary, differencing may be a necessary step. Then the time series is transformed to consist of the differences between consecutive

values of the original time series. This transformation can be applied several times, and how many times decides the *order* of the difference. A useful notation when working with differencing is the *backward shift operator* B. This operator shifts the data back one period ($Bx_t = x_{t-1}$). With the backward shift operator, a difference of order d can be written as

$$(1-B)^d x_t. \tag{2.9}$$

After the introduction of the backward shift operator, the AR(p) model in Equation (2.5) may be rewritten as a function of ϕ and B:

$$\phi_p(B)x_t = (1-\alpha_1 B - \alpha_2 B^2 - ... - \alpha_p B^p)x_t = z_t, \quad z_t \sim WN(0, \sigma_z^2). \tag{2.10}$$

Equivalently, Equation (2.7) can be written as a function of θ and B:

$$x_t = \theta_q(B)z_t = (1-\beta_1 B - \beta_2 B^2 - ... - \beta_p B^p)z_t, \quad z_t \sim WN(0, \sigma_z^2). \tag{2.11}$$

The I in ARIMA stands for *integrated* and represents the stationary time series. By combining the AR(p), MA(q), and the stationary time series differenced d times, an ARIMA(p,d,q) mathematically take the form

$$\underbrace{\phi_p(B)\underbrace{(1-B)^d x_t}_{\text{Stationary } x_t}}_{\text{AR(p)}} = \underbrace{\theta_q(B)z_t}_{\text{MA(q)}}, \quad z_t \sim WN(0, \sigma_z^2). \tag{2.12}$$

The parameters p and q are decided by looking at the plots of the autocorrelation function (ACF) and partial autocorrelation (PACF) of the differenced series. p is decided by looking at significant lags at the PACF plot, while q is decided by the number of significant lags in the ACF plot.

If time series have a seasonal component, ARIMA can be extended to include a model for the seasonal component, Seasonal ARIMA (SARIMA). SARIMA adds three hyperparameters to ARIMA; autoregression(AR), differencing(I), and moving average(MA) for the seasonal component. The additional seasonal elements that must be specified are the seasonal AR order, P, the seasonal MA order, Q, the seasonal difference order, D, and the seasonal period, m. For example, if m for hourly data is 24, it suggests a daily seasonal cycle. P and Q are decided in the same way as for ARIMA using ACF and PACF plots looking at the seasonal lags. The seasonal difference (I) is the difference between an observation and the corresponding observation from the previous season and is defined as

$$x_t - x_{t-m} \iff (1-B^m)x_t. \tag{2.13}$$

A SARIMA$(p,d,q)(P,D,Q)_m$ model takes the form

$$\phi_p(B)\Phi_P(B^m)(1-B)^d(1-B^m)^D x_t = \theta_q(B)\Theta_Q(B^m)z_t, \quad z_t \sim WN(0, \sigma_z^2), \tag{2.14}$$

where $\phi_p(B)$ and $\theta_q(B)$ are the equations defined in (2.10) and (2.11), while

$$\Phi_P(B^m) = 1 - \Phi_1 B^m - \Phi_2 B^{2m} - ... - \Phi_P B^{Pm}$$
$$\Theta_Q(B^m) = 1 + \Theta_1 B^m + \Theta_2 B^{2m} + ... + \Theta_Q B^{Qm}.$$

2.4 Machine Learning Forecasting Methods

Machine learning (ML), first formulated by Arthur Samuel [9] in 1959, is a branch of Artificial Intelligence where computers learn patterns from data automatically by experience without being explicitly programmed for it. *Supervised* ML algorithms learn from labeled data and are introduced further in section 2.4.2. One can distinguish between *classification* and *regression*, where classification models want to, for example, predict if an ECG measure is normal or not. Regression maps the input to its continuous response value, like the price of a house. Time series forecasting can be transformed into a supervised learning regression problem suitable for machine learning.

Deep learning is a branch under machine learning where most models are based on Artificial Neural Networks (ANN) [10]. The next section gives an introduction to different types of ANNs.

2.4.1 Artificial Neural Networks

Artificial Neural Networks (ANN) are a branch of Artificial Intelligence (AI) that dates back to the 1940s, which was when McCulloch and Pitts [11] developed the first neural model. Initially, ANNs were developed as mathematical models of the information process in biological brains [12]. Human brains act on previous experiences and make fewer mistakes over time. This is also the goal of ANNs in machine learning. For over 70 years, scientists have been developing neural networks inspired by [11].

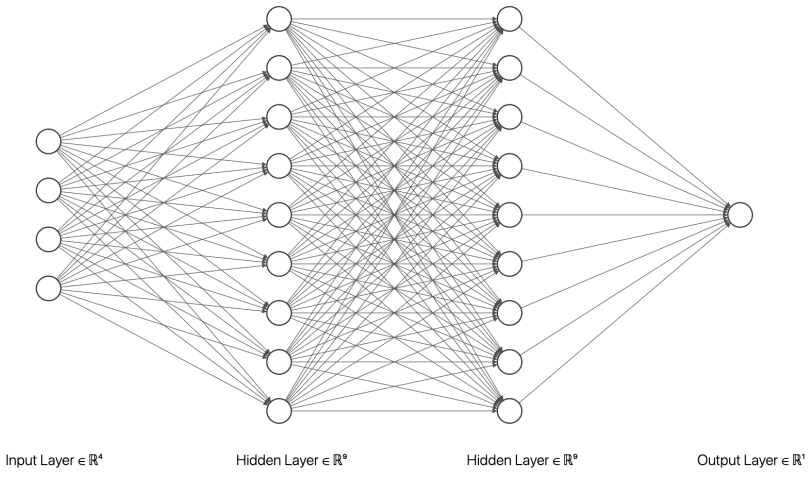


Figure 2.1: An example of a fully connected artificial neural network. This network has an input layer of size four, an output layer of size one, and two hidden layers of size nine.

ANNs consist of an input layer, an output layer, and hidden layers between

them. If the number of hidden layers is more than one, the network is called *deep*. Figure 2.1 shows an example of a deep ANN with two hidden layers. The layers contain artificial neurons, represented as circles in Figure 2.1. Edges between the neurons connect the layers, and if all neurons have all possible connections in all layers, it is called a fully connected ANN [13]. The ANN in Figure 2.1 is fully connected. The edges are directed and go from a neuron i to another neuron j. The edges have associated weights, $w_{i,j}$, deciding how important the specific edges are.

The network is activated by providing input to the neurons. The edges propagate the activation a_i from i to j through the weighted edges. Each neuron j in the network starts by computing a weighted sum of its inputs,

$$\sum_{i=0}^{n} w_{i,j} a_i,$$

which is further applied an activation function g to. This leads to neuron j's activation,

$$a_j = g\Big(\sum_{i=0}^{n} w_{i,j} a_i\Big). \tag{2.15}$$

An illustration of the process can be seen in Figure 2.2.

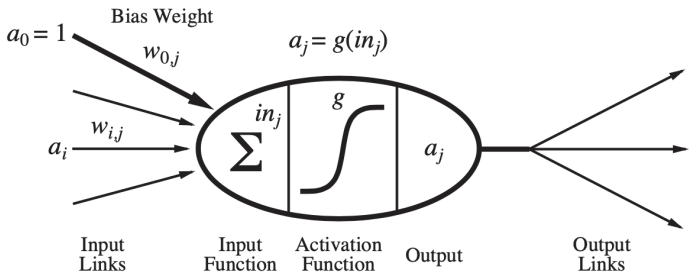


Figure 2.2: This is an illustration of a simple mathematical model for a neuron. The illustration is from [13].

Some of the most popular activation functions are the hyperbolic tangent

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{2.16}$$

the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{2.17}$$

and the ReLU function

$$ReLU(x) = \max(0, x). \tag{2.18}$$

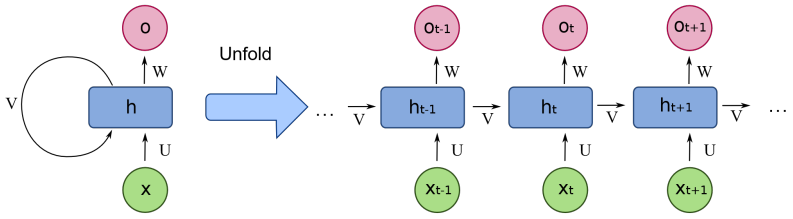


Figure 2.3: Illustration of an RNN. The unfolded version is shown to the right and is there to get a better understanding of the process.

A common feature of the three activation functions is their nonlinearity. Nonlinear networks are more powerful than linear networks since they among other things are capable of representing nonlinear patterns.

The objective to learn is the weights of the network, and with differentiable activation functions, they are trainable using gradient descent and backpropagation. The learning process is explained further in section 2.4.2.

How the neurons are connected and organized in layers decides what type of neural network it is. Two distinct methods of doing this are feed-forward neural networks and recurrent neural networks. They are both named after how they pass information through the networks.

Feed forward Neural Networks

Feed forward Neural Networks (FNN) are simple ANNs where information flows in only one direction, not allowing any cycles to appear (also indicated by the name). Hence, a specific layer can only receive input from the preceding layer, and a neuron is never fed with information more than once. The network in Figure 2.1 is an FNN. An FNN has no notion of order in time or memory and only concentrates on the current input [14]. This means that they can not remember previous inputs, which often is an advantage for sequence data like time series. In order to handle this limitation, the *Recurrent Neural Networks*(RNN) were developed.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) got attention in 1982 after the work of Hopfield [15]. The main difference from FNNs is that RNNs allow cycles to be present. At every time step, the network is fed with the current input sequence and its recent output. In this manner, RNNs are said to have a memory and are popular for sequence data. Information exists in the sequences, and RNNs with their memory can utilize this information. The hidden states preserve the memory for an amount of time decided by the weights and inputs of the RNN [16], finding correlations between events across time steps that are called *long-term dependencies* [14].

The memory process can mathematically be formulated as

$$h_t = \phi(W x_t + U h_{t-1}), \tag{2.19}$$

meaning that the hidden state at time t, h_t, is a function of the current input x_t multiplied with the weight matrix, added to hidden state at the previous time step h_{t-1} multiplied with a hidden-state-to-hidden-state matrix U. W is deciding the importance of the present input and U the importance of the previous hidden state. ϕ is a nonlinear activation function, where $tanh$ and $ReLU$ are the most used. Figure 2.3 is an illustration of what is going on.

The weights in W and U are adjusted through the Backpropagation of the loss function until convergence [14]. However, since RNNs have two inputs at each time step, and both are weighted by the same matrices W and U, the backpropagation of gradients in RNNs involves propagation through time, and each time step must sum all the previous contributions until the current one. Two problems may occur during this, namely vanishing and exploding gradients caused by the following ratio included in the backpropagation equation:

$$\frac{\partial h_t}{\partial h_{t-1}}.$$

If $||\frac{\partial h_t}{\partial h_{t-1}}||_2 < 1$, the term exponentially goes towards zero, challenging the learning of long-term dependencies because the parameter updates become insignificant. This is called the *vanishing gradient problem*. If $||\frac{\partial h_t}{\partial h_{t-1}}||_2 > 1$, the term goes exponentially towards infinity and is called the *exploding gradient problem* [17]. A solution to the vanishing gradient problem is to use *Long Short-Term Memory Neural Networks* (LSTM).

Long Short-Term Memory Neural Networks

Long Short-Term Memory Neural Networks [18] (LSTM) is a type of RNNs, designed to extract patterns in sequence data like time series where the dependencies may be several lags behind the current input value. LSTMs also addresses the problem of vanishing gradients. LSTM cells have three different gates; input-, output- and forget gates. These are regulating the information flow in the units. The cells can remember values of interest over arbitrary time intervals. Figure 2.4 shows an illustration of a LSTM cell.

The yellow boxes represent layers with different non-linear activation functions, and the pink ones are elementwise operations. Following the notation of PyTorch [20], the process can be written mathematically as:

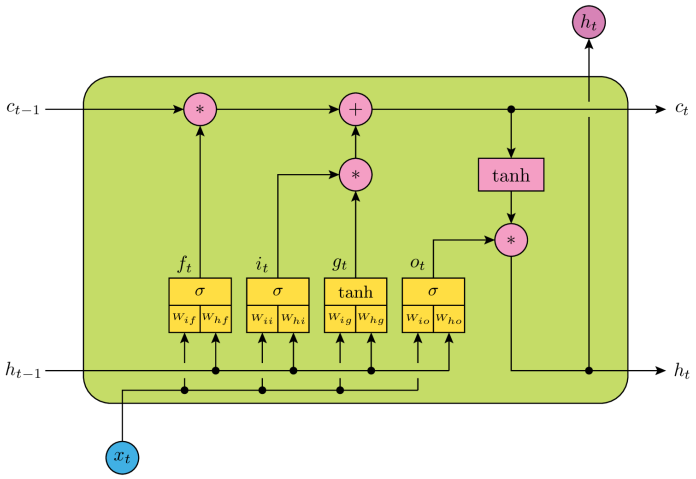


Figure 2.4: Structure of LSTM cells. Illustration from [19].

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{2.20}$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{2.21}$$
$$g_t = \tanh(W_{ig}x_t + b_{ig}W_{hg}h_{t-1} + b_{hg}) \tag{2.22}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{2.23}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.24}$$
$$h_t = o_t \odot \tanh(c_t) \tag{2.25}$$
$$\tag{2.26}$$

i_t, f_t and o_t are the input, forget, and output gates respectively at time t. h_t and c_t are the hidden state and cell state at time t, and x_t the current input. σ is the sigmoid function from equation (2.17) and $\odot$ the element-wise product. The weights W_{ii}, W_{if}, W_{ig} and W_{ig} are the non-recurrent weights associated with the input, forget, output and cell gate. W_{hi}, W_{hf}, W_{hg} and W_{hg} are the corresponding hidden-to-hidden weights.

Equations (2.20)-(2.25) are computed for each element in the input sequence in each layer. The forget gate decides what information can be forgotten, the input gate what new information to store, while the output gate determines what to output. The output gate is a filtered version of the cell state.

Bidirectional LSTMs is an extension of LSTMs, where two hidden layers of opposite directions are connected to the same output. One hidden layer is running the input forward while the other is running a reversed copy of the input. [21]. This approach may provide extra context to the network, and is the type of network used in the model of this thesis.

2.4.2 Learning

The following sections describe how the models are trained to learn their tasks. This includes the concepts of supervised learning, backpropagation, gradient descent and loss functions.

Supervised Learning

Supervised learning is a method to train machine learning models. Each of the training input variables X has corresponding output variables y, and the learning algorithm aims to learn a function mapping the input to the correct output. The correct output variables are given. Thus the algorithm iteratively predicts the training data output variables and gets corrected by making updates based on the accuracy. Supervised learning problems can be divided into *regression* and *classification* problems. When the output variables are categories, the problem is classification. The problem is a regression when the output variables are real values [22].

Times series has to be transformed to be appropriate for supervised learning. A method that makes univariate time series applicable is the *sliding window* method. For a particular time step, previous time steps are used to predict the next time step. Hence, the time series is divided into overlapping windows of size l with the following value as the corresponding output variable. An illustration of the sliding window method can be seen in Figure 2.5.

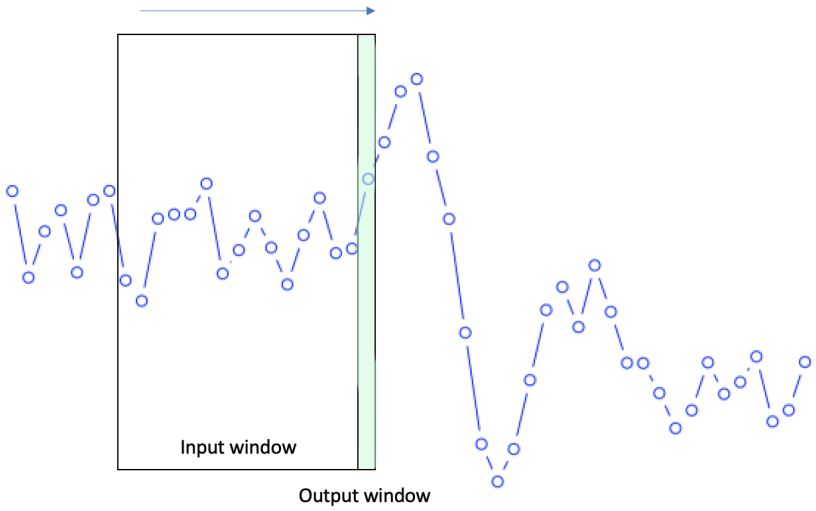


Figure 2.5: An illustration of how to make time series data appropriate for supervised learning using the sliding window method.

For a univariate time series of length T, $[x_1, ..., x_T]$, the windows with corresponding outputs will look like the following:

$$
\begin{aligned}
&X_1 = [x_1, ..., x_l], && Y_1 = x_{l+1} \\
&X_2 = [x_2, ..., x_{l+1}], && Y_2 = x_{l+2} \\
&\dots \\
&X_{T-l-1} = [x_{T-l-1}, ..., x_{T-1}], && Y_{T-l-1} = x_T.
\end{aligned}
$$

A parameter that must be decided is the length of the sliding windows, l. The larger they are, the more information about the time series is taken into account. The length of the sequence window should be at least bigger than the seasonal order such that we keep the autocorrelation of interest.

Loss Function

The forecasting models are learning by looking at the loss function. These are evaluating how well the model predicts based on the given input data. For regression problems, the following loss functions are widely used.

Mean Absolute Error (MAE) or *L1 Loss* is mathematically defined as

$$MAE = \frac{\sum_{i=1}^{n} |x_i - \hat{x_i}|}{n}, \tag{2.27}$$

where x_i is the true value and $\hat{x_i}$ is the predicted value. Hence, MAE is the average of the absolute differences between the predicted and the actual observations. MAE is a measure without direction, meaning it only measures the magnitude of the error. This function is robust against outliers [23].

Mean square error (MSE) or *L2 Loss* is defined as

$$MSE = \frac{\sum_{i=1}^{n} (x_i - \hat{x_i})^2}{n}. \tag{2.28}$$

This is the average of squared differences between predicted and actual observations. MSE is also only looking at the magnitude of the error, but it penalizes more the predictions that are far from the observed value compared to MAE. Another advantage is that it is easier to find the gradient of this metric [23]. The loss function used in this thesis is MSE, denoted by the letter $\mathcal{L}$.

Backpropagation and Gradient Descent

Training a neural network is usually done by using backpropagation and the Gradient Descent algorithm.

The forward propagation gives the model output and calculates the loss. Backpropagation is the process of going through the network backward and finding the partial derivatives of the loss with respect to the different weights. Further, gradient descent uses the derivatives to update the weights accordingly to what

minimizes the loss function [24]. One repetition of the procedures forward- and backward propagation is referred to as an epoch. Typically a neural network is trained with the number of epochs necessary to minimize the loss function. With this approach, there is a risk of overfitting the model to the training data. *Early stopping* is a regularization method for avoiding this, setting bounds for how many epochs to use, and stops the training once the model accuracy of a hold-out validation data set no longer improves [25].

Gradient descent [26] repeatedly takes small fixed-size steps toward the negative error gradient of the loss function:

$$\Delta w^n = -\alpha \frac{\partial \mathcal{L}}{\partial w^n}. \tag{2.29}$$

Δw^n represents the nth update of the weights, and w^n is the weights before the update. α is the learning rate, a number in the interval $[0, 1]$ that controls how much to change the model in response to the estimated error each time the model weights are updated[12].

Adam Optimization algorithm [27] is an extension of the gradient descent algorithm. While gradient descent has one single learning rate α, Adam computes individual adaptive learning rates for different parameters. The method combines the benefits of Adaptive Gradient Algorithm [28] (AdaGrad) and Root Mean Square Propagation [1] (RMSProp). From AdaGrad, they use the per-parameter learning rate that improves performance on problems with sparse gradients. From RMSProp, they maintain the method of the per-parameter learning rate but based on how quickly they are changing. RMSProp adapts the learning rates based on the mean, while Adam base the learning rate adaption on the variance. More specifically, Adam uses an exponential moving average process to represent the gradient and the squared gradient. The corresponding parameters control the decay rates [29].

Early stopping and Adam Optimizing algorithm are used in this thesis.

2.5 Missing Values

The increasing amount of real-world time series often includes errors in the form of missing values. They usually occur in the collection or the recording process. Defect devices, technical faults, and human errors are possible reasons for their existence. Some simple forecasting methods like Naïve forecasts can deal with the missing values by simply providing the last observed value as the forecast. However, most methods require complete time series data. There are many ways of dealing with missing values; however, choosing the best one depends on both types of data and missing values. Missing data handling may influence the statistical power and bias of the forecasting model. Hence the choice of method is essential.

[1] RMSProp is an unpublished adaptive learning rate optimizer proposed by Geoff Hinton.

Before presenting different missing data handling methods, introducing the three different missing value types is vital.

It was Rubin in 1976 [30] that divided missing values into the three different classes we widely use today. The classification system is telling the relationship between the data and the probability of missing data. He added a missing value indicator, a binary variable denoting whether a value is observed (0) or missing (1). This indicator defines a pattern of the missing data. Rubin examined how the missing value indicators probability distribution depends on the observed data in a Bayesian manner.

Data is *Missing Completely at Random (MCAR)*if the probability of missing data on a variable X is unrelated to the other measured variables included to the value X itself. Can consider it as a random sample of the complete data. In univariate time series, the probability of a value x_i missing is independent of the observation time.

Data is *Missing at Random (MAR)* if the probability of a value on variable X missing only depends on the observed values and not on the missing ones. MAR is less restrictive than MCAR. For univariate time series data, there are no other variables other than time, thus it is assumed that the probability of a value missing depends on the observation time.

Data is *Missing Not at Random (MNAR)*when the probability of values missing on the variable X is dependent on both the observed and the missing data. MNAR is the most complicated type of missing value. It is complicated to both find out if it is MNAR, and to deal with it. The missing values may be dependent on the observation time.

Missing data due to sensor recording failures will be treated as MCAR since the events have no dependencies on the missing data. However, if the sensor fails and keeps failing for some time, the data is MAR. If the sensor is out due to some latent factor as limited energy or memory, the data is MNAR [31].

The characteristics and differences between the types of missing data are essential for understanding why some techniques for dealing with them will result in valid statistical inferences and others not. Most of the simplest methods in theory like *ignoring* and *deletion* only work under the assumption of the missing data being MCAR. If this is not true, the methods will give biased estimates [32]. In this thesis data missing completely at random is considered.

2.5.1 Imputation

Imputation methods replace the missing values with appropriate estimates. Selecting the appropriate imputation method is important, and there exist both statistical and machine learning methods for imputation. Some statistical examples are mean/mode imputation, regression, interpolation, expectation and maximization, and multiple imputation [33].

Mean/median/mode imputation replaces all missing values by the mean/median/mode of the observed values. Compared to deletion and ignoring, this method

often yields better results and keeps the number of values constant. On the other hand, this method may introduce bias since all the imputed values are the same [34]. *Multiple imputations* imputes the missing values m times to represent the uncertainty about which value to use. The m different imputed datasets are combined to find estimates. This method was proposed by Rubin in 1987 [35]. Some of these statistical techniques are not applicable for univariate time series. They either are not suitable for temporal dependence or are based on the correlation between features.

Imputation methods more suitable for univariate time series are *Last observation carried forward* (LOCF), *Next observation carried backward* (NOCB) and interpolation. LOCF is one of the most used methods. Whenever a value is missing, it gets replaced with the last observed value. NOCB is similar to LOCF but works in the opposite direction by taking the first observation after the missing value and carrying it backward. Interpolation imputes the missing values based on their neighbors. However, these methods require that adjacent observations are similar to one another.

2.5.2 Missing value indicator

Rubin [30] used a missing value indicator to segregate missing values into the three different categories MCAR, MNAR, and MNAR. The same indicator is in this thesis used as an extra feature to handle missing values. The missing value indicator is a binary feature indicating if the value corresponding to the same index in the time series are missing (1) or observed (0). The missing value indicator is added to the univariate time series, meaning the learning model is fed with two-dimensional input. All the missing values (null, NaN, or NA values) in the time series are set to the same value, in this thesis chosen to be zero. The missing value indicator is also referred to as a *mask* feature.

This approach is a form of feature engineering. An essential part of machine learning is the preprocessing of the data, where the aim is to transform the data into a representation making the machine learning model effective and successful [36]. Feature engineering is a part of this process, where new features are constructed from the raw data. Hopefully, the new features can capture additional information and increase predictive power. In this thesis, the goal is also to make the model more robust against missing values with the missing value indicator.

2.6 Global Model

The traditional models are trained with the single time series they will predict, implicating the need of N models to predict N different time series. A model is *global* if it is trained with N time series and can forecast all of them. The main idea is to extract information from numerous time series as they could potentially have common patterns even though the time series are different. When training an ML forecasting model for a single time series, and the number of data points available

is small, capturing the time series characteristics and the model parameters can be challenging. This can lead to poor accuracy and is one of the limitations of ML. The use of global models can potentially cope with this problem and is one of the most important arguments in favor of this approach [37].

Another advantage with global models is the shorter time used to forecast multiple time series compared to the traditional series-by-series approach. When building a forecasting model, the most time-consuming part is the training process. Though the training cost is higher for a global model, it's only paid once compared to the series-by-series approach where the building and training must be repeated for every time series. If we want to forecast a new unseen time series in the traditional approach, a new model must be built and trained. A neural network ML model based on cross-learning can produce forecasts for the new series outside the training set in a short amount of time [37].

[38] has proven that local and global algorithms for time series forecasting are equivalent. The proposition proved is the following:
Let

- $\mathbb{A}_{Local}$ = *set of all local learning algorithms*
- $\mathbb{A}_{Global}$ = *set of all global learning algorithms*
- $\mathbb{F}_{L,\mathbb{S}}$=$\{[A_{Local}(X_i)|X_i \in \mathbb{S}], A_{Local} \in \mathbb{A}_{Local}\}$, *the set of all possible local forecasts of* $\mathbb{S}$
- $\mathbb{F}_{G,\mathbb{S}}$=$\{[A_{Global}(\mathbb{S}))(X_i)|X_i \in \mathbb{S}], A_{Global} \in \mathbb{A}_{Global}\}$, *the set of all possible global forecasts of* $\mathbb{S}$.

Then

$$\mathbb{F}_{L,\mathbb{S}} = \mathbb{F}_{G,\mathbb{S}}.$$

This implies that the local and global forecasting algorithms are theoretically able to produce the same forecasts.

From the M4[39] competition, one of the most innovative and promising forecasting approaches was global learning [40], utilized by the two best-performing submissions of the competition. However, the time series in this competition were mainly uncorrelated, and the submissions did not show the full potential of global learning. In the recent M5 competition[6], the data consisted of more correlated time series, making it easier to demonstrate the power of global models. Global methods achieved superior results compared to series-by-series methods in the M5 competition [6].

2.7 Multitask Learning

The traditional method for creating models for different tasks is to build, train and fine-tune individual ML models for the different tasks until the performance no

longer increases. However, with this approach, the model cannot capture information from related tasks that could have been helpful. This is where Multi-Task learning (MTL) can be helpful. In MTL, a model is trained to learn several tasks jointly. This is an approach to inductive transfer, defined as the ability of a model to improve performance on a specific task after having learned a different but related concept on a previous task [41].

[42] defines *related tasks* as

$$Related(MainTask, ExtraTask, LearningAlgorithm) = 1$$
$$\equiv$$
$$LearningAlgorithm(MainTask, ExtraTask) > LearningAlgorithm(MainTask),$$

meaning that if the tasks are related, this implies that a learning algorithm trained jointly on both tasks has better performance than the learning algorithm trained only for the main task. This definition is allowing the benefit of multi-task learning to be dependent on the learning algorithm.

MTL improves generalization by using domain information from the training signals of related tasks as an inductive bias. This is done by learning tasks jointly while using a shared representation. In this way, the model can use knowledge from one task to help other tasks[43]. Other advantages are reduced memory usage and faster inference speed. This because layers are shared between the tasks, avoiding features from being calculated repeatedly for each task [44]. However, achieving these advantages has shown to be challenging [45].

Several difficulties can occur in multi-task learning settings that are not present in single-task learning problems. Negative transfer/destructive interference is a phenomenon where improved performance of one task hurts other tasks. Hence, a goal in multi-task learning is to minimize the negative transfer [45]. Choosing which tasks to learn jointly and how much to share between them are of great importance.

There are two main types of multi-task learning for neural networks, namely a distinction between *hard parameter sharing* and *soft parameter sharing*. The difference is how the hidden layers are shared between the different tasks. In hard parameter sharing the hidden layers are shared between all tasks while the task-specific output layers are kept separate. This approach has been shown to reduce the risk of overfitting, and [46] shows that the risk of overfitting the shared parameters is a lot smaller than the risk of the task-specific parameters. The difference is of the same order as the number of tasks considered. Soft parameter sharing is a different approach. Here each task has a model with its parameters. Now it is the distance between the models' parameters that is encouraged to be similar. The L2 norm is often used as a regularization of the distance [45]. An illustration of the difference between the two types of parameter sharing can be found in Figure 2.6. Hard parameter sharing is what we use in this thesis.

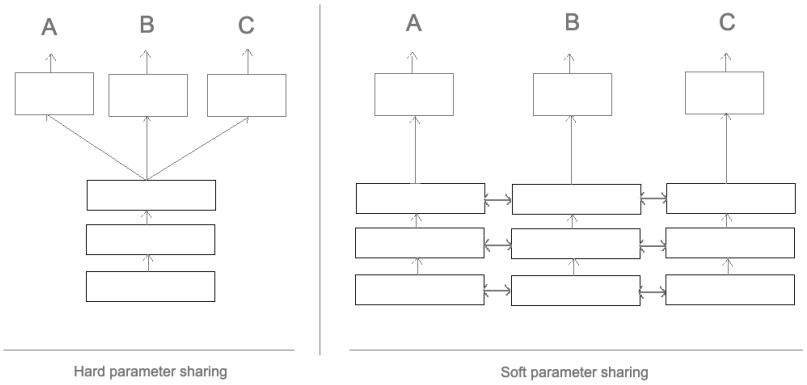


Figure 2.6: Illustration of the difference between hard parameter sharing and soft parameter sharing in multi-task learning.

2.8 Model

The forecasting model explored in this thesis is a Bi-directional LSTM neural network trained jointly with an imputation task in a multi-task manner. The aim is to determine whether information from the imputation task can improve the accuracy of the forecasting task by making the loss a weighted combination of losses from both tasks and still be robust against missing values. Missing values are handled by expanding the input with a missing value indicator. The model is also global, meaning the model is trained on numerous time series for generalization. Before further details about the proposed model, an introduction to the specialization project I did last semester will be introduced to emphasize some key findings that motivated a part of this thesis.

2.8.1 Specialization project

In the great quantity of available time series, missing values appear. The specialization project aimed to explore if expanding the input with a missing value indicator could make the forecasting model more robust against the missing values.

The model implemented was an LSTM neural network with a single hidden layer of size 100, followed by a dense linear layer giving the one-step-ahead forecasts. An illustration can be found in Figure 2.7.

The model got trained with data containing different ratios of missing values. Each of the trained models was tested on data with varying amounts of missingness. The models were also tested on imputed versions of the same test data sets for comparison. Two imputation methods were used; Last Observation Carried Forward (LOCF) and Moving window (MW) that averaged the three closest neighbors to the missing value.

The key findings from the experiments were that the forecasting model became significantly more robust by introducing only a small ratio of missing values during training, leading to a significant improvement of the forecasting accuracy,

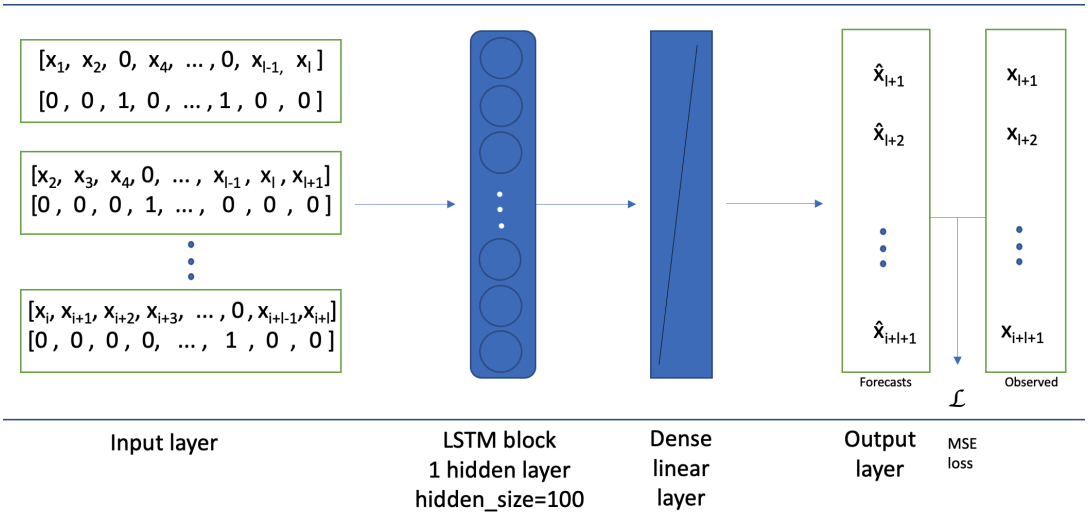


Figure 2.7: Model from specialization project. A LSTM NN with one hidden layer of size 100 followed by a dense linear forecasting layer. The hats indicates predicted values.

especially for the test data with larger ratios of missing values. The forecasting model tested with data containing missing values handled with the indicator also did it better than the model tested on imputed datasets. This was the case for all the trained models except the model trained with no missing data. The result from the experiments on the dataset "PJM Hourly Energy Consumption Data" with missing value indicator input (details in section 4.1.2) can be seen in the left plot in Figure 2.8. To the right, the test losses for the three different test data sets tested on the model trained with 2% missing values are shown.

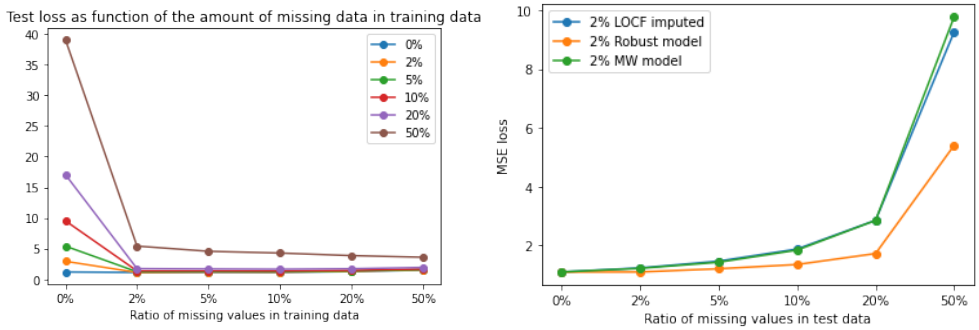


Figure 2.8: Results from the specialization project using the PJM Hourly Energy Consumption Data (see section 4.1.2). The plot to the left is showing test loss as a function of the amount of missing values the model is trained on. Each graph represents a test data set with a missing ratio. The plot to the right is showing the test losses for the three different test data sets on the robust model trained with 2% missing data.

Since the model became significantly more robust by introducing missing data during training, and the accuracy of the missing value indicator method performed better than the imputed data sets, the master thesis is based on these findings. Therefore the multi-task model is trained with 2% missing data.

2.8.2 A multitask forecasting model

The model explored in this thesis is the specialization project model (section 2.8.1) added a second task, namely an imputation task. The aim is to find out whether imputation and forecasting are somehow similar, such that the forecasting task is improved using information from the imputation task. Hence, it is a multi-task model with forecasting as the main task with imputation as an auxiliary task.

The tasks share a bi-directional LSTM neural network, while they have individual linear layers for their respective tasks; forecasting and imputation. An illustration of the model in Figure 2.9. Since the model is trained for both tasks jointly, the total loss given as feedback to the model during training is a weighted combination of the two losses,

$$\mathcal{L} = \mathcal{L}_1 + \alpha \mathcal{L}_2, \tag{2.30}$$

where $\mathcal{L}_1$ is the forecasting loss, $\mathcal{L}_2$ the imputation loss, and α the weight of the imputation loss.

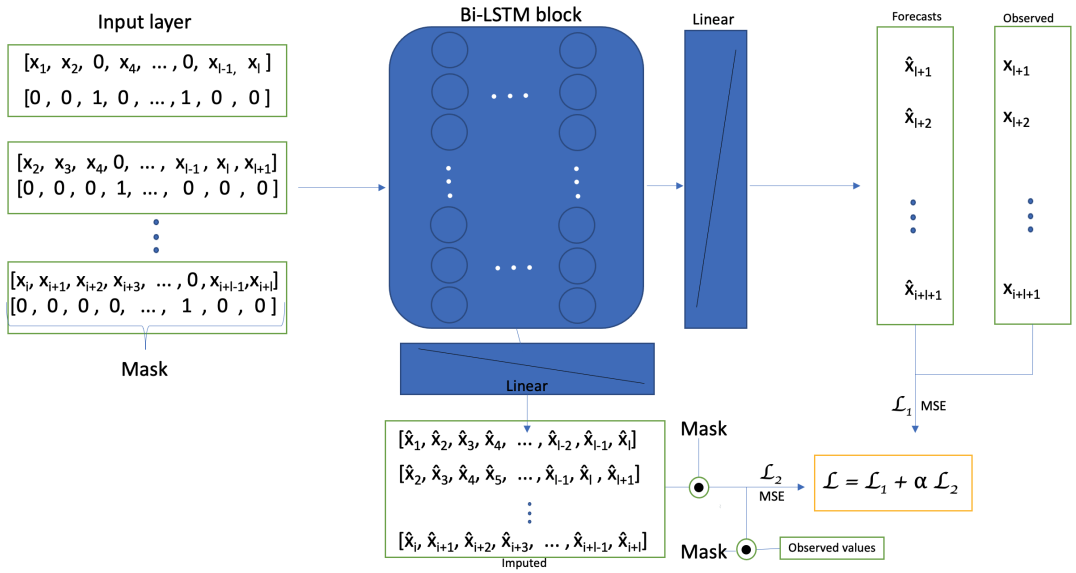


Figure 2.9: The proposed model: A multi-task Bi-directional LSTM model for forecasting using imputation as an auxiliary task.

The output from the bi-LSTM block has shape (batch_size, sequence_length, 2*hidden_size) and contains the hidden features (h_t) from the last layer of the LSTM for each t. This is illustrated in Figure 2.10. The *forecast* task passes the

hidden feature h_t for the last time step through a dense linear layer giving the one-step-ahead forecasts for each sequence in the batch. For the *imputation* task, by looping over the time steps, the hidden feature corresponding to the current time step is passed through a linear layer predicting the current value at time t for all sequences in the batch. In this manner, all observations in the batch are predicted. However, the imputation estimates are multiplied with the missing value indicators making sure the loss is only calculated for the values actually missing.

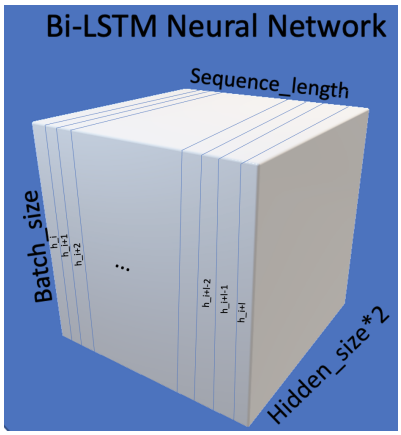


Figure 2.10: Illustration of output from the last layer of the Bi-LSTM network.

The number and size of the hidden layers in the Bi-LSTM network are varied during the experiments to explore whether the capacity affects the outcome of the multi-task model. However, the base bi-LSTM model contains one hidden layer of size 100.

Chapter 3

Literature review

3.1 Global Forecasting Models

Statistical forecasting methods like ARIMA and Exponential Smoothing have traditionally given the most accurate predictions in the Makridakis competitions. More advanced and complex methods have not been able to compete with the accuracy. However, the M4 competition had a winner that utilized both statistical and machine learning approaches [47]. The winner was *ES-RNN* [48] created by Slawek Smyl from Uber Technologies.

ES-RNN is a partly global, hybrid of Exponential Smoothing (ES) and LSTM blocks. In ES, the time series can be decomposed into trend, seasonality, and level. In ES-RNN, LSTM predicts the trend, while ES predicts the level and the seasonality. Even though the combination of ML and statistical methods did it well, the pure ML contributions to the M4 competition did not even beat the statistical benchmarks, leading to the conclusion by Makridadis that pure ML models can not be as accurate as statistical methods [47]. *N-BEATS* disproved this conclusion.

N-BEATS (Neural Basis Expansion Analysis for interpretable Time Series forecasting)[49] is a pure deep learning forecasting method without time-series specific components. It was the startup ElementAi co-founded by Yoshua Bengio that published this method. The architecture consists of modified multilayer FC networks with ReLU activations. In this manner, the architecture is simple and generic yet deep. N-BEATS outperforms statistical benchmark methods on M3, M4, and Tourism datasets. The method shows that the deep learning model can be trained on multiple time series and successfully sharing their individual information. This method also beats the score of the winner of the M4 competition, ES-RNN.

Recently [38] proved that local and global models for time series forecasting are equivalent, meaning they are theoretically able to produce the same forecasts. Hence, the literature is showing promising tendencies for pure ML global forecasting models.

3.2 Multi-Task learning

Multi-task learning is the process of training a model to learn several tasks simultaneously, with the goal of common advantage. The idea has been successful in Natural Language Processing (NLP), where [50] created a convolutional neural network (CNN) language model to be trained for multiple tasks jointly. The tasks learned were predictions of part-of-speech tags, chunks, semantic roles, a likelihood of the sentence making sense, entity tags, and semantically similar words when fed sequences of words.

[51] developed a method called *Fast R-CNN*, and is using multi-task learning for image processing where the tasks object detection and their spatial placement is trained jointly. They find their model to be much faster than other methods for object detection.

BRITS [52] is a multi-task method for imputation and classification/regression of multivariate time series data. BRITS is built on a bi-directional recurrent neural network. The model is fed input containing the time series, in addition to a masking indicating missing values (0 where a value is missing and 1 if observed) and a time gap feature with the time gaps from the last observed value to the current value for all variables. They find BRITS to outperform state-of-the-art methods for both imputation and classification/regression.

3.3 Missing data

Missing data handling is a problem well-studied across all domains for which they occur. Traditional methods for dealing with the missing values are deletion methods, simple imputation methods, and learning-based methods. Imputation for time series can be divided into two classes; imputation for multivariate time series and imputation for univariate time series.

Multivariate time series methods often utilize the relationship between different variables to fill in missing values like in the Generative Adversarial Network (GAN) based methods [53] and [54].

MICE [55] is a multiple imputation method by chained equations widely used for multivariate time series and MAR missing values. The method accounts for the statistical uncertainty in the imputations and can handle different types of variables. *MAIN* [56] is an ML imputation method for multivariate time series based on multi-head attention to deal with missing data. The method uses a Positional Encoded Vector (PEV) consisting of a binary mask to indicate missing features and an orthogonal basis of the feature existence. The method has been shown to outperform many state-of-the-art methods.

NAOMI [51] is a non-autoregressive imputation method for sequences, meaning the imputations are not based only on previous time steps. Instead of conditioning only on the past, NAOMI uses both the past and the (predicted) future to impute. This method uses masking of the missing values in the sequences, with zeros indicating missing values.

[53] is a GRU-based classification method for multivariate time series that not use imputation, but handles the missing data directly using a mask vector to indicate where the data is observed (1) and missing (0). The method also maintain the time since last observed observation for each of the variables in an own variable.

BRITS is a bi-directional RNN prediction model similar to what is explored in this thesis, however it contains an additional time-series specific feature and have the opposite masking in the missing value indicator.

Thus, there are several imputation methods using the missing value indicator. Some methods are using the missing value indicator with opposite masking and in combination with other time series specific features. The disadvantages with this is that is makes the model advanced to implement. In this thesis the missing value indicator is not used solely for imputation, and we use it for univariate time series.

Chapter 4

Experimental Setup

4.1 Data

In the following sections, the different time series data sets used in the experiments are presented. In addition, all necessary preprocessing steps for making the time series suitable for LSTM neural networks and supervised learning is described.

4.1.1 SARIMA Data

The SARIMA data set consists of synthetical data from numerous SARIMA processes. There are 100 time series of length 50 simulated in R [57] using the function `sim.ssarima` from the `smooth` [58] package. The function `sim.ssarima` takes as input the parameters p, d, q, P, D, Q and m, and simulate random time series from the specified SARIMA$(p,d,q)(P,D,Q)_m$ model. The parameters used are $m = 24$, $p, P, q, Q \in \{0, 1, 2, 3\}$, and $d, D \in \{0, 1\}$. Ten of the simulated time series are plotted in Figure 4.1.

Key features of this data set is that the time series are of the same length, short with only 50 observations, synthetic and regular.

4.1.2 PJM Hourly Energy Consumption Data

The second data set consists of 10 real-world hourly electricity time series from the data set 'PJM Hourly Energy Consumption Data' from the open source Kaggle database [1].

The data set contains over 10 years of hourly energy consumption data from PJM in Megawatts. PJM is a regional transmission organization in the United States which serves electric power to many states in the US. In Figure 4.2, the time series are plotted together. As observable, the time series are of different lengths, means, and variances. They are also heavily seasonal, having daily seasonal patterns since the data is hourly.

[1] `https://www.kaggle.com/robikscube/hourly-energy-consumption`

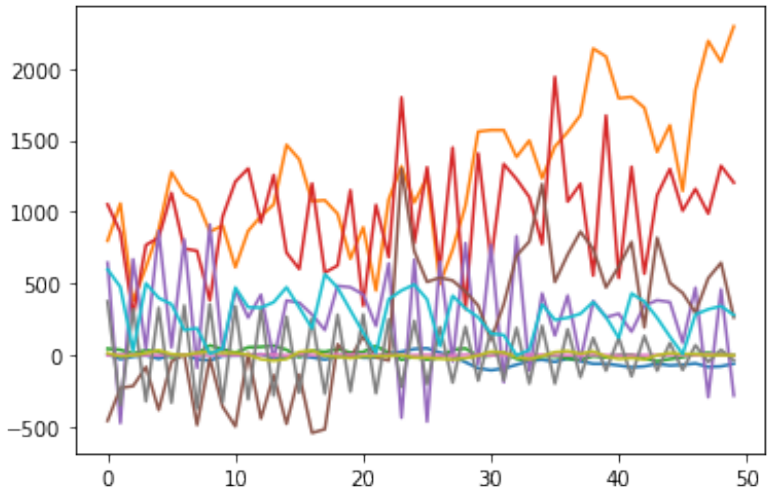


Figure 4.1: A plot of ten different time series from the SARIMA dataset.

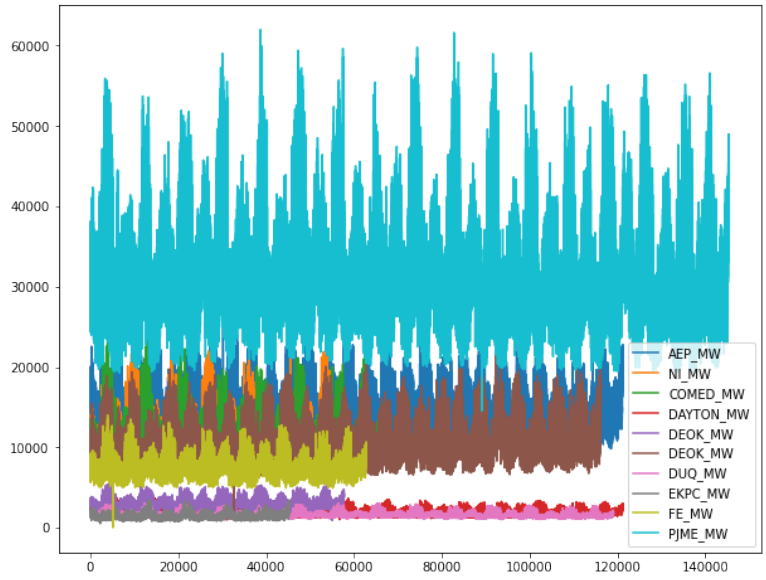


Figure 4.2: The ten electricity time series plotted together.

Key features of this data set is that the time series have different lengths, where the longest time series is containing 145366 values corresponding to over 16 years of hourly observations, while the shortest time series of 45334 values includes about five years of hourly observations. The time series are also seasonal with $m = 24$ and from the real world.

4.1.3 Preprosessing

During the preprocessing, the time series must be made appropriate for supervised learning, scaled, added missing values, and split into train- and test data sets.

Supervised learning

In order to train the model in a supervised manner, the time series must be transformed. Supervised learning requires each input variable to have a corresponding output variable. The method used to make the univariate time series appropriate for this type of learning is the *sliding window* method (see section 2.4.2 for details). For a particular time step, previous time steps are used to predict the next time step. Hence, the time series is divided into overlapping windows of size l with the following value as the corresponding output variable. A parameter that must be decided is the length of the sliding windows, l. The larger they are, the more information about the time series is taken into account. l should therefore be at least greater than the seasonal order m if the time series contains a seasonal pattern to preserve the autocorrelation of interest. For hourly data, $l >= 24$. In this thesis, l is set to 25 since both data sets are hourly, and 25 covers the daily pattern.

Missing data simulation

In the experiments, the ratio of missingness in the data is varied. Hence missing data must be simulated accordingly. Also, complete data is needed to compute the loss of the forecasted and imputed values, which is another reason why the missing data must be simulated.

The type of missing values simulated is MCAR. This is done by giving each value in the time series a probability of being missing corresponding to the desired ratio of missing values, p_{miss}.

Hence, a missing data indicator is created using a Bernoulli distribution with parameter p_{miss}. The missing value indicator is binary and takes the value 1 where data is missing, and 0 if not. In this thesis, the missing value indicator is added as a feature to the time series. The missing values are set to zero in the original time series. A time series of length T can then look like the following:

$$
\begin{bmatrix} x_1, x_2, 0, ..., x_{T-1}, 0 \\ [0, 0, 1, \quad ... \quad 0, \quad 1] \end{bmatrix}
$$

Train/test split

The time series are split into training and test data in the following way: The 70% first values in the time series are used as training data, the following 15% are used as validation data, and the remaining 15% is used as test data. An illustration can

be seen in Figure 4.3. The validation data is held out from training and is used to give an unbiased estimate of how good the model predictions are during the hyperparameter tuning of the model.

Figure 4.3: An illustration of how the time series is split.

Since we are interested in training a global model, the train-test split approach explained above is applied to all the time series used for training. After the split, the training data from all time series are merged into one training dataset, and the same for validation- and test data.

Scaling

The data must be scaled before feeding it to the neural network. This is important because the time series has different spreads, and the network weights trained on unscaled data can then become very large, making the model unstable with high generalization error [59]. Both the input and output variables must be scaled. In this thesis, the `scikit-learn`[60] RobustScaler is used. This method removes the median and scales according to the interquartile range and is robust against outliers. The scaler is only fitted to the training data, and then the scaler is used to transform the validation- and test data. This approach is necessary to avoid information leakage between the datasets.

4.2 Experiments

The experiments aim to determine whether the imputation task improves the forecasting task by training the models in a multi-task manner. The experiments are based on the robust forecasting model investigated in the specialization project. The findings indicated that the forecasting model became significantly more robust when 2% missing values were introduced during training. From the specialization project to the model in this thesis, some adjustments are made. First, the LSTM block is made bi-directional. Secondly, the imputation task is added, making the model a multi-task model. The model is trained on data containing 0% and 2% missing data due to the findings in the specialization project. Training the model on complete data is to monitor how the model robustness behaves when introducing missing values during training for the model in this thesis. In addition, this model can be used as a baseline for the multi-task model.

The base architecture of the model is a bi-LSTM block with one hidden layer of size 100 with linear task-specific output layers. First, the model is trained without any imputation loss. Secondly, the imputation loss weight (α) is gradually increased. For all α's considered, the model is tested on time series with six different

ratios of missing data; 0%, 2%, 5%, 10%, 20%, and 50%. The missing values are handled directly without imputation using missing value indicators as an extra feature. The training, validation, and test data all have missing value indicators. The imputation task added is there only to help the forecasting loss. The estimated imputation values are never included in the input variables.

First, the base architecture with one hidden layer of size 100 is the focus. Secondly, the bi-LSTM block architecture is first made deeper and then wider to explore how the capacity affects the utilization of the imputation task. As a final experiment, more non-linearity in the form of a ReLU layer between the last hidden layer of the bi-LSTM block and the linear forecasting layer is added. For all architectures, the imputation task is increased gradually in the same way as described for the base architecture.

4.2.1 Implementation

Multi-task model

The models are implemented using `PyTorch` [61], a machine learning library in `Python` [62] and the wrapper `PyTorch Lightning` [63]. Pytorch Lightning makes the code easier to read and train, and makes the model scalable to run on any hardware of interest (CPU, GPU, or TPU). The code is written in `Google Colab`, a free cloud service delivered by Google that provides a fast training process when trained on their Tesla K80 GPU, which is used.

The neural network is trained using the Adam optimizer with a learning rate of 0.001 for the SARIMA data and 0.0001 for the Electricity data. For the bi-LSTM block, a dropout probability of 0.2 is used to avoid overfitting. The network is fed batches of sequences, where the batch size is the number of sequences to work through before updating the model parameters. For the SARIMA data, a batch size of 64 is used, while for the electricity data the batch size is 128. A maximum of 15 epochs is used, meaning this is the maximum number of times the learning algorithm works through the whole training data set. Another regularization tool to avoid overfitting used is early stopping, which stops the training when the validation performance is no longer improving.

In Table 4.2.1 an overview of different values and training parameters used can be found. The code is available at GitHub [2].

[2] https://github.com/elenek97/TMA4900

Description	Explored
Time Series	Univariate
Loss	MSE
Optimizer	Adam
Learning rate	[0.001, 0.0001]
Dropout	[0.2]
Epochs	Maximum 15
Early stopping	validation loss
Window size	[25]
Batch size	[64,128]
LSTM layers	[1,2,3,5]
Size of LSTM layers	[100, 200]
Output layer	Linear, ReLU+Linear
Ratio of missing data	[0, 2%, 5%, 10%, 20%, 50%]
Weigth of imputation loss	0.0-1.5

Table 4.1: Information and training parameters for the thesis model.

SARIMA model

As a baseline, SARIMA$(p,d,q)(P,D,Q)_m$ models are fitted to each of the time series, and the forecasts are compared to the forecasts by the multi-task model. The Algorithm 1 describes how the forecasting is done. Implemented in Python using the `pmdarima` package [64] and the function `auto_arima` with seasonality that automatically finds the best SARIMA model for each time series.

Algorithm 1 Forecast With SARIMA

```
0: procedure FORECASTWITHSARIMA(Dataset)
0:   ForecastMatrix←[] {Matrix to store forecasts}
0:   for TimeSeries in Dataset do
0:     t ← the first time step in test data
0:     model ← AutoSARIMA(TimeSeries)
0:     ForecastVector ← []
0:     for i ← t to length(test data) do
0:       Fit model to time series up to time t-1
0:       Forecast time series at time t
0:       Add forecast to ForecastVector
0:     end for
0:     Add ForecastVector to ForecastMatrix
0:   end for
  return ForecastMatrix
  end procedure=0
```

Chapter 5

Results and Discussion

The thesis is motivated by the robustness findings from the specialization project. Consequently, the first experiment is to check whether the model still behaves more robust when introducing 2% missing values during training when the network changes from uni- to a bi-directional LSTM. Further, the forecasting model is extended with the second task, imputation. Experiments are done for different weightings of the imputation loss to see how this affects the forecasting accuracy for the different ratios of missing values in the test data. After that, the model architecture is varied to determine if the model needs more capacity to utilize the information from the imputation task. Lastly, the model is added an additional layer, namely a non-linear ReLU layer between the last hidden layer of the bi-LSTM block and the linear forecasting output layer. The forecasting accuracies will be compared to the statistical forecasting baseline SARIMA. All results with discussion are presented in this chapter. The best results for the individual models are highlighted with bold font in the tables.

5.1 From LSTM to bi-directional LSTM

The first thing to check is whether the change from LSTM (specialization project model) to bi-directional LSTM still gives more robust forecasts when the model is introduced to missing values during training. The change from uni- to bi-directional LSTM is due to the property of bi-LSTM running the inputs in two ways. In this manner, information from both past and future is preserved, and thus bi-LSTM often understands the context better. This adjustment hypothesizes that the imputation task will perform better with the bi-directional LSTM since the values imputed are not only based on past values.

The model has not been introduced for the imputation task during this experiment, meaning the imputation loss weight α equals zero. Table 5.1 is showing the forecasting test losses for both LSTM and bi-LSTM trained with 0% and 2% missing data, tested with data between $0\% - 50\%$ missing data. Both have one hidden layer of size 100.

Dataset	Type of LSTM	% missing data in training data	Test loss					
			0% missing in test data	2% missing in test data	5% missing in test data	10% missing in test data	20% missing in test data	50% missing in test data
SARIMA	LSTM	0%	0.5094	0.5225	0.5353	0.5666	0.6397	0.8142
SARIMA	LSTM	2%	0.5065	0.5170	0.5281	0.5579	0.6232	0.7964
SARIMA	Bi-LSTM	0%	0.5138	0.5233	0.5282	0.5290	0.5703	0.7075
SARIMA	Bi-LSTM	2%	0.5127	0.5221	0.5272	0.5267	0.5671	0.7018
ELECTRICITY	LSTM	0%	1.1789	2.9214	5.3970	9.5165	17.0470	39.0487
ELECTRICITY	LSTM	2%	1.1047	1.1047	1.2123	1.3579	1.7299	5.4064
ELECTRICITY	Bi-LSTM	0%	1.0948	3.9040	8.1008	14.9697	27.7122	56.4444
ELECTRICITY	Bi-LSTM	2%	1.0487	1.1066	1.1992	1.4148	2.0929	9.5105

Table 5.1: MSE test losses from the specialization project model (LSTM) with one hidden layer of size 100 trained with 0% and 2% missing data, and likewise for the thesis model (bi-LSTM). The results for both the SARIMA data and the Electricity data can be found in this table.

For each combination of data set and model, the test losses decrease from the model trained with 0% missing data to the model trained with 2% missing data for all the different test data. For the SARIMA data, the improvements are not very big. However, since this data set is synthetic and very regular, the fact that it is stable indicates that the model is working, at least not making the results worse. For the electricity data set, the improvements are still significant when changing to bi-LSTM. These results indicate that the model still becomes more robust after introducing 2% missing data during training even though the model has changed from uni- to bi-directional LSTM. The overall performance of the bi-LSTM model is worse than the uni-directional LSTM. However, since the hypothesis is that the bi-directional network may understand contexts better and be a good choice for the imputation task that has not yet been introduced, the bi-LSTM network is chosen to be the base model.

5.2 From Single-task to Multi-task model

In the following experiments, the model change from a single- to a multi-task model. The hypothesis is that forecasting and imputation of missing values are two tasks that are somehow related. If the hypothesis is true, the model trained jointly on both tasks will give more accurate forecasts than the model trained solely for forecasting. The architecture is the same as before, i.e., the bi-LSTM block has one single layer of size 100. The two tasks have task-specific dense linear layers while they are sharing the bi-LSTM block. The weight of the imputation loss α is gradually increased during the experiment to study how the loss weight affects the forecasting performance. Table 5.2 and 5.3 shows the results from the two data sets individually.

It is observable in Table 5.2 that the imputation task does not improve the forecasting accuracy before the missing ratio in test data is 50%. However, the improvement is minimal and could be random. Overall, the SARIMA data results indicate that the imputation task does not make the forecasts any worse. For the electricity data results in Table 5.3, the benefits of the imputation task are not

α (weight of imputation loss)	Bi-LSTM layers	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				0.5138	0.5233	0.5282	0.5290	0.5703	0.7075
	1	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				**0.5127**	**0.5221**	**0.5272**	**0.5267**	**0.5671**	0.7018
0.02				0.5128	0.5222	**0.5272**	**0.5267**	**0.5671**	0.7017
0.5				0.5153	0.5244	0.5296	0.5285	0.5677	0.7006
1.0				0.5162	0.5253	0.5305	0.5288	0.5675	**0.6992**

Table 5.2: Test losses for SARIMA data. The bi-LSTM has one hidden layer of size 100.

α (weight of imputation loss)	Bi-LSTM layers	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				1.0948	3.9040	8.1008	14.9697	27.7122	56.4444
	1	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				**1.0487**	**1.1066**	**1.1992**	**1.4148**	2.0929	9.5105
0.02				1.0785	1.1330	1.2225	1.4187	**2.0177**	9.0475
0.5				1.1739	1.2306	1.3287	1.5311	2.1003	**6.7117**
1.0				1.1941	1.2512	1.3408	1.5423	2.0881	6.9421
1.2				1.1943	1.2483	1.3341	1.5298	2.0762	6.7869

Table 5.3: Test losses for electricity data. The bi-LSTM has one hidden layer of size 100.

seen until the test data contains 20% missing data, however, it is for 50% missing data the improvements become significant. The test loss greatly improves for 50% missing data when $\alpha = 0.5$. This improvement substantiates the hypothesis of the tasks being related and that forecasting can benefit from imputation at least when the missingness is high. A theory of why the multi-task model does not improve the performance on the test data with lower missing ratios is that the problem could be too complex for the simple model. As a result of this, the model is changed in the following experiments to have increased capacity. Capacity can be increased by either making the network deeper (adding hidden layers) or wider (making the hidden layer bigger).

5.3 Increased Capacity

With the new hypothesis of the forecasting model needing more capacity to take advantage of the imputation task, the capacity is increased. The first approach considered is making the bi-LSTM block deeper by adding hidden layers of size 100, and secondly, the width of the hidden layer is doubled from 100 to 200.

5.3.1 Deeper model

Models with $1, 2, 3$ and 5 hidden layers of size 100 in the bi-LSTM block are studied to determine how the model's deepness affects the forecasting performance. The task-specific layers are unchanged. Table 5.4 and Table 5.5 are showing the results for the SARIMA data and the electricity data, respectively. The tables are showing

how the forecasting accuracies act with respect to the number of hidden layers in the bi-LSTM block and the weight of imputation loss.

For both data sets, the deepness seems to have an impact on the forecasting performances. For the SARIMA data, it is observable from Table 5.4 that the model with two hidden layers performs better when trained on 2% missing data and $\alpha = 0$ compared to the one-layered model for the three lowest ratios of missing data. On the other hand, for the higher ratios of missing data in the same model, the loss increases compared to the base model. However, when $\alpha > 0$, the model performances of all missing ratios are improved. Hence it seems like the increased capacity is affecting the benefit from the imputation task. For the three lowest test missing ratios, the loss weight $\alpha = 0.02$ gives the most significant improvements from the model without imputation loss. For the highest test missing ratios, $\alpha = 1.2$ gives the best performances and beats the base model both when trained with complete and missing data for $\alpha = 0$ (except for test data with 50% missing but this could be random since it is very close). The results for 3 and 5 hidden layers show that the overall model performance worsens for this data set. For 3 hidden layers, the performance improves when adding the imputation task. However, this is not the case for 5 hidden layers. This is indicating that the deepest models probably is overfitted to the training data. The SARIMA data set is small, and a too big model easily overfits the data. Even though the individual time series used for training is short, the model with 2 hidden layers seems to benefit from the multi-task setting across time series to improve the forecasting results.

Alpha (weight of imputation loss)	Bi-LSTM layers	Relu	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5138	0.5233	0.5282	0.5290	0.5703	0.7075
	1	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.5127**	**0.5221**	**0.5272**	**0.5267**	**0.5671**	0.7018
0.02					0.5128	0.5222	**0.5272**	**0.5267**	**0.5671**	0.7017
0.5					0.5153	0.5244	0.5296	0.5285	0.5677	0.7006
1.0					0.5162	0.5253	0.5305	0.5288	0.5675	**0.6992**
	2	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5123	0.5130	0.5236	0.5597	0.5873	0.7371
	2	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5105	0.5108	0.5215	0.5568	0.5838	0.7323
0.02					**0.5096**	**0.5098**	**0.5203**	0.5551	0.5818	0.7282
0.5					0.5114	0.5205	0.5275	0.5269	0.5636	0.7080
1.0					0.5105	0.5196	0.5266	**0.5249**	**0.5612**	0.7051
1.2					0.5107	0.5198	0.5266	**0.5249**	**0.5612**	**0.7048**
	3	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5284	0.5288	0.5393	0.5726	0.6001	0.7475
	3	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5291	0.5293	0.5399	0.5723	0.5978	0.7395
0.02					0.5278	**0.5281**	**0.5388**	0.5715	0.5972	0.7375
0.5					0.5306	0.5397	0.5466	0.5473	0.5857	0.7283
1.0					**0.5264**	0.5358	0.5423	**0.5420**	**0.5796**	0.7190
1.2					0.5277	0.5372	0.5434	0.5427	0.5803	**0.7179**
	5	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.5467**	**0.5507**	**0.5587**	0.5789	**0.5897**	**0.7330**
	5	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5609	0.5648	0.5719	0.5737	0.6291	0.7744
0.02					0.5559	0.5601	0.5672	**0.5695**	0.6266	0.7762
0.5					0.5573	0.5611	0.5721	0.5759	0.6334	0.7802
1.0					0.5648	0.5725	0.5805	0.5840	0.6244	0.7620
1.2					0.5627	0.5701	0.5784	0.5821	0.6228	0.7626

Table 5.4: The test losses from the deeper bi-LSTM models for the SARIMA data.

The Electricity data set has bigger and more significant forecasting improvements with deeper model architecture than for the SARIMA data. Table 5.5 is showing that the forecasting performance improves for all depths of the bi-LSTM block when the imputation task is introduced. The expansion to two hidden layers is showing that the forecasting task benefits from the imputation task also for lower test missing ratios compared with the findings from the base model. The overall performance of the forecasting model with two hidden layers is better than for the base model containing only one hidden layer. By expanding the model to consist of three hidden layers, the forecasting performance further improves compared to the two-layered model and is the best-performing model for the electricity data in this experiment. The weight of the imputation loss performing best for the electricity data set in this architecture is $\alpha = 0.5$. The model with 5 hidden LSTM layers has overall worse performance compared to the shallower models.

To summarize the effect of deeper models, both data sets needed more capacity to benefit from the imputation task, especially for the lower ratios of missing data. The best deepness for the electricity data was three hidden layers, while for SARIMA, the model with two hidden LSTM layers scored best. The fact that the electricity data needed more capacity than the SARIMA data makes sense since the electricity data set is much bigger and more complex being from the real world,

Alpha (weight of imputation loss)	Bi-LSTM layers	Relu	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0948	3.9040	8.1008	14.9697	27.7122	56.4444
	1	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**1.0487**	**1.1066**	**1.1992**	**1.4148**	2.0929	9.5105
0.02					1.0785	1.1330	1.2225	1.4187	**2.0177**	9.0475
0.3					1.1389	1.1978	1.2974	1.5060	2.1250	6.7248
0.5					1.1739	1.2306	1.3287	1.5311	2.1003	**6.7117**
1.0					1.1941	1.2512	1.3408	1.5423	2.0881	6.9421
1.2					1.1943	1.2483	1.3341	1.5298	2.0762	6.7869
	2	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.9173**	2.4841	4.8049	8.4069	15.4964,	35.6437
	2	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0667	1.1141	1.1837	1.3464	1.8619	6.7990
0.02					1.0573	1.0976	1.1637	1.3037	1.7229	6.1126
0.10					0.9912	1.0316	1.0943	1.2472	1.6872	6.9211
0.3					0.9670	**1.0092**	**1.0746**	**1.2288**	1.6693	7.0290
0.5					1.0442	1.0831	1.1463	1.2616	1.6203	**5.8342**
1.0					0.9851	1.0261	1.0896	1.2312	**1.6196**	6.2019
1.2					1.0012	1.0433	1.1123	1.2626	1.6653	6.0116
	3	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.9915	2.3730	4.3879	7.8836	14.4134	34.5529
	3	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0365	1.0774	1.1403	1.2885	1.7470	6.7621
0.02					0.9350	0.9736	1.0422	1.1842	1.5962	5.9331
0.1					0.9269	0.9647	1.0243	1.1649	1.5664	6.3320
0.5					**0.8766**	**0.9110**	**0.9635**	**1.0912**	**1.4488**	6.0827
0.7					0.9445	0.9753	1.0266	1.1447	1.4749	**5.1761**
1.0					0.9167	0.9506	1.0060	1.1304	1.4932	5.2851
	5	No	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.1137	2.4062	4.5246	8.0138	14.1822	33.8879
	5	No	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0282	1.0664	1.1326	1.2729	1.7801	7.50434
0.02					0.9780	1.0171	1.0849	1.2361	1.7338	7.5986
0.5					**0.9292**	**0.9645**	**1.0206**	**1.1504**	**1.5229**	**7.0362**
1.0					0.9653	1.0019	1.0557	1.1860	1.5584	7.6966

Table 5.5: The test losses from the deeper bi-LSTM models for the Electricity data.

compared to the small, synthetic and regular SARIMA data set. A final observation is that the higher ratios of missing values in many of the experiments benefit from a higher weighting of the imputation task compared to the lower missing ratios.

5.3.2 Wider model

Now the other variation of increased capacity is explored, namely expanding the width of the hidden layer. The only experiment in this category is a doubling of the hidden layer from the base model from 100 to 200.

For the SARIMA data set, this architecture still makes the model more robust when introducing missing values during training. However, as seen in Table 5.6, the forecast performance significantly worsens when the imputation task is added. Most likely the model overfits the small training data in the SARIMA data case.

Table 5.7 is showing that the wider model performs better on the electricity data. For the test data containing 50% missing data, the wide model with $\alpha = 0.5$ is actually giving the overall best performance across the explored models for this missing ratio. However, for the remaining missing ratios, the wide model improves

a little for $\alpha = 0.02$, however not as significantly as for the deeper model added imputation loss.

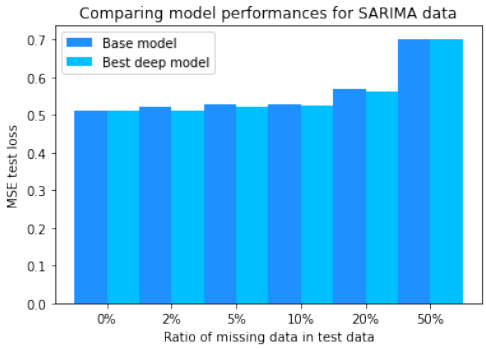
Alpha (weight of imputation loss)	Bi-LSTM layers	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	200	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				0.5153	0.5166	0.5262	0.5622	0.5921	0.7375
	1	200	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				**0.5143**	**0.5159**	**0.5247**	**0.5599**	**0.5884**	**0.7211**
0.02				0.5147	0.5163	0.5251	0.5604	0.5889	0.7217
0.5				0.5261	0.5283	0.5373	0.5745	0.6041	0.7419
1.0				0.5334	0.5360	0.5455	0.5837	0.6136	0.7530

Table 5.6: Wider model results for the sarima data set.

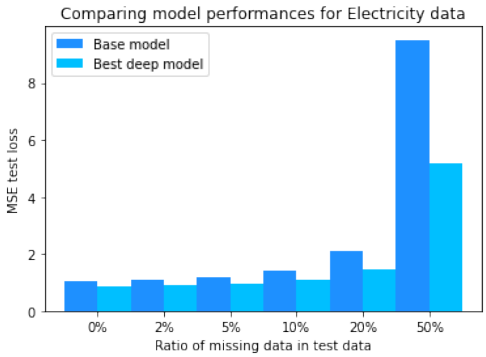
Alpha (weight of imputation loss)	Bi-LSTM layers	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	200	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				**0.9222**	4.0276	8.6972	15.9872	29.1945	56.2056
	1	200	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0				1.0691	1.1169	1.1977	1.3489	1.7870	6.0463
0.02				1.0639	**1.0991**	**1.1641**	**1.2855**	**1.6796**	5.9834
0.5				1.1163	1.1690	1.2587	1.4173	1.8602	**5.0121**

Table 5.7: Wider model results for the electricity data set.

The barplot in Figure 5.1 is showing the improvements from the base model with one hidden layer and no imputation loss (left bar) to the best performance from the models with increased capacity (right bar) for each missing ratio. All the best models with increased capacity utilize the imputation task. The SARIMA data are showing little or no improvement, however, for the electricity data, the imputation task is improving the performance significantly for all missing ratios. This indicates that the tasks are related and can benefit from each other and improve the forecasting.



(a) SARIMA data



(b) Electricity data

Figure 5.1: Barplot comparing the performance of the forecasting models. For each missing ratio in test data, the bar to the left is showing the performance with no imputation loss. The middle bar is showing the best performance of the deeper models, and the right bar is showing the best performance when ReLU is added.

5.4 Increased Non-linearity: ReLU

The last hypothesis explored is that the forecasting task with the auxiliary imputation task is highly complex and non-linear such that the non-linear activation functions *tanh* and *sigmoid* internally in the bi-LSTM block is not sufficient to capture the relationship. ReLU layer between the bi-LSTM output and the dense linear layer is added to test the hypothesis. The models extended with the additional ReLU layer are the models containing $1, 2$ and 3 hidden layers of size 100.

One might think that adding a non-linear layer only is transforming the already non-linear representations and that this would be unnecessary and perhaps not make sense. However, the additional layer seems to affect the forecasting performance.

All results can be found in Table 5.8 for SARIMA data and Table 5.9 for the Electricity data. For the SARIMA data, it is observable that the extra non-linear

layer is helpful for the model with two hidden layers, which we found to be best for this data before. For the three highest missing ratios the forecasting performances significantly improve from $\alpha = 0$ to $\alpha = 1.0$ and also slightly beats the performance from the model with two layers without the extra ReLU layer.

Table 5.9 is showing that the additional non-linear layer is also having a positive impact on the electricity data for benefiting from the imputation. The first interesting observation from Table 5.9 is that the ReLU layer is significantly improving the base model with one hidden layer when $\alpha = 0$, indicating the ReLU layer makes the base model more robust against missing values. This architecture only benefits from the imputation task when there are 10% or more missing values. The architecture benefiting from the imputation task for all missing ratios is again the one with three hidden LSTM layers for the electricity data. However, the performance does not beat the corresponding model without the ReLU layer for the best α.

Alpha (weight of imputation loss)	Bi-LSTM layers	Relu	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5172	**0.5250**	**0.5313**	0.5307	0.5684	0.7124
	1	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.5170**	**0.5250**	0.5315	**0.5300**	**0.5669**	0.7090
0.02					0.5174	0.5253	0.5319	0.5304	0.5671	**0.7086**
0.5					0.5186	0.5268	0.5333	0.5315	0.5688	0.7116
1.0					0.5211	0.5294	0.5359	0.5333	0.5707	0.7122
	2	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.5128**	**0.5106**	**0.5228**	0.5501	0.5734	0.7202
	2	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5153	0.5130	0.5250	0.5502	0.5713	0.7178
0.02					0.5132	0.5114	0.5233	0.5501	0.5728	0.7183
0.5					0.5129	0.5198	0.5275	0.5247	0.5593	0.7125
1.0					0.5134	0.5202	0.5279	**0.5235**	**0.5570**	0.7088
1.2					0.5134	0.5204	0.5279	0.5242	0.5581	0.7099
1.5					0.5148	0.5216	0.5291	0.5242	0.5574	**0.7067**
	3	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5297	**0.5281**	**0.5403**	0.5669	0.5910	0.7451
	3	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.5333	0.5318	0.5441	0.5707	0.5930	0.7490
0.02					0.5327	0.5313	0.5438	0.5702	0.5932	0.7489
0.5					0.5269	0.5330	0.5408	0.5383	0.5734	0.7227
0.7					**0.5265**	0.5327	0.5404	**0.5376**	**0.5727**	**0.7218**
1.0					0.5269	0.5334	0.5414	0.5382	0.5734	**0.7218**

Table 5.8: Results for the sarima data when a ReLU is added between the last hidden layer of the bi-LSTM and the linear output layer.

Alpha (weight of imputation loss)	Bi-LSTM layers	Relu	hidden_size	%missing in training data	Test loss forecasting (MSE)					
	1	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.8363	2.7714	5.5988	10.2466	19.0253	41.6118
	1	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					**0.8230**	**0.8580**	**0.9151**	1.0482	1.4122	5.2965
0.02					0.9627	1.0121	1.0928	1.2270	1.6058	5.4793
0.5					0.8772	0.9106	0.9694	1.0918	1.4242	**4.8301**
1.0					0.8324	0.8694	0.9250	**1.0472**	**1.3955**	5.7361
	2	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.1579	2.6646	4.9460	8.3416,	15.5222	36.6040
	2	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0384	1.0868	1.1690	1.3460	1.8993	7.6438
0.02					1.0458	1.0872	1.1605	1.3188	1.8181	7.5278
0.3					**0.9745**	1.0156	1.0821	1.2219	1.6435	8.4865
0.4					**0.9745**	**1.0124**	**1.0724**	**1.2037**	**1.5812**	6.7925
0.5					0.9994	1.0397	1.1005	1.2321	1.5941	**6.0525**
1.0					1.0124	1.0483	1.1075	1.2369	1.6119	6.2905
	3	Yes	100	0%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					0.9384	2.1442	4.0487	7.2348	13.4826	32.9579
	3	Yes	100	2%	0 %missing in test data	2 %missing in test data	5 %missing in test data	10 %missing in test data	20%missing in test data	50 %missing in test data
0					1.0009	1.0401	1.1077	1.2562	1.6917	7.2984
0.02					0.9059	0.9423	1.0063	1.1451	1.5314	5.7842
0.5					0.9245	0.9585	1.0098	1.1257	**1.4440**	**4.6647**
1					**0.8861**	**0.9205**	**0.9698**	**1.0966**	1.4462	5.6243
1.2					0.9097	0.9452	0.9990	1.1331	1.4821	5.8007

Table 5.9: Results for the electricity data when a ReLU is added between the last hidden layer of the bi-LSTM and the linear output layer.

5.5 Compare performance with other baselines

The focus in this thesis has not been to find the best model for forecasting the time series, but rather how the imputation task improves the forecasting task in a simple neural network model. In addition, we have looked at the model's robustness when there is missing data in the time series. To get an insight into how bad or good the model performance is, a comparison against two local-based baseline models is given. The methods considered are Naïve forecasts and SARIMA forecasts. MCAR missing values are simulated in the same way as earlier, however, now the missing values are filled using LOCF imputation. The data set considered for the comparisons is the SARIMA data set.

The first baseline model is the Naïve forecasting method where the forecasted values are the last observed value. The test losses from the Naïve approach are found in Table 5.10. The test losses in this table are unscaled, hence we back transform the predicted values from the bi-LSTM model for the best model for this data set found earlier in Table 5.11. If we compare these two models, it is observable that for the three lowest ratios of missing values, the thesis model outperforms the naive method. However, this is not the case for the three higher missing ratios where the naive forecasts outperform the thesis model. A possible reason for this is that since the test data set is only containing 5 values from each time series and that the data set used for the naive approach have imputed values, there are limits for how bad it can get when the forecast are based on the last value of the training data and five values ahead in time. For the thesis model, the values

are not imputed, meaning all five values in a time series could be missing (set to zero when scaled), and then the model is only leaning on the missing value indicator to forecast the values.

Amount of missing data						
	0%	2%	5%	10%	20%	50%
MSE	541214	541122	541156	478768	510733	444043

Table 5.10: MSE test losses from local Naive forecasting models on the SARIMA time series with LOCF imputed values. The losses are the total MSE loss across the time series.

The local SARIMA model beats the thesis model for all missing ratios as seen in Table 5.12. However, the way these models are fitted has an advantage compared to the thesis model. First of all, we use a function that automatically finds the best-suited model for each individual time series. Secondly, the SARIMA models get fitted to all values except for the value to forecast next.

By focusing on the three lower ratios of missing data, the thesis model performs better than the naive forecast but worse than local SARIMA models. An advantage of the thesis model over the SARIMA approach is that it is global, meaning there is only one model forecasting all the different time series. In this way, it is much faster than finding local SARIMA models for all time series in the data set individually.

Amount of missing data						
	0%	2%	5%	10%	20%	50%
MSE	193530	195273	191427	204116	203803	278467

Table 5.11: MSE test losses from local SARIMA models on the SARIMA time series with LOCF imputed values. The losses are the total MSE loss across the time series.

Amount of missing data in test						
	0%	2%	5%	10%	20%	50%
$\alpha = 0$	364301	367925	396055	577013	742956	1401883
$\alpha = 0.02$	354741	358363	386743	570079	735687	1380210
$\alpha = 1.2$	347963	358146	425437	386720	471059	989168

Table 5.12: Back-transformed test losses for the best model for the SARIMA data set with two hidden layers of size 100 in the LSTM-block.

Chapter 6

Conclusion

In this thesis, the relationship between forecasting and imputation of univariate time series with missing data has been explored. A robust forecasting model has been added an imputation task, trained in a multi-task manner with a weighted combination of the losses of the two tasks. The benefit of this approach is explored. How the ratio of missing values affects the utilization of the auxiliary task and the robustness of the model are also investigated. In addition, the research questions encouraged to explore whether the model architecture affects the answers of these questions.

We found that the forecasting task does benefit from the imputation task. The benefit of the imputation task is most significant for higher ratios of missing data, however, the improvements become significant for lower ratios of missing data when the capacity of the neural networks are increased from 1 hidden layer to 2 and 3 hidden layers for the SARIMA and the Electricity data, respectively. For the SARIMA data, the best deep model utilized the additional task best when the imputation loss weight was small ($\alpha = 0.02$) for lower missing ratios, and for the higher ratios, the model trained with $\alpha = 1.2$ utilized the additional task best. For the Electricity data, the best deep model improved the forecast performance for all missing ratios with $\alpha = 0.5$. Another advantage with the deeper models is that the robustness against an increasing amount of missing values for the single-task model ($\alpha = 0$) increases. The robustness of the base model with one hidden layer of size 100 became significantly improved by adding a ReLU layer between the last hidden layer and the forecasting linear output layer. With these findings, we can conclude with the two tasks forecasting and imputation being related and can utilize information from each other.

6.1 Future work

Based on the findings in this thesis, it would be natural to further investigate forecasting models for univariate time series with missing data utilizing information from imputation tasks. The model should be tested on several datasets with other

characteristics, and compared to other baselines. The results for the electricity data should also be compared against other baseline models that I, unfortunately, did not have the time to.

Replacing the bi-LSTM block with other standard neural network architectures to find the best choice for this problem should be done to find a baseline forecasting model robust against missing values or other types of noise on univariate time series. The relation between forecasting and imputation should also be further worked with due to the potential it has shown in this thesis. Perhaps the type of missing data affects the benefit of the missing value indicator and the imputation task, and the model should thus be explored with MAR and MNAR missing values to find out.

Bibliography

[1] D. Montgomery, C. Jennings and M. Kulahci, *Introduction to Time Series Analysis and Forecasting*, ser. Wiley Series in Probability and Statistics. Wiley, 2011, ISBN: 9781118211502. [Online]. Available: https://books.google.no/books?id=-qaFi0oOPAYC.

[2] S.-F. Wu, C.-Y. Chang and S.-J. Lee, 'Time series forecasting with missing values,' in *2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, 2015, pp. 151–156. DOI: 10.4108/icst.iniscom.2015.258269.

[3] R. Shumway and D. Stoffer, *Time Series Analysis and Its Applications: With R Examples*, ser. Springer Texts in Statistics. Springer New York, 2010, ISBN: 9781441978646. [Online]. Available: https://books.google.no/books?id=dbS5IQ8P5gYC.

[4] S. Makridakis and M. Hibon, 'Accuracy of forecasting: An empirical investigation,' *Journal of the Royal Statistical Society: Series A (General)*, vol. 142, no. 2, pp. 97–125, 1979. DOI: https://doi.org/10.2307/2345077. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.2307/2345077. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.2307/2345077.

[5] Wikipedia, *Makridakis Competitions — Wikipedia, the free encyclopedia*, http://en.wikipedia.org/w/index.php?title=Makridakis%20Competitions&oldid=994875525, [Online; accessed 10-May-2021], 2021.

[6] S. Makridakis, E. Spiliotis and V. Assimakopoulos, 'The m5 accuracy competition: Results, findings and conclusions,' Oct. 2020.

[7] R. Hyndman and G. Athanasopoulos, *Forecasting: Principles and Practice*, English, 2nd. Australia: OTexts, 2018.

[8] S. Palachy, *Stationarity in time series analysis*, Sep. 2019. [Online]. Available: https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322.

[9] A. L. Samuel, 'Some studies in machine learning using the game of checkers,' *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959. DOI: 10.1147/rd.33.0210.

[10] Y. Bengio, 'Learning deep architectures for ai,' *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009, ISSN: 1935-8237. DOI: 10.1561/2200000006. [Online]. Available: https://doi.org/10.1561/2200000006.

[11] W. Mcculloch and W. Pitts, 'A logical calculus of ideas immanent in nervous activity,' *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[12] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in computational intelligence. Berlin: Springer, 2012. DOI: 10.1007/978-3-642-24797-2. [Online]. Available: https://cds.cern.ch/record/1503877.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[14] C. Nicholson, *A beginner's guide to lstms and recurrent neural networks*. [Online]. Available: https://wiki.pathmind.com/lstm.

[15] J. J. Hopfield, 'Neural networks and physical systems with emergent collective computational abilities,' *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982, ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. eprint: https://www.pnas.org/content/79/8/2554.full.pdf. [Online]. Available: https://www.pnas.org/content/79/8/2554.

[16] Y. Bengio, P. Simard and P. Frasconi, 'Learning long-term dependencies with gradient descent is difficult,' *IEEE Transactions on Neural Networks*, vol. 5, 1994.

[17] B. Or, *The exploding and vanishing gradients problem in time series*, Oct. 2020. [Online]. Available: https://towardsdatascience.com/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22.

[18] S. Hochreiter and J. Schmidhuber, 'Long short-term memory,' *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[19] A. Holzner, *Lstm cells in pytorch*, Oct. 2017. [Online]. Available: https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c.

[20] *Lstm¶*, 2019. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html.

[21] J. Brownlee, *How to develop a bidirectional lstm for sequence classification in python with keras*, Jan. 2021. [Online]. Available: https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/.

[22] J. Brownlee, *Time series forecasting as supervised learning*, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/time-series-forecasting-supervised-learning/.

[23] R. Parmar, *Common loss functions in machine learning*, Sep. 2018. [Online]. Available: https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

[24] N. Donges, *A guide to rnn: Understanding recurrent neural networks and lstm*, Jun. 2019. [Online]. Available: https://builtin.com/data-science/recurrent-neural-networks-and-lstm.

[25] J. Brownlee, *Use early stopping to halt the training of neural networks at the right time*, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/.

[26] C. Lemaréchal, *Cauchy and the gradient method*, 2012.

[27] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[28] J. C. Duchi, E. Hazan and Y. Singer, 'Adaptive subgradient methods for online learning and stochastic optimization.,' *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/jmlr/jmlr12.html#DuchiHS11.

[29] J. Brownlee, *Gentle introduction to the adam optimization algorithm for deep learning*, Jan. 2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[30] D. B. RUBIN, 'Inference and missing data,' *Biometrika*, vol. 63, no. 3, pp. 581–592, Dec. 1976, ISSN: 0006-3444. DOI: 10.1093/biomet/63.3.581. eprint: https://academic.oup.com/biomet/article-pdf/63/3/581/756166/63-3-581.pdf. [Online]. Available: https://doi.org/10.1093/biomet/63.3.581.

[31] J. Du, M. Hu and W. Zhang, 'Missing data problem in the monitoring system: A review,' *IEEE Sensors Journal*, vol. PP, pp. 1–1, Jul. 2020. DOI: 10.1109/JSEN.2020.3009265.

[32] S. Buuren, *Flexible Imputation of Missing Data, Second Edition*. Jul. 2018, ISBN: 9780429492259. DOI: 10.1201/9780429492259.

[33] G. Dumedah and P. Coulibaly, 'Evaluation of statistical methods for infilling missing values in high-resolution soil moisture data,' English, *Journal of Hydrology*, vol. 400, no. 1-2, pp. 95–102, 2011, ISSN: 0022-1694. DOI: 10.1016/j.jhydrol.2011.01.028.

[34] I. Pratama, A. Permanasari, I. Ardiyanto and R. Indrayani, 'A review of missing values handling methods on time-series data,' Oct. 2016, pp. 1–6. DOI: 10.1109/ICITSI.2016.7858189.

[35] D. B. Rubin, *Multiple Imputation for Nonresponse in Surveys*. Wiley, 1987, p. 258.

[36] Y. Bengio, A. Courville and P. Vincent, *Representation learning: A review and new perspectives*, 2014. arXiv: 1206.5538 [cs.LG].

[37] Ö. Gür Ali and K. Yaman, 'Selecting rows and columns for training support vector regression models with large retail datasets,' *European Journal of Operational Research*, vol. 226, no. 3, pp. 471–480, 2013. DOI: 10.1016/j.ejor.2012.11.01. [Online]. Available: https://ideas.repec.org/a/eee/ejores/v226y2013i3p471-480.html.

[38] P. Montero-Manso and R. J. Hyndman, *Principles and algorithms for forecasting groups of time series: Locality and globality*, 2020. arXiv: 2008.00444 [cs.LG].

[39] S. Makridakis, E. Spiliotis and V. Assimakopoulos, 'The m4 competition: 100,000 time series and 61 forecasting methods,' *International Journal of Forecasting*, vol. 36, no. 1, pp. 54–74, 2020, M4 Competition, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2019.04.014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207019301128.

[40] A.-A. Semenoglou, E. Spiliotis, S. Makridakis and V. Assimakopoulos, 'Investigating the accuracy of cross-learning time series forecasting methods,' *International Journal of Forecasting*, 2020, ISSN: 0169-2070. DOI: https://doi.org/10.1016/j.ijforecast.2020.11.009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207020301850.

[41] R. Vilalta, C. Giraud-Carrier, P. Brazdil and C. Soares, 'Inductive transfer,' in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, pp. 545–548, ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_401. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_401.

[42] R. Caruana, 'Multitask learning,' in *Learning to Learn*, S. Thrun and L. Pratt, Eds. Boston, MA: Springer US, 1998, pp. 95–133, ISBN: 978-1-4615-5529-2. DOI: 10.1007/978-1-4615-5529-2_5. [Online]. Available: https://doi.org/10.1007/978-1-4615-5529-2_5.

[43] R. Caruana, 'Multitask learning,' *Journal of Complexity*, vol. 28, no. 2, 1997. [Online]. Available: https://doi.org/10.1023/A:1007379606734.

[44] P. Vafaeikia, K. Namdar and F. Khalvati, *A brief review of deep multi-task learning and auxiliary task learning*, 2020. arXiv: 2007.01126 [cs.LG].

[45] M. Crawshaw, *Multi-task learning with deep neural networks: A survey*, 2020. arXiv: 2009.09796 [cs.LG].

[46] Y. S. Abu-Mostafa, 'Learning from hints in neural networks,' *Journal of Complexity*, vol. 6, no. 2, pp. 192–198, 1990, ISSN: 0885-064X. DOI: https://doi.org/10.1016/0885-064X(90)90006-Y. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0885064X9090006Y.

[47] N. Y. Peng, *N-beats-beating statistical models with neural nets*, Jun. 2020. [Online]. Available: https://towardsdatascience.com/n-beats-beating-statistical-models-with-neural-nets-28a4ba4a4de8.

[48] S. Smyl, *M4 forecasting competition: Introducing a new hybrid es-rnn model*, Dec. 2018. [Online]. Available: https://eng.uber.com/m4-forecasting-competition/.

[49] B. N. Oreshkin, D. Carpov, N. Chapados and Y. Bengio, 'N-BEATS: neural basis expansion analysis for interpretable time series forecasting,' *CoRR*, vol. abs/1905.10437, 2019. arXiv: 1905.10437. [Online]. Available: http://arxiv.org/abs/1905.10437.

[50] R. Collobert and J. Weston, 'A unified architecture for natural language processing: Deep neural networks with multitask learning,' in *Proceedings of the 25th International Conference on Machine Learning*, ser. ICML '08, Helsinki, Finland: Association for Computing Machinery, 2008, pp. 160–167, ISBN: 9781605582054. DOI: 10.1145/1390156.1390177. [Online]. Available: https://doi.org/10.1145/1390156.1390177.

[51] R. Girshick, 'Fast r-cnn,' in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[52] W. Cao, D. Wang, J. Li, H. Zhou, L. Li and Y. Li, 'BRITS: bidirectional recurrent imputation for time series,' *CoRR*, vol. abs/1805.10572, 2018. arXiv: 1805.10572. [Online]. Available: http://arxiv.org/abs/1805.10572.

[53] Z. Che, S. Purushotham, K. Cho, D. A. Sontag and Y. Liu, 'Recurrent neural networks for multivariate time series with missing values,' *CoRR*, vol. abs/1606.01865, 2016. arXiv: 1606.01865. [Online]. Available: http://arxiv.org/abs/1606.01865.

[54] Y. Luo, X. Cai, Y. ZHANG, J. Xu and Y. xiaojie, 'Multivariate time series imputation with generative adversarial networks,' in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/96b9bff013acedfb1d140579e2fbeb63-Paper.pdf.

[55] S. van Buuren and K. Groothuis-Oudshoorn, 'mice: Multivariate imputation by chained equations in r,' *Journal of Statistical Software*, vol. 45, no. 3, pp. 1–67, 2011. [Online]. Available: https://www.jstatsoft.org/v45/i03/.

[56] S. Mouselinos, K. Polymenakos, A. Nikitakis and K. Kyriakopoulos, *Main: Multihead-attention imputation networks*, 2021. arXiv: 2102.05428 [cs.LG].

[57] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013. [Online]. Available: http://www.R-project.org/.

[58] I. Svetunkov, *Smooth: Forecasting using state space models*, R package version 3.1.1, 2021. [Online]. Available: https://CRAN.R-project.org/package=smooth.

[59] J. Brownlee, *How to use data scaling improve deep learning model stability and performance*, Aug. 2020. [Online]. Available: https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/.

[60] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, 'Scikit-learn: Machine learning in Python,' *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, 'Pytorch: An imperative style, high-performance deep learning library,' in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Eds., vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[62] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[63] e. a. Falcon WA, 'Pytorch lightning,' *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.

[64] T. G. Smith *et al.*, *pmdarima: Arima estimators for Python*, [Online; accessed <today>], 2017–. [Online]. Available: http://www.alkaline-ml.com/pmdarima.

NTNU

Norwegian University of
Science and Technology