Mathias Opland

Forecast Uncertainty for Univariate Time Series Using Generative Adversarial Networks

Master's thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
June 2020

Master's thesis

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

Mathias Opland

Forecast Uncertainty for Univariate Time Series Using Generative Adversarial Networks

Master's thesis in Applied Physics and Mathematics
Supervisor: Erlend Aune
June 2020

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

Summary

The forecast uncertainty is an important aspect of assessing the quality of a forecast. Recent forecasting competitions have shown the power of deep neural networks in time series forecasting, however, for a standard neural network, the forecast uncertainty is not a mathematically grounded statistic. Concurrently, generative adversarial networks have proved to be a powerful tool for generating realistic images. This thesis aims to show how generative adversarial networks can be used to estimate the forecast uncertainty, and compare the results to well-known baseline models and a state-of-the-art method for estimating forecast uncertainty with neural networks. Further, the thesis also aims to investigate how forecasting multiple steps ahead affect the performance of the uncertainty estimates and the forecast accuracy.

Inspired by recent research in the field, a conditional generative adversarial network for forecasting is presented, namely ForGAN. We first investigate the ability of a GAN to estimate simple distributions without temporal dependencies, and thereafter explore how some key hyperparameters affect the performance of the distribution estimation. Further, the ForGAN is compared to the baseline and state-of-the-art models across one synthetic and three real time series data sets, forecasting multiple steps ahead. Prediction intervals are used to measure the quality of the uncertainty estimates, where the coverage and the mean scaled interval score (MSIS) is used as a measure of the performance. The multi-step performance is investigated through the coverage of the prediction intervals over the forecast horizon, using a recursive strategy to forecast multiple steps ahead.

The results show that the generative adversarial network is able to estimate the forecast uncertainty comparable to the baseline models and the state-of-the-art model. For two of the three real time series, the ForGAN scores best in terms of MSIS. Although some problematic behavior occurs, the ForGAN model shows promising results. Investigating the performance, some ideas for further research arises, in order to improve the performance of the ForGAN.

Sammendrag

Når man skal predikere fremtidige verdier av en tidsrekke er usikkerheten i prediksjonene en viktig faktor. Nylig har konkurranser innen tidsrekke-prediksjon vist at nevrale nettverk presterer svært godt, men usikkerhetsmålet for disse er ikke matematisk basert. Samtidig har generative adverseriale nettverk (GAN) vist seg å være et nyttig verktøy for å generere realistiske bilder. Denne masteroppgaven vil undersøke hvordan GAN kan brukes til å estimere den nevnte prediksjonsusikkerheten. Resultatene sammenlignes så med kjente statistiske modeller og moderne metoder for å estimere prediksjonsusikkerheten i nevrale nettverk. Videre vil vi vise hvordan prediksjon flere steg frem i tid påvirker usikkerhetsestimatene og prediksjonsnøyaktigheten.

Vi presenterer et betinget generativt adverserialt nettverk for tidsrekkeprediksjon som er inspirert av andre studier på området, referert til som ForGAN. Først undersøker vi GAN'en sin evne til å estimere enkle sannsynlighetsfordelinger, deretter analysere hvordan noen av de viktigste parameterne påvirker resultatene. Videre sammenligner vi ForGAN modellen med de statistiske metodene og det moderne nevrale nettverket på én syntetisk tidsrekke og tre ekte tidsrekke-datasett, hvor vi predikerer flere steg frem i tid. Prediksjonsintervaller blir brukt for å representere usikkerhetsestimatene, der dekningen og gjennomsnittlig skalert intervallverdi (MSIS) måler kvaliteten på prediksjonsintervallene. Masteroppgaven vil også ta for seg hvordan prediksjonshorisonten påvirker prediksjonsintervallenes dekningen, ved bruk av en rekursiv metode for å predikere flere steg fram i tid.

Resultatene viser at ForGAN'en klarer å estimere gode prediksjonsintervaller for prediksjonsusikkerheten sammenlignet med de statiske modellene og det moderne nevrale nettverket. ForGAN-modellen har også best MSIS for to av de tre ekte tidsrekke-datasettene. Selv om vi oppdager noe problematisk oppførsel, viser ForGAN modellen lovende resultater. Som følge av å ha undersøkt resultatene presenterer vi noen idéer for videre arbeid med å forbedre modellen.

Preface

This thesis finalizes my master's degree in Industrial Mathematics, as part of the study program Applied Physics and Mathematics M.Sc. at the Norwegian University of Science and Technology. The work continuous my specialization project, and has been conducted in the spring of 2020.

I would like to direct a huge thanks to my supervisor, Associate Professor Erlend Aune, for giving me the opportunity to explore topics that I find truly exciting. The ideas, guidance and knowledge he has shared throughout the last year have been invaluable.

This marks the end of five years in Trondheim, where I have had the opportunity to evolve, both academically and on a personal level. I would like to thank my friends, family and girlfriend for their support, and for making this the best experience of my life so far.

Trondheim, June 2020

Mathias Opland

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Finding a way of looking into the future has intrigued the human mind through centuries. The Babylonians tried to forecast the weather using cloud formations as early as 650 B.C (NASA Earth Observatory (2002)). Astrologers have looked to the stars for answers, whereas religious populations have looked for foreshadows through scriptures, prayers and revelations. As a greater understanding of nature and physics rose from the Renaissance and Scientific Revolution through the work of scientists such as Nicolaus Copernicus, Galileo Galilei and Isaac Newton, the predicting power increased substantially. The idea of statistical analysis of time series dates back to the introduction of modern statistics (Tsay (2000)), however, the usage did not really start until the 1950s, when computer modeling became established.

Today, forecasting is everywhere. We plan our lives around the weather forecast and expect it to be precise. The finance sector rely heavily on forecasting to predict stock prices, insurance profit and loan defaults. The tourism industry forecasts where you are going on vacation years from now and the electricity companies are forecasting future power consumption. Real estate developers are predicting which housing areas will increase in price in the next few years, and betting companies are forecasting the odds of different sports results. Hundreds of similar examples can be made, and illustrate how forecasting plays an important part in many industries. However, as the domain can vary greatly, the forecasting task is quite similar; forecast the next value(s) of a quantity given previously observed values and possible auxiliary information.

Forecasting is a difficult task, as the dynamics may vary from domain to domain and even between related time series. While one can obtain a long time series, the observations way back may not be as relevant for the succeeding time series. Therefore one might not have as many samples available as one has for other tasks, such as image recognition and text classification. We may also have external factors that affect the dynamics of the time series, however, it might be hard to know which ones. Including all relevant information may lead to high dimensional data, with a relatively limited number of samples, which can make it hard to distinguish temporal dynamics[1] from random noise. Learning across multiple time series can be a way to obtain more data, battling difficulties related to high dimensional data and further hoping that there is some common temporal dynamic that can be transferred between the time series. Despite the difficulties, researchers continue to invest time and resources into inventing and improving forecast models due to the benefits of improved forecasts.

A common aim of model development is to create the most accurate forecasting model, however a likewise important metric is the forecast uncertainty. In everyday life, humans rely not only on forecasts but also uncertainty estimates of those forecasts. For example, if one has plans that depend on avoiding rain, a weather forecast of sunny weather may help, but the more relevant information is the probability of downfall. Likewise, a forecast of a stock price increasing in value may be rendered useless if the uncertainty is large. Uncertainty estimates increase the insight and interpretability of the forecasts, which increases the usefulness and our chance of making good decisions based on forecasting.

Whereas statistical models have defined the forecast uncertainty through theoretical knowledge of the models, neural networks are considered ”black-box” models where no such statistic is theoretically defined. However, as deep learning has advanced in image classification, speech recognition and latest time series forecasting, as shown in the M4 competition (Makridakis et al. (2020)), a natural step forward is to find ways to estimate the forecast uncertainty. While the M4 competition appointed a winner for the best uncertainty estimation with regards to the 95% prediction interval, the latest edition, and now ongoing M5 competition[2], aims to compare the estimated uncertainty distribution using 50%, 67%, 95% and 99% prediction intervals. This shows how the forecasting community has recently come to emphasize uncertainty estimation.

1.1 Research Questions

Generative adversarial networks (GANs) have seen extensive use in the image generation task, due to the ability to generate realistic images[3]. Due to its success, the GAN framework has been adapted to other domains. In this thesis, we will use GAN to forecast both point forecasts and uncertainty estimates, and compare them to baseline models, as well as a state-of-the-art model to assess forecast uncertainty in neural networks. This will be done by first investigating the properties of the proposed model on two synthetic data sets, before comparing the forecast results to the baseline models on three real time series data sets. The data sets are chosen from different domains and with different seasonal frequency, to hold dissimilar dynamics.

The aim of this thesis is stated in the following research questions:

- Can generative adversarial networks be used to estimate forecast uncertainty?
- How well does the estimated uncertainty perform compared to theoretically grounded

[1]Temporal dynamics are dependencies and patterns in a time series.
[2]`https://mofc.unic.ac.cy/m5-competition`
[3]`https://thispersondoesnotexist.com`

uncertainties for statistical methods and forecast uncertainty obtained by state-of-the-art methods?

- How does the forecasting horizon affect the uncertainty estimates?

In order to investigate the research questions, we have to determine a scope for the thesis. To simplify the data processing, we will only investigate time series with on variable, namely univariate time series. Univariate time series also works well with regard to the recursive multi-step method chosen. Further, multivariate time series will make the training both more computational and time-demanding. We will also only investigate simple neural network architectures, with only one recurrent layer. More complex architectures may increase the performance, however, the models will also be more computational demanding and probably more data-hungry.

The thesis will include a literature review in Chapter 2, where we will investigate what has been done related to the topic of forecasting uncertainty in a neural network, and time series forecasting with generative adversarial networks. Further in Chapter 3 we will introduce the theory related to the models used to conduct the experiments, as well as discuss the performance metrics used to evaluate the models. In Chapter 4 we will introduce the experimental setup, the data sets and the specific model used to conduct the experiments. Chapter 5 will present results for each of the data sets, compare the performance of the different models and discuss the results obtained. Finally, in Chapter 6 we will provide a conclusion of the experiments and results, answer the research questions and propose further work related to the topic and results presented in this thesis.

Chapter 2

Literature Review

In this chapter we will investigate work related to forecasting and especially forecast uncertainty; both well-known statistical models and more recent models for estimating the forecast uncertainty with neural networks. The aim is to provide context to forecasting and forecast uncertainty estimation, and further investigate what has been done in order to estimate the forecast uncertainty with deep neural networks. Finally, we will present work related to time series modeling and forecasting with generative adversarial networks.

2.1 Forecast Uncertainty

Forecast uncertainty estimation is a subtask of forecasting, where the goal is to capture the error distribution of a future event. This error can be related to how well the model is able to capture the time series dynamics, but also to the amount of irreducible error in the data set. The irreducible error accounts for the noise not explained by the data, and as indicated by the name, cannot be reduced unless supplying additional data. On the other hand, the model uncertainty can be reduced by achieving more accurate models. Moreover, the goal of forecast uncertainty estimation is to correctly assess both the model uncertainty and the irreducible error.

2.1.1 Classical Models

As time series modeling and forecasting is a difficult task, less complex models have been regarded the best for decades (Makridakis and Hibon (2000)). It is only recently that more complicated models, such as neural networks, has shown superior performance to the older statistical models (Makridakis et al. (2020)). We will therefore present two statistical models for time series modeling and forecasting, which will later be used as baseline models in order to compare the performance of the proposed generative adversarial network.

Exponential Smoothing

Exponential smoothing was suggested in the late 1950s (Holt (2004), Winters (1960)) as a way of modeling and forecasting time series. The model bases its forecast on a weighted sum of past observations, where the weights are exponentially decaying, thereof the name. In addition, trend and seasonality can be added in order to model basic temporal dynamics. Despite its simplicity, the M3 competition (Makridakis and Hibon (2000)) identified a variant of exponential smoothing, dampen trend exponential smoothing, as one of the best performing forecasting models. Due to its good performance and longevity in the field of forecasting, we will use it as a baseline model. The variations of exponential smoothing will be discussed in section 3.2.

ARIMA

Autoregressive integrated moving average (ARIMA) (Brockwell and Davis (2016)) models have been a staple of time series modeling for decades. It is however somewhat more sophisticated than the aforementioned exponential smoothing. The autoregressive (AR) part of ARIMA models a value as a linear combination of prior values, the integrated (I) part removes trends and seasonality by differencing[1] the time series, and the moving average (MA) models the output as a linear combination of prior residuals. By combining these components, the ARIMA is able to model a wide variety of time series dynamics. In addition, one can add seasonal AR, I, and MA components, all of which will be explained in detail in section 3.3.

2.1.2 Deep Neural Networks

Deep learning and deep neural networks have shown remarkable results across various domains, including forecasting (Makridakis et al. (2020)). The introduction of recurrent neural networks (Rumelhart et al. (1986)) and later Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber (1997)) aimed to solve problems where there is some temporal dynamic between the inputs, for example natural language processing, speech recognition or time series modeling. The correlation between the past values is not established prior to fitting the model, making for a flexible model where the actual dynamics of the data can be captured despite prior beliefs made by humans or model limitations. However, as the model is more flexible, the optimization task becomes correspondingly harder. The model can overfit easily, and pick up random noise as temporal dynamics. Large amounts of data can be a cure, however, time series may not have that much correlated data. Despite these difficulties, it has been developed well-performing deep neural networks for time series forecasting, some of which will be presented below.

2.1.3 State of the Art

We will now introduce some state-of-the-art techniques, the MC dropout and the pinball loss, for estimating the forecast uncertainty with neural networks, where the former will be

[1] $y'_t = y_t - y_{t-1}$ is a differencing, where y'_t replaces y_t in the time series. More information on ARIMA and differencing can be found in section 3.3.

used as a comparison for the generative adversarial network.

Gal and Ghahramani (2015) suggested that the model uncertainty in a neural network could be modeled by a well-known regularization technique; dropout (N. Srivastava and Salakhutdinov (2014)). Usually, dropout is applied during training to regularize the model, however by introducing this stochastic behavior during testing, it approximated a Gaussian process over the weights. Thus by sampling from the neural network, one could sample the model uncertainty, a method which is referred to as Monte Carlo dropout (MC dropout). Zhu and Laptev (2017) built further on this uncertainty estimate by adding model misspecification through dropout in an encoder-decoder and estimating the inherent noise as the validation mean squared error to obtain the forecast uncertainty. We will explain this solution later in section 3.5 and use it as our state-of-the-art comparison model, where the implementation was carried out as a specialization project (Opland (2020)).

As the winning solution of the M4 competition (Makridakis et al. (2020)), Smyl and Pasqua (2018) suggests a hybrid model where an exponential smoothing is modeling the trend and seasonality, and a recurrent neural network is modeling the random component of the time series. To obtain accurate point forecasts, the Pinball loss (Steinwart and Christmann (2011)) is used to counter some positive bias. The prediction intervals are obtained by minimizing the mean scaled interval score (MSIS) (Gneiting and Raftery (2007)). However, as the MSIS is merely a way to score the performance of the prediction interval, which was used as the comparison metric in the M4 competition, it has some bias which we will discuss later in section 3.8.5.

2.2 Generative Adversarial Networks

Goodfellow et al. (2014) introduced a machine learning framework that has made a substantial impact on the machine learning field: generative adversarial networks (GAN). The goal is to model a generative network $\mathcal{G}$ to capture and generate samples from a data distribution. This is done by creating a min-max two-player game where a discriminative network $\mathcal{D}$ wants to label the generated samples as ”fake” and real samples as ”real”. On the other hand, the generator $\mathcal{G}$ aims to ”fool” the discriminator $\mathcal{D}$ to label the generated samples as ”real”. The idea is not radically new, Schmidhuber (2019) even claims that it is only a special case of artificial curiosity (Schmidhuber (1990), Schmidhuber (1991b)) and related to predictability minimization (Schmidhuber (1991a)), which was introduced three decades ago. Further work on the subject by Mirza and Osindero (2014) introduced the conditional GAN, cGAN, which is able to capture multimodal distributions better, and instrumental in generating samples with specific behavior. This could be generating images of handwritten numbers, where the condition may determine which number to be generated. Following these introductions, the use of GANs has skyrocketed. Revolutionary applications in image generation such as style transfer of images (Karras et al. (2019), Zhu et al. (2017)), deepfakes (Tolosana et al. (2020), Wang (2019)) and image-to-text generation (Gorti and Ma (2018)) has led to intensive publicity related to GANs.

2.2.1 Forecasting with Generative Adversarial Networks

Due to the obvious properties of generative adversarial networks, namely generating data, the GAN framework has been most utilized in the image and text generation context. However, there has been some effort lately in adopting the successful GAN framework to other tasks, among these time series forecasting.

Esteban et al. (2017) propose a recurrent conditional GAN architecture in order to generate synthetic data from a real time series data set. They show results where models trained on the synthetic generated data only have minor degradation in performance when tested on real time series. Brophy et al. (2019) further investigate the properties of time series generation through mapping real time series to grayscaled images, then using an image-based GAN to generate new images of time series, and mapping them back to synthetic time series. Yoon et al. (2019) introduce a more sophisticated architecture using a mixture of a supervised and adversarial framework for time series generation. This allows for a more accurate capture of the temporal dynamics that can be found in deterministic models, while still possessing the stochastic properties of generative models. Whereas these examples are not directly related to the forecasting task, they show that the generator has the ability to learn time series dynamics through the GAN framework.

Further, Husein et al. (2019) aim to forecast the drug sales for the following week using the GAN framework. The performance is measured in both point accuracy through MAE[2], RMSE[3] and MAPE[4], and classification error of either sales increasing or decreasing in relation to current level. While the paper claims that the model performs well in terms of the aforementioned metrics, the results are not compared to any baseline model performance or any known results on the data set used. It is therefore infeasible to assert the performance of the models used.

Zhou et al. (2018) forecast one-step-ahead stock prices on high-frequency stock market time series, using a conditional GAN framework where the conditional input is previously observed stock prices. The model is using an LSTM layer in the generator and convolutional layers in the discriminator, and they compare the performance to baseline models such as ARIMA-GARCH[5] and a non-adversarial version of the generator. They obtain better forecasting accuracy for their proposed GAN model than any of the baseline and comparison models. Work done by Zhang et al. (2019) show similar results on stock market data where they forecast the closing price of the next day using the previous five days. They also show that their proposed GAN model performs better than a standard LSTM network and a Support Vector Regression (SVR) on the data set.

2.2.2 Forecast uncertainty with Generative Adversarial Networks

While there has been work related to forecasting with GANs, as shown in the previous section, not many have focused on forecast uncertainty. However, we will present work

[2]Mean absolute scaled error, see section 3.8.3.

[3]Root mean squared error, see section 3.8.1.

[4]Mean absolute percentage error, see section 3.8.2.

[5]`https://www.mssanz.org.au/modsim2013/F2/yaziz.pdf`

by Fu et al. (2019), Koochali et al. (2019) and Koochali et al. (2020), whose work we will build on in this thesis.

Fu et al. (2019) simulate financial time series data with the use of a conditional GAN with feed-forward architecture and shows that it can generate predictive conditional distributions. While the examples are most related to point forecasting, they show the uncertainty estimation through Value-at-Risk (VaR) and Expected Shortfall (ES), which are two uncertainty measures especially related to financial and economic uncertainty estimates. VaR (Holton (2014)) estimates the risk of loss, and how much it potentially looses. ES (Acerbi and Tasche (2001)) is the average loss in the $q \cdot 100\%$ worst-case scenarios. Although these performance measures are not that relevant in the general time series forecasting case, the results still show the abilities of GANs. They restrict the architectures of the neural networks to feed forward connections, and suggest further work where deep convolutional or recurrent layers are utilized. They also use the Wasserstein loss (Frogner et al. (2015)), which has shown improved training stability and convergence in GANs (Arjovsky et al. (2017)). Both applying recurrent layers and using the Wasserstein loss is something we will investigate in this thesis.

Koochali et al. (2019) introduce ForGAN as a one-step-ahead probabilistic forecasting model. By utilizing a conditional GAN setup with previous values of the time series as input, they argue that the generator is able to model the full probability distribution of the forecast. The model employs a recurrent layer, either LSTM or GRU, in both the generator and the discriminator. Further, they use this model to forecast the one-step-ahead distribution on three different data sets, two synthetic and one real time series data set. They compare the results obtained to a G-regression, a non-adversarial trained generator, along with the state-of-the-art results on the respective data sets. The performance is measured in both point forecast accuracy and Kullback-Leibler divergence (section 3.8.6). While they report excellent performance in terms of Kullback-Leibler divergence, the state-of-the-art model does not provide this quantity, and the G-regression is not well suited for a probabilistic forecast. The G-regression has also higher point forecast accuracy for two of the three data sets, among them the real time series. Concluding the paper, Koochali et al. (2019) mention that forecasting multiple steps ahead and comparing it to state-of-the-art models is a way to further research GAN in the forecast setting. This thesis will further investigate the ForGAN framework suggested by Koochali et al. (2019), and further forecast multiple steps ahead with comparisons of the results to well-known statistical models described in section 2.1.1.

Koochali et al. (2020) provide a probabilistic conditional GAN model for multivariate time series forecasting. In addition, they propose a framework for transforming a deterministic forecast model into a probabilistic model and compare results on two real time series data sets. The performance is measured in the negative form of Continuous Ranked Probability Score (CRPS*) (Gneiting and Raftery (2007)), which can be interpreted to measure the sharpness and precision of the probabilistic forecast, however, reduces to the mean absolute error (MAE) for a deterministic forecast. Thus it is useful for comparing probabilistic and deterministic models. They show that the probabilistic conditional GAN performs better than a deterministic trained variant of the generator in terms of CRPS* for the one-step-ahead forecast.Likewise, in this thesis we will compare the ForGAN to a standard neural network, however, we will use the MC dropout (section 2.1.3) to obtain probabilistic forecasts with the neural network.

Chapter 3

Theory

In this section, the theory behind the methods used when running experiments will be introduced. This includes various forms of forecasting techniques such as exponential smoothing, ARIMA models, different neural network architectures and lastly the generative adversarial network. In addition, we assess how we can use these models to forecast, and how the forecast uncertainties are estimated. We derive the recursive multi-step method used for forecasting multiple steps ahead, and finally introduce the performance metrics used to compare the results. A lot of the theory presented here is included in Python packages, making a good foundation and lowering the probability of error. However, an understanding of the theory behind is crucial in order to develop the right models, interpret results and making necessary adjustments.

3.1 Forecasting

A time series $\{Y_t\}$ is a set of observations y_t, where t denote the specific time they were recorded (Brockwell and Davis (2016)). The time t does not need to correspond with a specific time format, but has to define the timely order of the data. It is also useful to know which time-frequency the observations y_t corresponds to, as this can be used to develop better models. An example will be observations of the temperature, where we would expect the temperature to correlate with the season. If the observations are daily, we know that the year consists of 365 days (or 366), and we would expect similar temperatures a year apart. Also, one often wants time series with the same time interval between each observation, so-called equally spaced. An equally spaced time series can have any given frequency, as long as it is consistent within the time series. In this thesis, we will look at equally spaced time series, some with monthly observation frequency, some with weekly observation frequency and a high-frequency time series with hourly observations.

A univariate time series is the most simple form of time series, where y_t only consists of one observation. This can be the weather temperature, stock prices, number of passengers on public transportation, demand for taxi transportation, electricity consumption, etc. Common for the univariate time series is that we only have past values of the given quantity, called the endogenous variable. On the contrary, a multivariate time series consists of not only a time dependent sequence of observations, but multiple time dependent components where there is some interdependence between the different components of the time series (Brockwell and Davis (2016)). These can be additional explanatory variables to the endogenous time series, named exogenous variables, or one model forecasting multiple time series of the same quantity simultaneously due to the interdependence. Examples of this can be forecasting the temperature over closely related areas simultaneously, or exogenous variables such as precipitation and ocean current, in order to improve the temperature forecast. In this thesis, we will focus on the univariate time series, where the models will base their predictions solely on past values of the quantity at hand.

The goal of time series modeling is to find the optimal function f, such that:

$$y_t = f(X_{t-1}) + \epsilon_t, \tag{3.1}$$

where X_{t-1} is any previously observed values of y and possible auxiliary information, and ϵ_t is the irreducible error at time t with mean 0 and finite variance σ^2_ϵ (Tsay (2000)). There are various ways to estimate the function f, which we will explore in this chapter.

3.1.1 Naive Forecast

The simplest form of forecasting can be achieved by simply guessing that the value will stay the same, so called last-day forecast:

$$\hat{y}_{t+1|t} = y_t. \tag{3.2}$$

Here $\hat{y}_{t+1|t}$ is the prediction of the value y_{t+1} at time t, called a one-step-ahead forecast. Forecasting multiple steps in the future, is referred to as an h-step-ahead forecast, and is denoted as $\hat{y}_{t+h|t}$. The last-day method can be extended to an h-step-ahead forecast:

$$\hat{y}_{t+h|t} = y_t. \tag{3.3}$$

3.1.2 Average Forecast

Another way to perform a simple forecast is by taking the average of past values. This can either be done by taking the average value of all previous values:

$$\hat{y}_{t+h|t} = (y_t + y_{t-1} + ... + y_1)/t, \tag{3.4}$$

or a rolling average given a window length ℓ:

$$\hat{y}_{t+h|t} = (y_t + y_{t-1} + ... + y_{t+1-\ell})/\ell. \tag{3.5}$$

3.1.3 Trend

Some usual characteristics of a time series are trend and seasons and are often modeled in order to obtain a more accurate forecast. Trend accounts for an increase or decrease over time, which we often want to model as a rather smooth function. This can be either linear, polynomial, exponential, or logarithmic, depending on the nature of the change. A time-independent trend is called drift, and we can add drift to the equation (3.2):

$$\hat{y}_{t+1|t} = c + y_t, \tag{3.6}$$

where c is the drift term. Further, we can expand the model to include time-dependent trend by:

$$\hat{y}_{t+1|t} = c + bt + y_t, \tag{3.7}$$

where bt is the trend term. Thus expanding this, the h-step-ahead forecast can then be given by:

$$\hat{y}_{t+h|t} = c + b(t + h - 1) + \hat{y}_{t+h-1|t} = ch + b\sum_{i=0}^{h-1}(t + i) + y_t. \tag{3.8}$$

3.1.4 Seasonality

Seasonality is another important characteristic of time series, and likewise important to model in order to obtain accurate forecasts. m denotes the seasonal period, corresponding to the number of observations within a season. The seasonal period depends on the nature of the data source and the observation frequency. For example, weather temperature data is expected to have yearly seasonality, and if the observation frequency is monthly, $m = 12$ is a natural choice. A naive seasonal forecast can be to predict the value of the last observation of the same seasonal occurrence (as shown in Hyndman et al. (2008)), which for the case of temperature data is the previous observation of the same month. This can be expressed as a forecast function:

$$\hat{y}_{t+h|t} = y_{t+h-m\cdot(k+1)}, \tag{3.9}$$

where $k = \text{int}((h - 1)/m)$.

Now we will move to more complicated models, but the fundamentals are based on the concepts explained in this section.

3.2 Exponential Smoothing

Simple exponential smoothing was suggested in the late 1950s, and has since been one of the most used forecasting methods (Hyndman and Athanasopoulos (2018)). Whereas in the moving average, the last observations are weighted equally, the idea of exponential smoothing is a weighted average where the weights are decaying exponential. The one-step-ahead forecast at time t is given by:

$$\hat{y}_{t+1|t} = \alpha y_t + (1 - \alpha)\hat{y}_{t|t-1}, \tag{3.10}$$

where $0 \leq \alpha \leq 1$ is a smoothing parameter. Notably, we can rewrite this function to be:

$$\hat{y}_{t+1|t} = \hat{y}_{t|t-1} + \alpha z_t = \hat{y}_{t|t-1} + \alpha(y_t - \hat{y}_{t|t-1}), \tag{3.11}$$

where z_t is the residual of time-step t. Further, his is a recursive function such that:

$$\begin{aligned} \hat{y}_{t+1|t} &= \alpha y_t + \alpha(1-\alpha)y_{t-1} + \alpha(1-\alpha)^2 y_{t-2} + ... + (1-\alpha)^t y_0 \\ &= (1-\alpha)^t y_0 + \sum_{j=0}^{t-1} \alpha(1-\alpha)^j y_{t-j}, \end{aligned} \tag{3.12}$$

where y_0 is the initial value of the exponential smoothing. As the residual z_{t+1} is unknown, a simple assumption is that $\hat{z}_{t+1} = 0$, and the multiple steps forecast function of a simple exponential smoothing is thus given by (Hyndman et al. (2008)):

$$\hat{y}_{t+h|t} = \hat{y}_{t+1|t} = \alpha y_t + (1-\alpha)\hat{y}_{t|t-1}, \tag{3.13}$$

which forecasts the h-step-ahead forecast as a last day forecast of the previous forecast. This forecast is not very enlightening, and will not model trend or seasonality. Thus Holt-Winters' additive method with trend and seasonality can be used to obtain more accurate forecasts:

$$\begin{aligned} \hat{y}_{t+h|t} &= \ell_t + hb_t + s_{t+h+m(k+1)} \\ \ell_t &= \alpha(y_t - s_{t-m}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \\ s_t &= \gamma(y_t - l_{t-1} - b_{t-1}) + (1-\gamma)s_{t-m}, \end{aligned} \tag{3.14}$$

where α, β and γ are smoothing parameters for the level ℓ_t, the trend b_t and the seasonal component s_t respectively. m denotes the seasonal frequency and $k = \text{int}((h-1)/m)$. The multiplicative method of Holt-Winters can be written as:

$$\begin{aligned} \hat{y}_{t+h|t} &= (\ell_t + hb_t)s_{t+h-m(k+1)} \\ \ell_t &= \alpha\frac{y_t}{s_{t-m}} + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\ b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \\ s_t &= \gamma\frac{y_t}{(l_{t-1} + b_{t-1})} + (1-\gamma)s_{t-m}, \end{aligned} \tag{3.15}$$

A more thorough derivation can be seen in chapter 7.1-7.3 in Hyndman and Athanasopoulos (2018).

3.2.1 State Space Formulation

As shown in Hyndman et al. (2008), formulating the exponential smoothing as a State Space model will make it possible to derive forecasting uncertainty for Holt-Winters' exponential smoothing models. We will not go into detail on State Space models in this thesis, but one can find the derivations in Hyndman et al. (2008). In short terms it is a general form of writing a number of forecast methods, including exponential smoothing. However, we will refer to versions of exponential smoothing as ETS(Error, Trend, Season), where the error can be additive (A) or multiplicative (M), the trend can be absent (N), additive (A) or dampened additive (A$_d$), and the season can be additive (A) or multiplicative. As an example, the simple exponential smoothing (3.10) will take the State Space form of ETS(A,N,N), the additive Holt-Winters' (3.14) will be referred to as ETS(A,A,A) and the multiplicative Holt-Winters (3.15) as ETS(A,A,M).

3.2.2 Prediction Uncertainty in Exponential Smoothing

Assuming independently distributed Gaussian noise, it is possible to derive the uncertainty of a forecast. From Hyndman et al. (2008), we have the following uncertainty estimate for the h-step-ahead forecast for a ETS(A,N,N) (3.10) model:

$$\sigma_h^2 = \Big[1 + (h-1)\alpha^2\Big]\sigma^2. \tag{3.16}$$

Further, we can add a trend, ETS(A,A,N), and this gives the forecast uncertainty:

$$\sigma_h^2 = \Big[1 + (h-1)\big[\alpha^2 + \alpha\beta h + \frac{1}{6}h(2h-1)\beta^2\big]\Big]\sigma^2. \tag{3.17}$$

Lastly, we can add seasonality to the model, ETS(A,A,A), and from Hyndman et al. (2008) we have the uncertainty of the h-step-ahead forecast:

$$\sigma_h^2 = \Big[1 + (h-1)\big[\alpha^2 + \alpha\beta h + \frac{1}{6}h(2h-1)\beta^2\big] + \gamma k\big[2\alpha + \gamma + \beta m(k+1)\big]\Big]\sigma^2, \tag{3.18}$$

where m denotes the seasonal frequency and k the integral part of h/m. It should be noted that removing the seasonality (setting $\gamma = 0$) in equation (3.18) will lead to equation (3.17). Further removing trend (setting $\beta = 0$) will reduce the expression to equation (3.16).

Estimating σ^2 as the residual variance:

$$\hat{\sigma}^2 = \frac{1}{n-2}\sum_i^n (y_i - \hat{y}_i)^2, \tag{3.19}$$

we can obtain an estimate $\hat{\sigma}_h^2$ of σ_h^2 using equation (3.16), (3.17) or (3.18). Due to the model being linear, we have that if the error ϵ_i is Gaussian, then $y_{t+h}|y_t$ is also Gaussian (Hyndman et al. (2008)). Thus we can use that the forecast is Gaussian distributed, and express the prediction interval of the h-step-ahead forecast $\hat{y}_{t+h|h}$ as:

$$[\hat{y}_{t+h|t} - z_{\alpha/2} \cdot \hat{\sigma}_h, \quad \hat{y}_{t+h|t} + z_{\alpha/2} \cdot \hat{\sigma}_h], \tag{3.20}$$

where $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of a standard Normal distribution.

It should be noted that the uncertainty estimates only apply to additive exponential smoothing models, and if any of the terms are multiplicative, as shown in (3.15), the calculations would be more complicated. We will not go into estimating uncertainty of multiplicative models in this thesis, but this can be seen in Hyndman et al. (2008).

3.2.3 Model Selection with AICc

The Holt-Winters' exponential smoothing with lowest corrected Akaike information criterion (AICc) will be chosen for the data set at hand. The AICc is a extention of the Akaike information criterion (AIC), which aims to find the model with the maximum logarithmic likelihood. AICc will in addition penalize the number of parameters used, which makes it better for comparing models with different complexity. AIC and AICc is given by:

$$\begin{aligned} \text{AICc} &= -2\ln\mathcal{L}\big(\hat{\phi},\hat{\theta},\frac{S(\hat{\phi},\hat{\theta})}{n}\big) + \frac{2kn}{n-k-1} \\ &= \text{AIC} + \frac{2k(k+1)}{n-k-1}, \end{aligned} \tag{3.21}$$

where $k = p + q + 1$ is the model complexity. $\mathcal{L}\big(\hat{\phi},\hat{\theta},\frac{S(\hat{\phi},\hat{\theta})}{n}\big)$ is the likelihood function, given the estimated parameters $\hat{\phi}$ and $\hat{\theta}$. As the goal is to maximize the likelihood function, the AIC and AICc has to be minimized. AICc will also be used for model selection for the ARIMA model.

3.3 ARIMA

Auto-regressive integrated moving average (ARIMA) has been the staple of time series analysis and forecasting for years, and is a class of models that represents a time series as a linear function of previous values and previous residuals. An ARIMA process is composed of different dependencies to model the time series as well as possible. First, an AR(p) model describes a linear combination of previous values, and we have that $\{Y_t\}$ is an AR(p) process if:

$$\phi(B)Y_t = Z_t, \tag{3.22}$$

where $\phi(B) = (1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p B^p)$, B is the backshift operator so that $B^k Y_t = Y_{t-k}$ and $Z_t \sim WN(0, \sigma^2)$ (Brockwell and Davis (2016)). Further, we have a MA(q) model that describes the value as a linear combination of previous residuals (or forecast errors). We have that $\{Y_t\}$ is a MA(q) process if:

$$Y_t = \theta(B)Z_t, \tag{3.23}$$

where $\theta(B) = (1 - \theta_1 B - \theta_2 B^2 - ... - \theta_q B^q)$, and B and Z_t is given above. Differencing is also a common technique that is being used to obtain stationary time series by removing trend or seasonality. The idea is to transform the time series by:

$$Y_t' = (1 - B)^d Y_t, \tag{3.24}$$

where Y_t' is the differenced time series, and d is a non negative integer. Thus we have that $\{Y_t\}$ is an ARIMA(p, d, q) process:

$$\phi(B)(1-B)^d Y_t = \theta(B) Z_t, \tag{3.25}$$

where $\phi(B)$, $\theta(B)$ and Z_t is given above.

Handling seasonality, we can analogous to the ARIMA process obtain a model for the seasonal part as an ARIMA process with seasonal parameters P, D and Q, and seasonal frequency s. A seasonal ARIMA(p, d, q)x(P, D, Q)$_s$-process (often referred to as SARIMA) is then defined by:

$$\phi(B)\Phi(B^s)(1-B)^d(1-B^s)^D Y_t = \theta(B)\Theta(B^S) Z_t, \tag{3.26}$$

where $\phi(z) = (1-\phi_1 z - ... - \phi_p z^p)$, $\Phi(z) = (1-\Phi_1 z - ... - \Phi_P z^P)$, $\theta(z) = (1-\theta_1 z - ... - \theta_q z^q)$, $\Theta(z) = (1-\Theta_1 z - ... - \Theta_Q z^Q)$. $(1-B)^d$ and $(1-B^s)^D$ are differencing and seasonal differencing respectively (Brockwell and Davis (2016)). Note that if both regular and seasonal components of either AR or MA is present, we will obtain cross-terms.

The ARIMA model can be fitted by finding p, d, q, P, D, and Q, and then finding the parameters $(\hat{\phi}_1, ..., \hat{\phi}_p, \hat{\Phi}_1, ..., \hat{\Phi}_P, \hat{\theta}_1, ..., \hat{\theta}_q, \hat{\Theta}_1, ..., \hat{\Theta}_Q)$ by maximum likelihood estimation.

3.3.1 Forecasting with ARIMA Models

As shown, an ARIMA model will try to model the dynamics of the time series. Thus by assuming the same dynamics will continue into the future, one can use the obtained ARIMA model to forecast. Forecasting an ARIMA process is shown in great detail in Brockwell and Davis (2016), p. 87, however we will show the main results here. For a simple AR(p) process, the obtained forecast function will be:

$$\hat{y}_{t+1|t} = \sum_{j=1}^{p} \phi_j y_{t+1-j} = \phi_1 y_t + \phi_2 y_{t-1} + ... + \phi_p y_{t+1-p}, \tag{3.27}$$

which can be computed from equation (3.22). Further, a MA(q) process can be forecast by:

$$\hat{y}_{t+1|t} = \mu - \sum_{j=1}^{q} \theta_j z_{t+1-j} = \mu - \theta_1 z_t - \theta_2 z_{t-1} - ... - \theta_q z_{t+1-q}, \tag{3.28}$$

where μ is the mean of the time series.

Combining the AR(p) and the MA(q) formulas gives us the forecasting formula of an ARMA(p, q) process:

$$\hat{y}_{t+1|t} = \mu + \sum_{j=1}^{p} \phi_j y_{t+1-j} - \sum_{j=1}^{q} \theta_j z_{t+1-j}. \tag{3.29}$$

Analogous, the explicit forecast function can be derived for a SARIMA(p,d,q)x(P,D,Q)$_s$ model.

3.3.2 Prediction Uncertainty in ARIMA Models

Analogous to exponential smoothing, we want to estimate the prediction error of the h-step-ahead forecast, in order to obtain prediction intervals. We will show how this can also be done for an ARMA model, but this can be done with SARIMA models as shown in Brockwell and Davis (2016).

An ARMA process is causal if it is possible to represent the ARMA process as a MA process. We first define $\psi(z) = (1 - \psi_1 z - \psi_2 z^2 - ...)$, analogous to the definitions of $\phi(z)$ and $\theta(z)$. If the ARMA process is causal, we can write $\theta(B) = \phi(B)\psi(B)$, and then rewrite the ARMA process:

$$\begin{aligned} \phi(B)Y_t &= \theta(B)Z_t \\ Y_t &= \frac{\theta(B)}{\phi(B)}Y_t = \psi(B)Z_t. \end{aligned} \tag{3.30}$$

In order for this to be valid, we need that $\psi(z) \neq 0$ for $|z| \leq 1$. Further, we can obtain the forecast error by:

$$\hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n} Z_i^2, \tag{3.31}$$

and from Brockwell and Davis (2016) we have that the h-step-ahead forecast error is:

$$\hat{\sigma}_n^2(h) = \sum_{j=0}^{h-1} \psi_j^2 \sigma^2. \tag{3.32}$$

3.4 Neural Networks

For classical statistical methods, the theory regarding the models were often developed before one had the data and computational power available to utilize the methods. The idea of neural networks were developed long before they were usable, but they have stepped into the light in the last two decades. And while there has been research around the statistical properties of neural networks, the practical use has skyrocketed. In the hunt for better performance on specific task, the architectures have become more sophisticated and advanced. Due to the advances in practical use, it is not developed a theoretical foundation in the same degree as for statistical methods. In this section we will introduce the theory behind the basic neural network architectures and nodes utilized in this thesis. We will also introduce some newer developed theory to obtain uncertainty estimates of neural networks in the next section.

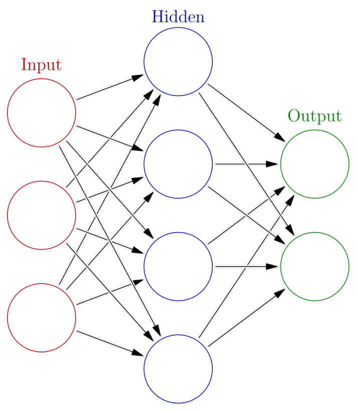


Figure 3.1: Feed forward neural network architecture (Glosser.ca (2019)).

Deep learning has shown great results in many areas of computing; as regression models, image recognition and processing sequential data. In many cases, sophisticated architectures are utilized, however we will first define the basic architecture of feed forward networks. A feed forward network consists of one or multiple layer(s), each with at least one node (neuron). The layers are connected by feed forward connections, where for a fully connected structure, the input of a layer is the weighted sum of the output of the previous layer. Figure 3.1 illustrates a fully connected feed forward network with three layers. Mathematically, a fully connected feed forward layer can be defined as:

$$z^{(l+1)} = g\big(W^{(l+1)}z^{(l)} + b^{(l+1)}\big), \tag{3.33}$$

where $z^{(l+1)} \in \mathbb{R}^{d^{(l+1)}}$ denotes the output of layer $(l+1)$ with dimension $d^{(l+1)}$. Thus $z^{(l)} \in \mathbb{R}^{d^{(l)}}$ denotes the output of the previous layer. We also have that $W^{(l+1)} \in \mathbb{R}^{d^{(l+1)} \times d^{(l)}}$ denotes the weights from layer (l) to layer $(l+1)$, and g is an activation function.

Activation Function

The activation function can be any appropriate function, but the functions below are the most commonly used. Rectified Linear Units (ReLU) is defined as:

$$g_{\text{ReLU}}(z) = z^{+} = \text{max}(0, z), \tag{3.34}$$

and thus obtain nonlinearity in $z = 0$. Nonlinearity is necessary in order to approximate nonlinear functions. If g is a linear function in equation (3.33), then the output will be a linear combination of the inputs, which is a linear regression. As ReLU is nonlinear, it can be used in Neural Networks to approximate nonlinear function. It also has the application of being unbounded, which can be useful in some cases. In addition, the gradient is easy to compute, and linear. To avoid "dying" nodes, where the $g_{ReLU} = 0$ and $\frac{\partial g_{\text{ReLU}}(z)}{\partial z} = 0$, one can use Leaky ReLU, as defined by:

$$g_{\text{Leaky ReLU}}(z) = \text{max}(0.01z, z), \tag{3.35}$$

which will avoid nodes outputting zeros. Further we have two hyperbolic activation functions that see some usage in neural networks. The sigmoid activation function is defined as:

$$g_{\sigma}(z) = \frac{1}{1 + e^{-z}}. \tag{3.36}$$

Sigmoid has the property of being bounded by the interval $[0, 1]$, as $e^{-z} \in (0, \infty), z \in \mathbb{R}$. Thus it proves useful as a gate, by either squashing the input to 0 or 1. As $|z| \to \infty$, the gradient $\frac{\partial g_{\sigma}(z)}{\partial z} \to 0$, which leads to the problem of vanishing gradients, where the weight wont update. Thus ReLU (3.34) is often preferred, unless the gating/squashing nature of sigmoid is needed.

Another activation function that resembles sigmoid, is the hyperbolic tangent function:

$$g_{\text{tanh}}(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{3.37}$$

which is bounded on $[-1, 1]$ for all $z \in \mathbb{R}$. Equal to sigmoid, the gradient of the hyperbolic tangent approaches zero as $|z|$ get large. Thus we have the same issues of vanishing gradient as with sigmoid.

Loss Function

In the general regression and forecasting setting, minimizing the distance from the predicted value to the real value is usually the goal. The mean squared error (section 3.8.1) is common choice when the objective is to minimize this distance. Let $\hat{y}$ be the predicted value of y, then we have the mean squared error (MSE) loss function:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i}^{N} (y_i - \hat{y}_i)^2, \tag{3.38}$$

where N is the number of samples. Other loss functions can be used depending on the task at hand, which we will introduce later.

Mini-batch Stochastic Gradient Descent

In order to improve the performance of the neural network, the parameters have to be updated, referred to as back-propagation. It has proven suboptimal to update over the entire set of samples at the same time, due to poor generalization (Yao et al. (2018)). On the other hand, updating the weights with regard to one sample at the time is not optimal either. Thus some mini-batch of size m, referred to as batch size, of samples $Y = (y_1, \dots, y_m)$ with the respective predictions $\hat{Y} = (\hat{y}_1, \dots, \hat{y}_m)$ is stochastically chosen. The parameters θ are updated according to:

$$\theta_i = \theta_{i-1} - \eta f\big(\nabla_{\theta_{i-1}} \mathcal{L}(Y, \hat{Y})\big), \tag{3.39}$$

where θ_i is the trainable parameters, including but not limited to the weights W, within the neural network at the i^{th} update. η is the learning rate, f an optimization function and $\nabla_{\theta_{i-1}} \mathcal{L}$ the gradient of an arbitrary loss function with respect to the parameters θ_{i-1}. In the simplest case, $f\big(\nabla_{\theta_{i-1}} \mathcal{L}(Y, \hat{Y})\big) = \nabla_{\theta_{i-1}} \mathcal{L}(Y, \hat{Y})$, which means the parameters θ are updated along the gradient of the $\mathcal{L}$. In this thesis, we will use the optimization function Adam (Kingma and Ba (2014)), which uses the momentum of the weights in order to adapt the learning rate accordingly.

3.4.1 Simple Recurrent Neural Networks

Recurrent neural networks (RNN) has shown promising results in time series forecasting, and recently being a part of the winning solution of the M4 competition (Makridakis et al. (2020)). In addition, RNNs has shown great results in other recurrent tasks, such as speech recognition and sentiment analysis. This makes it a natural choice when exploring uncertainty estimation in forecasting with neural networks. The method used is often referred to as simple RNN cell, and contains stacked fully-connected recurrent layers, where the output is fed back to the input. To simplify notation, x will denote the input of a layer, while y will denote the output, analogous to $z^{(l)}$ and $z^{(l+1)}$ from equation 3.33. From (Sezer et al. (2019), equation 8 and 9) we have a system of equations for a recurrent neural network, and by modifying according to the implementation of the 'Simple RNN' in `Keras` (Chollet et al. (2015)) we have:

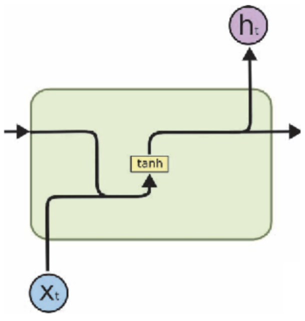


Figure 3.2: An illustration of the RNN cell (Mani (2019b)) as described in equation 3.40.

$$y_t = g(W_h y_{t-1} + W_x x_t + b_h), \tag{3.40}$$

where b_h is a bias term, W_h and W_x are weights, x_t the input, and y_{t-1} and y_t are the output of the previous and the current RNN cell respectively. In addition g is an activation function, usually the hyperbolic tangent function (3.37) for recurrent cells.

3.4.2 Long Short-Term Memory

Long short-term memory (LSTM) units was introduced by Hochreiter and Schmidhuber (1997), and has seen frequent use in sentiment analysis and speech recognition due to its ability to capture long-term dependencies (Hewamalage et al. (2019)). LSTM was also a part of Uber's winning hybrid ES-RNN solution of the highly regarded M4 competition (Smyl and Pasqua (2018), Makridakis et al. (2020)). The LSTM builds upon the simple recurrent neural network from equation (3.40), and are described by the following set of equations (Hewamalage

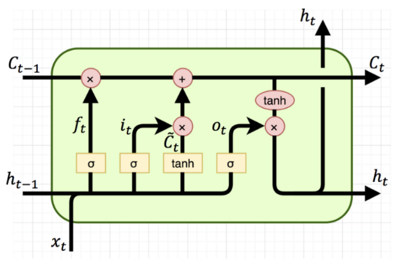


Figure 3.3: An illustration of the LSTM cell (Mani (2019a)) as described in equation 3.41.

et al. (2019)):

$$\begin{aligned}
i_t &= g_\sigma(W_i h_{t-1} + V_i x_t + b_i) \\
o_t &= g_\sigma(W_o h_{t-1} + V_o x_t + b_o) \\
f_t &= g_\sigma(W_f h_{t-1} + V_f x_t + b_f) \\
\tilde{C}_t &= g_{\text{tanh}}(W_c h_{t-1} + V_c x_t + b_c) \\
C_t &= i_t \odot \tilde{C}_t + f_t \odot C_{t-1} \\
h_t &= o_t \odot g_{\text{tanh}}(C_t) \\
z_t &= h_t.
\end{aligned} \tag{3.41}$$

Here $h_t \in \mathbb{R}^d$ is the hidden state which accounts for short-term dependencies (as in simple RNN) and $C_t \in \mathbb{R}^d$ is the cell state that captures long-term dependencies. $i_t, o_t, f_t \in \mathbb{R}^d$ is the input, output and forget gate vectors. The input gate determines how much of the input x_t should be added to the cell state C_t. The output gate determines how much of the cell state C_t should be outputted, and the forgot gate determines how much of the previous cell state C_{t-1} will be relied upon. $W_i, W_o, W_f, W_c \in \mathbb{R}^{dxd}$, $V_i, V_o, V_f, V_c \in \mathbb{R}^{dxd}$ and $b_i, b_o, b_f, b_c \in \mathbb{R}^d$ are weights for the hidden state, input and the bias term respectively. Further, $\odot$ is the element wise multiplication, $g_{tanh}(z)$ the hyperbolic tangent activation function (3.37) and g_σ the sigmoid activation function (3.36).

3.5 Monte Carlo Dropout in Neural Networks

This section will introduce the theory behind the MC dropout model, which is used as a state-of-the-art model of comparison to the generative adversarial network. Predicting uncertainty in neural networks is not a task that has been greatly explored. However the recent explosion in use of neural networks naturally provokes such research, as it entails great value. This section contains theory developed by Zhu and Laptev (2017), which in turn is based upon theory developed by Gal and Ghahramani (2015). Here it is suggested under the Bayesian neural network framework that uncertainty can be divided into three parts: model uncertainty, model misspecification and inherent noise. In this thesis we will only try to estimate the model uncertainty and the inherent noise, and leave the model misspecification out of the uncertainty estimate. Firstly, we will introduce dropout, as it is central to the methodology of the MC dropout.

3.5.1 Dropout

Dropout is a regularization technique that is widely used, easy to implement and has shown great results when training deep neural networks. The idea is to remove a random selected proportion of the units in a neural layer, "dropping" those units from the network temporarily. In addition, newer findings such as Gal and Ghahramani (2015) suggests that dropout approximates a deep Gaussian process, and thus can be used to estimate the uncertainty.

The standard form of dropout will for a given layer remove an unit with probability p (called dropout rate) when feeding a training batch through the network. The same units will be removed during back-propagation, in order to train the correct weights. For each training batch, a different subset of units from a layer is being dropped. This will be analogous to sampling smaller networks from the full network architecture (N. Srivastava and Salakhutdinov (2014)). The network will train slower, due to only parts of the weights being trained for each batch, but will in many cases lead to better generalizing. The idea is to disable the dropout during testing, which practically will be combining the sampled networks, making for a more robust and better model (N. Srivastava and Salakhutdinov (2014)). However, to estimate model uncertainty, dropout would be enabled during testing as well.

Let $z^{(l)}$ be the output vector of layer (l) with size $d^{(l)}x1$. Further, let $r^{(l)} \sim$ Bernoulli$(p^{(l)})$ be a vector of independent Bernoulli distributed values with probability $p^{(l)}$ of being 1, and with size $d^{(l)}x1$. $1 - p^{(1)}$ will be the dropout rate for layer (l). Thus we have the following modification from equation (3.33):

$$\begin{aligned} \tilde{z}^{(l)} &= \frac{1}{p^{(l)}} r^{(l)} \odot z^{(l)} \\ z^{(l+1)} &= g\big(W^{(l+1)} \tilde{z}^{(l)} + b^{(l+1)}\big), \end{aligned} \tag{3.42}$$

where $\odot$ is the element wise multiplication. Dropout can be applied to recurrent and convolutional layers as well, but it is only the feed forward connections that can be dropped out. Equation (3.40) can thus be rewritten to:

$$\begin{aligned} \tilde{x}_t &= \frac{r_t}{p} x_t \\ y_t &= W_h g(y_{t-1} + W_x \tilde{x}_t + b_h), \end{aligned} \tag{3.43}$$

where $r_t \sim$ Bernoulli(p) is a binary value.

As suggested by Gal and Ghahramani (2015), by sampling a prediction multiple times with dropout active, name Monte Carlo dropout, one can obtain a estimate of the distribution of the model predictions. As this approximates a deep Gaussian process (Gal and Ghahramani (2015)), the expected distribution is Gaussian, and one can obtain the model uncertainty.

3.5.2 Prediction Uncertainty in Neural Networks

Let $f_{\hat{\theta}}(\cdot)$ define the trained neural network, with model parameters $\hat{\theta}$, that tries to approximate the true model, $f_\theta(\cdot)$. Further, we denote the validation set as (X', Y'), and the test set of new observations as (X^*, Y^*). By assuming Gaussian distributed noise, we have that $y \sim N(f_\theta(x), \sigma^2)$ (Zhu and Laptev (2017)), and thus we can make a estimation of y^* by $\hat{y}^* = f_{\hat{\theta}}(x^*)$. Further we have that the prediction interval for a Gaussian distribution is given by:

$$[y^* - z_{\alpha/2}\hat{\sigma}, y^* + z_{\alpha/2}\hat{\sigma}], \tag{3.44}$$

where $\hat{\sigma}$ is the prediction standard error $\hat{\sigma} = \sqrt{Var(y^*)}$, and $z_{\alpha/2}$ is the upper $\alpha/2$ quantile of a standard Normal distribution. To estimate the $(1 - \alpha) \cdot 100\%$ prediction interval, we need to estimate y^* and $\hat{\sigma}$.

Further, we have from Zhu and Laptev (2017) that the distribution of the prediction is given by marginalizing out the parameters θ, and we obtain the following equation:

$$p(y^*|x^*) = \int_\theta p(y^*|f_\theta(x^*))p(\theta|X,Y)d\theta. \tag{3.45}$$

Then we have from the law of total variance that:

$$\begin{aligned} \text{Var}(Y) &= E[\text{Var}(Y|X)] + \text{Var}(E[Y|X]) \\ &= \sigma^2 + \text{Var}(f_{\hat{\theta}}(X)). \end{aligned} \tag{3.46}$$

Here σ^2 can be seen as the inherent noise in the observations: $y - f_\theta(x) = \sigma^2$, and $\text{Var}(f_{\hat{\theta}}(X))$ the model uncertainty.

Model Uncertainty

As shown by Gal and Ghahramani (2015), doing Monte Carlo estimations with dropout active, named MC dropout, approximates a probabilistic deep Gaussian process. Let x^* be an input, and $(\hat{y}^*_{(1)}, ..., \hat{y}^*_{(B)})$ be B Monte Carlo forecasts of y^*. In one-step ahead forecast, x^* will denote $(y_{t-\ell}, ..., y_t)$, where ℓ is the sequence length , and y^* denote y_{t+1}. Thus we can obtain a estimate for the model uncertainty:

$$\hat{Var}(f_{\hat{\theta}}(x^*)) = \frac{1}{B}\sum_{b=1}^{B}\left(\hat{y}^*_{(b)} - \overline{\hat{y}}^*\right)^2, \tag{3.47}$$

where $\overline{\hat{y}}^*$ is the sample mean.

Inherent Noise

From Zhu and Laptev (2017), we estimate the inherent noise level σ^2, by the residual sum of squares:

$$\hat{\sigma}^2 = \frac{1}{V}\sum_{v=1}^{V}(y'_v - f_{\hat{\theta}}(x'_v))^2, \tag{3.48}$$

where $X' = (x'_1, ..., x'_V)$, $Y' = (y'_1, ..., y'_V)$ is a validation set. As further shown by Zhu and Laptev (2017), if one assumes that $f_{\hat{\theta}}(\cdot)$ is an unbiased estimator of the true model $f_\theta(\cdot)$, that is $E[f_{\hat{\theta}}(\cdot)] = f_\theta(\cdot)$, we have for large training data that $\hat{\sigma}^2$ is an unbiased estimator for σ^2. By the use of Bias-Variance Trade-off, we obtain:

$$\begin{aligned} E[\hat{\sigma}^2] &= \frac{1}{V}\sum_{v=1}^{V} E\left[(y'_v - f_{\hat{\theta}}(x'_v))^2\right] \\ &= \text{Bias}[f_{\hat{\theta}}(X')]^2 + \text{Var}(f_{\hat{\theta}}(X')) + \sigma^2, \end{aligned} \tag{3.49}$$

where $\text{Bias}[f_{\hat{\theta}}(X')]^2$ can be removed, due to the assumption of $f_{\hat{\theta}}(\cdot)$ being an unbiased estimator. Thus, as shown by Zhu and Laptev (2017), $\hat{\sigma}^2$ is an asymptotically unbiased estimator of σ^2.

3.6 Generative Adversarial Networks

Generative Adversarial Network (GAN), introduced by Goodfellow et al. (2014), is a machine learning framework that has gotten a lot of recognition due to promising results in image generation and transformation. Among the most publicly available uses is Deep Fake (Tolosana et al. (2020)), where realistic videos and photos of people have been generated in order to create fake news, financial fraud etc. thispersondoesnotexist.com [1](Wang (2019)) is a website that generates realistic images of persons that are generated according to a StyleGAN suggested by Karras et al. (2019). Another use is the transformation of image-to-image translation by the use of CycleGAN (Zhu et al. (2017)). Examples show that paintings can be transformed to realistic looking pictures and reversely images can be transformed to look like paintings in the style of famous painters such as Van Gogh and Monet. Even tasks such as generating images from text and reversely describing images can be done using GAN (Gorti and Ma (2018)).

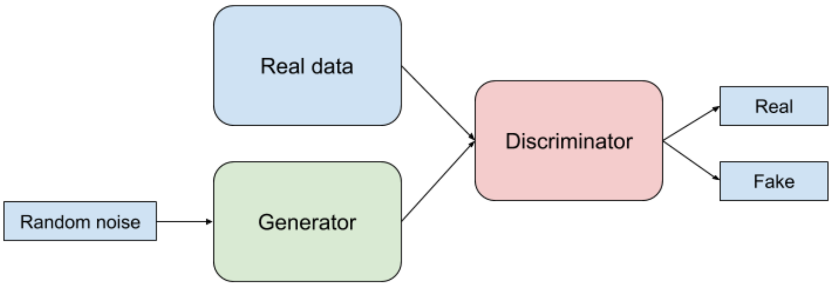


Figure 3.4: Generative Adversarial Network

The GAN framework is motivated by game theory; a generative model $\mathcal{G}$ aims to generate samples that fools the discriminative model $\mathcal{D}$. The discriminative model $\mathcal{D}$ on the other hand is fed both real data and samples generated by the generator $\mathcal{G}$, and tries to distinguish and label the real data and generated data as real and fake respectively. This is corresponds to a minimax two-player game, where the two models have competing objectives. In reality, if trained correctly to a minimum, this leads to the generator $\mathcal{G}$ capturing the distribution of the real data and being able to generate samples from the distribution. In order to achieve stochastic generation of samples, the generator $\mathcal{G}$ take a random latent code z as input. It may therefore be more correct to refer to the generator $\mathcal{G}$ as a transformer, as it is a transformation from random noise to the distribution of the real data. Figure 3.4 shows the framework, where the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$ can be any functions. However, the functions $\mathcal{G}$ and $\mathcal{D}$ often refers to neural networks. Let $V(\mathcal{D}, \mathcal{G})$ be the value function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \big[\log \mathcal{D}(x)\big] + \mathbb{E}_{z \sim p_z(z)} \big[\log (1 - \mathcal{D}(\mathcal{G}(z))\big], \qquad (3.50)$$

where $x \sim p_{\text{data}}(x)$ is data drawn from the true distribution p_{data} (data samples) and $z \sim p_z(z)$ is the latent code drawn from a noise distribution p_z. As noted by Goodfellow

[1]https://thispersondoesnotexist.com/

et al. (2014), a unique solution exists, and is shown to be where $\mathcal{G}$ learns the true distribution $p_g = p_{\text{data}}$, where p_g denotes the distribution generated from the generator. Further, the optimal discriminator given a fixed generator is given by:

$$\mathcal{D}^*_{\mathcal{G}}(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}, \tag{3.51}$$

which for $p_g = p_{\text{data}}$ yields $\mathcal{D} = \frac{1}{2}$. Further Goodfellow et al. (2014) shows that the maximization object of $\mathcal{D}$ can be formulated as:

$$\max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\log \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)} \right] + \mathbb{E}_{z \sim p_z(z)} \left[\log \frac{p_g(x)}{p_{\text{data}}(x) + p_g(x)} \right], \tag{3.52}$$

which for $p_g = p_{data}$ yields $\max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = -2\text{log}(2)$. We can further rewrite $V(\mathcal{D}, \mathcal{G})$ as:

$$\begin{aligned} V(\mathcal{D}, \mathcal{G}) &= \int_x p_{\text{data}}(x) \log \mathcal{D}(x)\, dx + \int_z p_z(z) \log \big(1 - \mathcal{D}(\mathcal{G}(x))\big)\, dz \\ &= \int_x p_{\text{data}}(x) \log \mathcal{D}(x) + p_g(x) \log \big(1 - \mathcal{D}(x)\big)\, dx, \end{aligned} \tag{3.53}$$

which for an optimal discriminator $\mathcal{D}^*_{\mathcal{G}}$ is given by:

$$C(\mathcal{G}) = V(\mathcal{D}^*_{\mathcal{G}}, \mathcal{G}) = D_{KL}(p_{\text{data}} \,||\, p_{\text{data}} + p_g) + D_{KL}(p_g \,||\, p_{\text{data}} + p_g), \tag{3.54}$$

where $D_{KL}(P \,||\, Q)$ is the Kullback-Leibler distance of the probability distribution P relative to the probability distribution Q (see more detailed explanation in section 3.8.6). Further, we can see that by extracting the optimal solution $C(\mathcal{G}^*) = -2\text{log}(2)$ from $C(\mathcal{G})$, we get:

$$\begin{aligned} C(\mathcal{G}) &= D_{KL}\Big(p_{\text{data}} \,||\, \frac{p_{\text{data}} + p_g}{2}\Big) + D_{KL}\Big(p_g \,||\, \frac{p_{\text{data}} + p_g}{2}\Big) - \log(4) \\ &= 2 \cdot D_{JS}(p_g \,||\, p_{\text{data}}) - 2\log(2). \end{aligned} \tag{3.55}$$

Here $D_{JS}(P \,||\, Q)$ is the Jensen-Shannon divergence (section 3.8.7), which is non-negative and $D_{JS} = 0$ only if $P = Q$. This was shown by Goodfellow et al. (2014), which implicates that given an optimal discriminator $\mathcal{D}^*_{\mathcal{G}}$, the value function of the generator $C(\mathcal{G})$ minimizes the Jensen-Shannon divergence, with global minimum at $p_g = p_{\text{data}}$.

Loss Function

In order to minimize the value functions described in equation 3.50, we use the loss function binary cross-entropy[2]:

$$\mathcal{L}_{\text{BCE}} = -(y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})), \tag{3.56}$$

[2]https://keras.io/api/losses/probabilistic_losses/binarycrossentropy-class

where y is the true class of the label $\{0, 1\}$ and $\hat{p}$ the predicted class by the discriminator. As the activation function of the final layer is given as the sigmoid function (3.36), the predicted class label will be $(0, 1)$, however not exactly 0 or 1. This gives $\lim_{\hat{p}\to y} L_{BCE} \to 0^{+}$, i.e. the loss function will minimize as the discriminator predicts the right class. When training the generator we will freeze the weights of the discriminator, however, we still feed the generated samples from the generator through the discriminator. By labeling the generated data as real during the training of the generator, the loss function will minimize when the discriminator interprets generated data as real.

3.6.1 Conditional Generative Adversarial Networks

As the performance of Generative Adversarial Networks (GAN) showed promising results, extensions in order to enhance performance and expand its use has been suggested. The conditional version of the GAN framework (CGAN), suggested by Mirza and Osindero (2014), is a useful extension that has seen widespread use. Conditional information, such as class labels or other modalities, can be fed to both the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$ in order to control the mode of the generated samples. In the standard GAN setting, training the generator $\mathcal{G}$ to produce samples of handwritten numbers, one has no control of which number from the original data set the generator generates. However, by feeding the label to the generator and discriminator as a condition, one can then control which number is generated by the conditional label. The results of this example is shown by Mirza and Osindero (2014), and show success in generating the correct number given the condition.

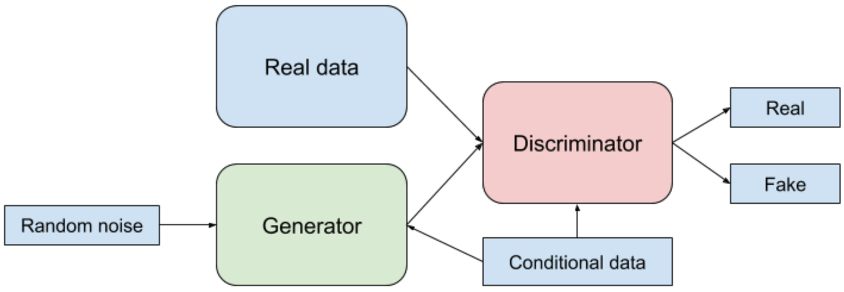


Figure 3.5: Conditional Generative Adversarial Network

Figure 3.5 shows the CGAN, where most of the setup is similar, but the conditional data is fed to both the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$. The value function is similar to the value function of the GAN (3.50):

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\log \mathcal{D}(x|c)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - \mathcal{D}(\mathcal{G}(z|c)|c)\right)\right], \quad (3.57)$$

where c is the conditional information.

3.6.2 Forecasting with Generative Adversarial Networks

ForGAN, as proposed by Koochali et al. (2019), is a natural setup when forecasting with a generative adversarial network. The ForGAN setup is identical to CGAN (3.57), where the conditional information c is defined as the previous observation of a time series $\{Y_t\}$. Let y_t denote a observation from the time series $\{Y_t\}$, and $X_t = (y_t, y_{t-1}, \dots, y_{t-\ell})$ be the previous observations of window length ℓ. Further, let $Z = (z_1, \cdots, z_k)$ be the latent vector of dimension k, where $(z_1, \dots, z_k)$ are i.i.d.[3] and $z_i \sim p_z(z)$, $Z \in \mathbb{R}^k$. The discriminator will then distinguish the real next observation y_{t+1} from the generated forecast $\hat{y}_{t+1} = \mathcal{G}(Z|X_t)$ given the conditional information X_t. The value function can thus be defined as:

$$\begin{aligned}\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = &\mathbb{E}_{y\sim p_t(y)}\big[\log \mathcal{D}(y_{t+1}|X_t)\big] \\ &+ \mathbb{E}_{z\sim p_z(z)}\big[\log\left(1 - \mathcal{D}(\mathcal{G}(Z|X_t)|X_t)\right]. \end{aligned} \tag{3.58}$$

Figure 3.6 illustrates the ForGAN framework as described above. Algorithm 1 further shows the training loop, where θ_d and θ_g is the trainable parameters of $\mathcal{D}$ and $\mathcal{G}$ respectively. $(X^{(i)}, y^{(i)})$ are samples randomly drawn from the training set, and the latent code $Z^{(i)}$ is independent of the time.

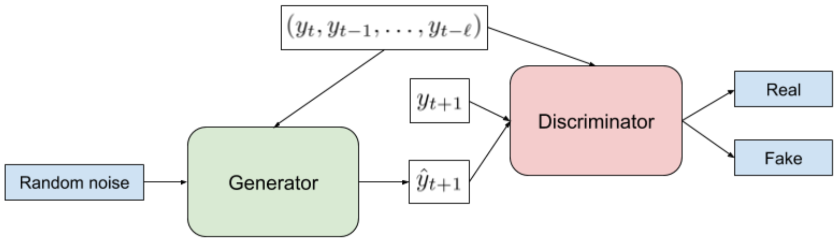


Figure 3.6: Forecasting generative adversarial network framework inspired by Koochali et al. (2019).

Estimating the Forecast Distribution

After the generator $\mathcal{G}$ is trained, then we want to use it to forecast time series values. As it is trained to sample from the estimated forecast distribution, rather than estimate the most likely value, we have to sample multiple values in order to obtain any useful information. We have that:

$$\hat{y}^{(b)}_{t+1|t} = \mathcal{G}(Z^{(b)}|X_t), \tag{3.59}$$

is a Monte Carlo sample of the estimated forecast distribution, where $b = 1, \cdots, B$. Further, we can estimate the forecast mean:

$$\bar{y}_{t+1|t} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}^{(b)}_{t+1|t}. \tag{3.60}$$

[3]Independent and identically distributed random variables.

Algorithm 1: Generative adversarial network training loop, where D_{iter} denotes the number of discriminator iterations per training iteration, which we will discuss later.

```
for number of training iterations do
    for number of $D_{iter}$ do
        Sample mini-batch of $m/2$ samples $\{(X^{(1)}, y^{(1)}), ..., (X^{(m/2)}, y^{(m/2)})\}$ from
          $p_{\text{data}}(y)$, where $X^{(i)} = (y_t, y_{t-1}, \dots, y_{t-\ell})$, $y^{(i)} = (y_{t+1})$.
        Sample mini-batch of $m/2$ latent code samples $\{Z^{(1)}, ..., Z^{(m/2)}\}$, where
          $Z^{(i)} = (z_1^{(i)}, \dots, z_k^{(i)})$ and $z_j^{(i)} \sim p_z(z)$.
        Forecast samples from $\mathcal{G}(Z|X)$:
            $\hat{y}^{(i)} = \mathcal{G}(Z^{(i)}|X^{(i)})$
        Update the discriminator to maximize correct predictions on real data:
            $\nabla_{\theta_d} \frac{2}{m} \sum_{i=1}^{m/2} \Big[\log \mathcal{D}\big(y^{(i)}|X^{(i)}\big)\Big]$
        Update the discriminator to maximize correct predictions on false data:
            $\nabla_{\theta_d} \frac{2}{m} \sum_{i=1}^{m/2} \Big[\log\Big(1 - \mathcal{D}\big(\hat{y}^{(i)}|X^{(i)}\big)\Big)\Big]$
    end
    Train $\mathcal{G}$ to make $\mathcal{D}$ label forecasted values as real ($\mathcal{D}$ is not trainable in this update):
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \Big[\log\Big(1 - \mathcal{D}\big(\mathcal{G}(Z^{(i)}|X^{(i)})|X^{(i)}\big)\Big)\Big]$
end
```

Now, let $(\hat{y}_{t+1|t}^{(1)}, \dots, \hat{y}_{t+1|t}^{(B)})$ be the strictly increasing order of the Monte Carlo samples, the estimated quantile function $\hat{q}_\alpha$ can be computed as:

$$\hat{q}_\alpha\big(\hat{y}_{t+1|t}^{(1)}, \dots, \hat{y}_{t+1|t}^{(B)}\big) = \hat{y}_{t+1|t}^{(B\cdot\alpha)}. \tag{3.61}$$

Thus we have the estimated $(1-\alpha)\cdot 100\%$ prediction interval given by:

$$\big[\,\hat{q}_{\alpha/2}\big(\hat{y}_{t+1|t}^{(1)}, \dots, \hat{y}_{t+1|t}^{(B)}\big),\ \hat{q}_{1-\alpha/2}\big(\hat{y}_{t+1|t}^{(1)}, \dots, \hat{y}_{t+1|t}^{(B)}\big)\,\big]. \tag{3.62}$$

3.6.3 Mode Collapse

One issue related to generative adversarial networks, and brought up already by Goodfellow et al. (2014), is ”the Helvetica scenario”, later named mode collapse. It refers to the situation where the generator $\mathcal{G}$ fails (to some degree, or completely) to generate from some of the modes contained in the data set. An example would be where one wants to generate images of handwritten numbers from 0 to 9. If the data set contains samples of all numbers, however, the generator $\mathcal{G}$ only generates images of some of the numbers, we have a mode collapse. However, the discriminator may then be able to learn that the generator only generates some of the numbers, say 5 and 9, and then predict all images of 5 as false. This may lead the generator to produce samples of another number, which may lead to an oscillation between collapsing to different modes.

However, one can also experience mode collapse where a GANs fail to generate from all of the distribution. This is due to many values of the latent code z collapses to the same value of $\mathcal{G}(z)$. This can be especially challenging in conditional GANs where some form of variation is also introduced through the auxiliary input, however not real randomness. In extreme cases this can lead to a deterministic transformation, where randomness introduced to the latent code z will be ignored (Yang et al. (2019)).

When estimating the forecast uncertainty, both aforementioned forms of mode collapse are unwanted. The former will lead to an underfitting of the time-series dynamics, where the generator $\mathcal{G}$ is not able to model and forecast time dependency. The second form will prevent the ForGAN (3.6.2 from sampling from the whole forecast distribution, which will lead to underestimation of the forecast uncertainty. Usually one wants to overestimate the uncertainty rather than underestimate it, which will be discussed more in section 3.8.4. Both forms of mode collapse are therefor unwanted properties when estimating the forecast uncertainty.

There are various ways to address the issue of mode collapse. Goodfellow et al. (2014) indicated that "$\mathcal{G}$ must not be trained too much without updating $\mathcal{D}$" to avoid mode collapse. This can be achieved by iterating multiple times over the discriminator for each generator iteration, and it's fairly standard to have D_{iter} larger than 1. Wasserstein GAN (Arjovsky et al. (2017), Gulrajani et al. (2017)) is a popular choice, using Wasserstein distance (Frogner et al. (2015)) in order to assess the distance between two probability functions, and have some other useful properties we will discuss in the next section. There are other versions such as Diversity-Sensitive GAN (Yang et al. (2019), which aims to penalize deterministic behavior of the generator by regularizing the relative distance of two samples generated by the generator in relation to the distance between their respective input latent code vectors. Boundary Equilibrium GAN (Berthelot et al. (2017) is another method trying to solve issues related to low diversity and mode collapse. The discriminator is here an auto-encoder, and the contribution of incorrect labeling of false samples to the discriminator loss is given some weight in order to balance the generator and discriminator.

3.6.4 Wasserstein Generative Adversarial Networks

The Wasserstein distance is derived from the cost of the optimal transport plan moving mass from one probability distribution to match the target distribution (Frogner et al. (2015)). The usual analogy is that if two probability distributions represent two piles of dirt, then the Wasserstein distance is the amount of dirt one has to move times the mean distance it has to be moved, and therefore often referred to as the earth mover's distance.

Mathematically it can be defined in many ways, however Arjovsky et al. (2017) introduce it in the GAN context as:

$$W(P,\, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{\gamma \sim (x,y)} \big[||x - y|| \big], \tag{3.63}$$

where P is the predicted distribution and Q the target distribution. $\Pi(P,\, Q)$ is the set of joint probability distributions $\gamma(x, y)$ having P and Q as margins. This can be interpreted as the transport plan where $\gamma(x, y)$ is the function describing the mass to be moved from x to y.

The value function of the Wasserstein loss can be found using the Kantorovich-Rubinstein duality (Arjovsky et al. (2017), Gulrajani et al. (2017)).

$$\min_{\mathcal{G}} \max_{\mathcal{D} \in D} V(\mathcal{D}, \mathcal{G}) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \big[\mathcal{D}(x) \big] - \mathbb{E}_{z \sim p_z(z)} \big[\mathcal{D}(\mathcal{G}(z)) \big], \tag{3.64}$$

where D is the set of K-Lipschitz[4] functions.

To enforce Lipschitzness of the discriminator, Arjovsky et al. (2017) suggest weight clipping[5] as a measure of constraint the weights $\mathcal{W}$. However, they admit that this is a terrible way to enforce the Lipschitzness due to either slow convergence or vanishing gradients when the weights reach the boundary. Gradient penalty suggested by Gulrajani et al. (2017) is a way of assuring Lipschitzness, while avoiding vanishing gradients. An additional term penalizing large gradients is added to the value function 3.64.

Arjovsky et al. (2017) show that if $\mathbb{E}_{z \sim p_z(z)}[|z|] < \infty$, where z is the latent code as denoted earlier, then the Wasserstein distance is continuous everywhere and differentiable almost everywhere. On the other hand, this is not true for the Jensen-Shannon divergence used in the original GAN, due to a locally saturated discriminator may lead to vanishing gradients. Arjovsky et al. (2017) further conclude that the Wasserstein loss leads to a loss metric that correlates with the sample quality of the generator, and improved stability during the optimization. They also note that there is no evidence of mode collapse during their empiric experiments.

Loss Function

From the value function (3.64), we have that the discriminator will maximize the value function, whereas the generator will minimize it. If labeling the class negative/positive, for example such that $\mathcal{G}(z)$ is labeled -1 and x is labeled 1, the problem reduces to the simple, unbounded loss function:

$$\mathcal{L}_W = y \cdot \hat{p}. \tag{3.65}$$

[4] f is Lipschitz continuous if there exists a positive constant L such that $|f(x_1) - f(x_2)| \leq L|x_1 - x_2|$.

[5] The weights $\mathcal{W}$ is bounded by $w \in [a, a]$, $\forall\, w \in \mathcal{W}$, and will be ”clipped” to be within this range after updating.

3.6.5 Hyperparameter Tuning in Generative Adversarial Networks

An important part of obtaining well-performing neural networks is using the right hyperparameters. These are parameters that will not get optimized during the learning process of neural networks, but we rather have to tune them ourselves. There are however many optimization methods that can be used to find the best set of hyperparameters, such as grid search, random search, genetic algorithms (Liashchynskyi and Liashchynskyi (2019)) and Bayesian optimization (Frazier (2018)). In addition to common hyperparameters tuned in neural networks, such as number of training iterations, batch size, learning rate, number of layers, type of layers, number of nodes, optimizer, activation function and loss function. We also have GAN-specific hyperparameters, such as type and dimension of the latent code inputted to the generator, number of discriminator iterations per generator iteration and one can design the generator and the discriminator differently, with different batch sizes, learning rates, optimizers, network architecture, etc.

A study regarding how the hyperparameters affect the performance of various GAN architectures was carried out by Lucic et al. (2017). Here, the learning rate, the number of discriminator iterations per training iteration, the β in Adam (”The exponential decay rate for the 1st moment estimates”, Kingma and Ba (2014)), batch normalization layers and some more model-specific hyperparameters that will not be relevant for our models were tested. The findings were that some hyperparameters such as β of Adam (Kingma and Ba (2014)) did not significantly affect the performance, whereas other hyperparameters such as learning rate where found improving the performance in a smaller range $[10^{-4}, 10^{-3}]$.

They also found that the number of discriminator iterations, when testing 1 and 5 iterations, did not affect performance in any notably way, leading to the preference of $D_{iter} = 1$ due to drastically faster learning. However, as Goodfellow et al. (2014) discuss in the closing arguments, updating $\mathcal{G}$ without updating $\mathcal{D}$ enough will lead to mode collapse. Due to the importance of avoiding mode collapse in our experiments, we will investigate how the number of discriminator iterations per training iteration D_{iter} affects the performance.

The original paper introducing batch normalization (Ioffe and Szegedy (2015)) argues that reduction of the internal covariate shift is the reason for improved and accelerated convergence, however, newer findings suggest that this may not be the reason for the increased performance of batch normalization. Santurkar et al. (2018) shows that the distributional stability obtained by reducing internal covariate shift does not explain the improvements gained by using batch normalization, and instead argues that batch normalization smooths the objective function. This in turn increases the Lipschitzness of both the optimization function and the gradients, which makes the convergence faster and more stable (Santurkar et al. (2018)). However, they both agree that batch normalization in fact accelerates the convergence of the training, regardless of the reasons behind. Lucic et al. (2017) is more lukewarm about the performance of batch normalization in the GAN setting, finding that it depends on the GAN framework, and suggest that it is a hyperparameter worth exploring.

3.7 Recursive Multi-Step Forecast

There are many ways of forecasting, depending on the wanted outcome. With neural networks, it depends both on network structure and on the partition of the training data. The obvious forecast will be the one-step-ahead forecast, which for a neural network $f^{\hat{W}}(\cdot)$ is defined as:

$$\hat{y}_{t+1|t} = f^{\hat{W}}(y_t, \dots, y_{t-\ell}). \tag{3.66}$$

However, when wanting to forecast multiple steps ahead there are multiple ways to do this. One way can be to fit a model to forecast h-steps-ahead directly:

$$\hat{y}_{t+h|t} = f^{\hat{W}}(y_t, \dots, y_{t-\ell}), \tag{3.67}$$

where h is the forecast horizon. However, this model will not be as flexible if we want different forecast lengths, as we may have to train the model again. It is also possible to train the model to forecast all horizons up till h:

$$(\hat{y}_{t+h|t}, \dots, \hat{y}_{t+1|t}) = f^{\hat{W}}(y_t, \dots, y_{t-\ell}), \tag{3.68}$$

although this can complicate the optimization task substantially.

Another way to forecast multiple steps ahead is by recursively feed the previous prediction into the network as a previously observed value. The benefit of doing this is that the model is really flexible in terms of the forecast horizon. On the other hand, we may observe drifts over time, where a slight incorrect prediction accumulate to large errors. However, we will in this thesis use this method, but will observe the forecast error and uncertainty estimates as the forecast horizon increases.

Let us define the recursive multi-step forecast; let y_t denote the true value of the time series and $\hat{y}_t$ denote the predicted value. Then we have a chain of one-step-ahead forecasts in order to obtain the recursive multi-step forecast of $y_{t+h|t}$:

$$\begin{aligned}
\hat{y}_{t+1|t} &= f^{\hat{W}}(y_t, \dots, y_{t-\ell}) \\
\hat{y}_{t+2|t} &= f^{\hat{W}}(\hat{y}_{t+1|t}, y_t, \dots, y_{t+1-\ell}) \\
&\vdots \\
\hat{y}_{t+h|t} &= f^{\hat{W}}(\hat{y}_{t+h-1|t}, \hat{y}_{t+h-2|t}, \dots, y_{t+h-1-\ell}).
\end{aligned} \tag{3.69}$$

Notably, if $h > \ell$, then the input will only contain previously forecasted values.

Combining the recursive multi-step forecast method with the Monte Carlo sampling of forecast distribution, we have that a multi-step Monte Carlo sample of an observation y_{t+h} is given by:

$$\hat{y}^{(b)}_{t+h|t} = f^{\hat{W}}(\hat{y}^{(b)}_{t+h-1|t}, \hat{y}^{(b)}_{t+h-2|t}, \dots, y_{t+h-1-\ell}). \tag{3.70}$$

3.8 Performance Metrics

In this section we will introduce the performance measures we will use in this thesis. This includes point accuracy metrics (section 3.8.1, 3.8.2 and 3.8.3) and distribution accuracy metrics (section 3.8.4, 3.8.5, 3.8.6 and 3.8.7).

3.8.1 Mean Squared Error

Mean squared error (MSE) is a widely used benchmark metric for measuring regression error in statistics and machine learning, along with its root-transform, root mean squared error[6]. Similarly, it can be used in the forecasting setting by measuring the squared distance from a forecast to the actual observation. The MSE is given by:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \tag{3.71}$$

where $\hat{y}$ is the predicted value of y.

The benefit of using squared error measures, compared to absolute error measurements is the heavy penalty of large errors. The mean absolute error[7] will penalize two medium errors ($\epsilon_1 = \epsilon_2 = 1/2$) equally to one large error and one perfect fit ($\epsilon_1 = 1, \epsilon_2 = 0$). However, the MSE will penalize the inconsistency in a larger degree. This will in turn favor a consistent model over an inconsistent one, given similar absolute errors.

The MSE is scale-dependent, which means that comparing results for different data sets may not be very informative. However, it has the property of being symmetric, which in many cases is useful.

3.8.2 Symmetric Mean Absolute Percentage Error

Symmetric mean absolute percentage error (sMAPE) (Fildes and Armstrong (1979), p. 348) is another error measurement that is common in the forecasting domain, for example in the M4 forecasting competition (Makridakis et al. (2020)). Contrary to MSE, sMAPE is unit-free, so it can be used to compare the performance across different data sets. It is based on the mean absolute percentage error $\text{MAPE} = 100 \cdot \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i}$, however addresses the issue of MAPE where forecasts $\hat{y}_i$ lower than the actual value y_i are favored. Th sMAPE is given by:

$$\text{sMAPE} = \frac{200}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i + \hat{y}_i}, \tag{3.72}$$

and is 0 if $y_i = \hat{y}_i = 0$.

However, as suggested by Hyndman and Koehler (2006), there can occur problems for small values of y, as $\hat{y}$ also likely will be close to zero. In addition, the sMAPE can take

[6] $\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$

[7] $\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$

negative values, which does not coincide with "percentage error". A way to address the issues of negative values is to apply absolute values in the denominator:

$$\text{sMAPE} = \frac{200}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{|y_i| + |\hat{y}_i|}, \tag{3.73}$$

however, it will still be problematic around zero. Hyndman and Koehler (2006) argues further that the sMAPE is indeed not symmetric either, due to the denominator of sMAPE$(y, \hat{y} = y + a)$ not being equal to the denominator of sMAPE$(y, \hat{y} = y - a)$, where $a \neq 0$, and further introduces a new forecasting measure, mean absolute scaled error (MASE), which avoid those problems.

While sMAPE is a widely used measure of the forecast error, it also has a rather circumstantial usage, where the interpretation of the relationship of the time series quantity is important. In the general setting, forecasting 2 units for the true value of 1 unit will yield an sMAPE of 67%. To achieve the same sMAPE for a true value of 20 units, the forecast has to be either 10 or 40 units. In cases where the percentage relationship between the quantities is relevant, this percentage behavior is a desired property. For example, we often denote a sale discount as the percentage off the original price, and therefore a percentage error measure is useful. On the other hand, the weather temperature is a quantity where we will not consider the two aforementioned forecast errors as equally precise. Consequently, we have to consider when to rely on this forecast error measurement based on the quantity at hand.

3.8.3 Mean Absolute Scaled Error

Hyndman and Koehler (2006) introduces a forecasting metric that scales according to some benchmark method. Thus, it should be comparable across data with different scales. It is used together with the sMAPE (3.8.2) in the M4 forecast competition. The MASE is the mean absolute forecast error scaled with the in-sample mean absolute error of the naive last-day forecast (3.2) for non-seasonal data:

$$\text{MASE} = \frac{\frac{1}{h}\sum_{t=T+1}^{T+h}|y_t - \hat{y}_t|}{\frac{1}{T-1}\sum_{t=2}^{T}|y_t - y_{t-1}|}, \tag{3.74}$$

with the $t = 1, \ldots, T$ being training data and h being the forecast horizon. However, as we have seasonal data, this may be very misleading. Therefore, one can use the naive last-season forecast (3.9) as the scale. The seasonal mean absolute scaled error is given by:

$$\text{MASE} = \frac{\frac{1}{h}\sum_{t=T+1}^{T+h}|y_t - \hat{y}_t|}{\frac{1}{T-m\cdot(k+1)}\sum_{t=m\cdot(k+1)+1}^{T}|y_t - y_{t-(k+1)\cdot m}|}, \tag{3.75}$$

where $k = \text{int}((h-1)/m)$. As it is scaled by a naive forecast, the value is immediately easier to interpret. Values larger than 1 indicate poor performance in terms that it has a higher mean absolute error than the naive method.

3.8.4 Prediction Interval Coverage

A common metric to assess the uncertainty of a prediction is the prediction interval. It is defined by a coverage probability $(1-\alpha)\cdot 100\%$, that represents the probability of the prediction interval covering the next observation $P(l < y < u) = 1-\alpha$, where l is the lower bound and u the upper bound of the prediction interval.

Under the assumption of Gaussian distributed noise, which in many cases is a reasonable assumption and an often used assumption, the estimated $(1-\alpha)\cdot 100\%$ prediction interval can be calculated by:

$$[\,\hat{y}_{t+1|t} - z_{\alpha/2}\cdot\hat{\sigma}_{t+1|t},\;\; \hat{y}_{t+1|t} + z_{\alpha/2}\cdot\hat{\sigma}_{t+1|t}\,], \tag{3.76}$$

where $\hat{y}_{t+1|t}$ is the estimated value of y_{t+1}, $\hat{\sigma}_{t+1|t}$ is the estimated forecast standard deviation and $z_{\alpha/2} = \Phi\Big(1-\frac{\alpha}{2}\Big)$, where Φ is the cumulative distribution function of a standard Gaussian distribution.

Avoiding the assumption of Gaussian distributed noise, one can find the prediction interval by the quantiles of the forecast distribution:

$$[\,q_{\alpha/2},\;\; q_{1-\alpha/2}\,], \quad 0 < \alpha < 1, \tag{3.77}$$

where $q_p = \inf\{y \in \mathbb{R} : p \leq F(y)\}$ is the discrete quantile function and F is the cumulative forecast distribution. In the numeric setting, one can sample from the estimated forecast distribution $\hat{f}$, and then estimate the quantiles of the estimated forecast distribution discussed in section 3.6.2.

When the prediction interval is estimated, the coverage is the proportion of samples covered by the prediction interval, and given by:

$$c = \frac{1}{N}\sum_{i=1}^{N} I(l_i < y_i < u_i), \tag{3.78}$$

where $I(\cdot)$ is the indicator function. The coverage is a useful metric for evaluating the performance of prediction intervals, however as concluded by Askanazi et al. (2018), defining the "best" prediction interval is an ambiguous task. In most settings, one can measure the distance from a target, and then assess the quality of the prediction. For prediction intervals, however, it is not that easy. As discussed by Askanazi et al. (2018), one can satisfy the "Christoffersen conditions" (CC) that requires $P(l < y < u) \geq 1-\alpha$. Thus the optimal prediction interval can be calculated as $\min\limits_{u,l}(u-l)$ st. $P\big(y \in [l,u]\big) = 1-\alpha$. However, as Askanazi et al. (2018) further argue, when comparing prediction intervals we would then write off any prediction interval estimate with coverage lower than the coverage probability. In addition any prediction interval, regardless of the width, satisfying the CC will be better than all prediction intervals not satisfying the CC, regardless of how close it is to satisfy the CC. As this seems unreasonable, Askanazi et al. (2018) discuss measurements that trade off the coverage against the width of the prediction intervals, while still heavily favors prediction intervals that satisfy the CC. One of which will be discussed in the next section.

When we are going to investigate the results, it is not an unambiguous interpretation of the best performing model in terms of prediction interval coverage. We rather have to analyse additional factors, such as the prediction interval width, to assess the quality of the prediction intervals. The correct trade-off between coverage and prediction interval width may change depending on the application and the data set at hand. If the application requires certainty of the prediction interval coverage, satisfying the ”Christoffersen conditions” may be required. When interpreting the results in this thesis, we have no specific application in mind and must assess a more nuanced conclusion regarding the performance.

3.8.5 Mean Scaled Interval Score

The mean scaled interval score (MSIS) (Gneiting and Raftery (2007)) is a metric used in the M4 forecasting competition (Makridakis et al. (2020)) to measure the performance of the prediction intervals. It is heavily based on the Winkler score (Doraiswami (1977)), which is commonly used for prediction interval evaluation. It is given by:

$$\text{Winkler score} = |u - l| + \frac{2}{\alpha}(l - y)I(y < l) + \frac{2}{\alpha}(y - u)I(u < y), \tag{3.79}$$

where l, u is the lower and upper bound of the $(1 - \alpha) \cdot 100\%$ prediction interval. The Winkler score is given by the width of the prediction interval, however, if the prediction interval does not cover the actual value, it is added a penalty corresponding to the distance from the nearest bound to the actual value. This distance is scaled by a factor of $2/\alpha$, ensuring that too narrow prediction intervals are heavily penalized for any value outside the prediction interval. As a reference, the scale for a 95% prediction interval will be $2/0.05 = 40$, ensuring that prediction intervals satisfying or almost satisfying the CC are favored.

The mean scale interval score, similarly to MASE (section 3.8.3) is scaled according to the naive one-season-ahead forecast (3.9):

$$\text{MSIS} = \frac{1}{h}\frac{\sum_{i=t}^{t+h} |u_i - l_i| + \frac{2}{\alpha}(l_i - y_i)I(y_i < l_i) + \frac{2}{\alpha}(y_i - u_i)I(u_i < y_i)}{\frac{1}{N-m}\sum_{i=m+1}^{N} |y_t - y_{t-m}|}, \tag{3.80}$$

where h is the forecast horizon.

3.8.6 Kullback-Leibler Divergence

The Kullback-Leibler divergence is a popular measure of the similarity between a probability distribution relative to a reference probability distribution. The Kullback-Leibler divergence (D_{KL}) is non-negative (≥ 0) and $D_{KL} = 0$ indicates that a probability distribution is equal to the reference distribution, and given by:

$$D_{KL}(\text{P} \,||\, \text{Q}) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{3.81}$$

The Kullback-Leibler divergence is asymmetric, which means that $D_{KL}(\text{P} \,||\, \text{Q}) \neq D_{KL}(\text{Q} \,||\, \text{P})$. In addition, it requires that for all x, $Q(x) = 0$ implies $P(x) = 0$. In addition, we can ignore $P(x) = 0$, as we have that $\lim_{x \to 0^+} x \log(x) \to 0$.

3.8.7 Jensen-Shannon Divergence

The Jensen–Shannon divergence is also a similarity measure of two probability distributions, derived from the Kullback Leibler divergence. However, instead of comparing the probability distribution to a reference distribution, both distributions are compared to a mixture distribution M. This introduces some useful properties, which prompt us to use the Jensen-Shannon divergence as a similarity measure. The Jensen-Shannon divergence is given by:

$$D_{JS}(\mathrm{P} \,||\, \mathrm{Q}) = \frac{1}{2} D_{KL}(\mathrm{P} \,||\, \mathrm{M}) + \frac{1}{2} D_{KL}(\mathrm{Q} \,||\, \mathrm{M}) \tag{3.82}$$

where $M = \frac{1}{2}(P + Q)$. Firstly, we have that $D_{JS}(\mathrm{P} \,||\, \mathrm{Q}\ ) = D_{JS}(\mathrm{Q} \,||\, \mathrm{P})$, in other words it is a symmetric distance metric. Similar to the Kullback-Leibler divergence, the Jensen-Shannon divergence is non-negative, but is also bounded. If one uses the natural logarithm (with e as base), we have that:

$$0 \leq D_{JS}(\mathrm{P} \,||\, \mathrm{Q}) \leq \ln(2). \tag{3.83}$$

Chapter 4

Experimental Setup

In this chapter, we will go through how the experiments are conducted. The theory behind the methods used are presented in chapter 3, but many of the models are used through the packages `keras` (Chollet et al. (2015)), `pmdarima` (Smith et al. (2017-)) and `statsmodels` (Seabold and Perktold (2010)). The use of those packages will reduce the risk of error, simplify code and reduce the time necessary to develop the code. We will go through the data sets used for experiments, model setup and present some hypotheses related to the experiments. The code necessary for running the experiments is available at `https://github.com/mattaop/ForGan/tree/Release/1.0`.

4.1 Data

4.1.1 Data Processing

The data is split into a training set (first t observations) and a test set (last $T-t$ observations), where the test set is exclusively used for obtaining unbiased results of the forecasts. When training models where hyperparameter tuning is not part of the ordinary training process, or model selection has to be done on a separate set of data, a small percentage of the last training observations will be used as a validation set. This can discover overfitting of the training, and still not compromising the test results.

The data is scaled using `scikit-learn` MinMaxScaler, which transforms the data in the range of $[0,1]$. As multiplicative versions of exponential smoothing models (3.15) are not usable with zero values, the scaling for exponential smoothing will be in the range $[1,2]$ instead. The scaling is done according to the range of the training data, and then the test data is transformed according to the same scale, in order to not use information that should not be available. This means that one can obtain test data outside the scale range. When calculating metrics, the data is transformed back, in order to obtain error measurements in the same scale as the original data. It is worth noting that the coverage is scale-independent.

For neural networks, the data is transformed into overlapping sequences of a specific length ℓ. The responding target is thus given by the next value in the time series:

$$\begin{aligned} X_1 &= [y_1, y_2, ..., y_\ell], \quad Y_1 = [y_{\ell+1}] \\ X_2 &= [y_2, y_3, ..., y_{\ell+1}], \quad Y_2 = [y_{\ell+2}] \\ &\vdots \\ X_{T-\ell} &= [y_{T-\ell}, y_{T-\ell+1}, ..., y_{T-1}], \quad Y_{T-\ell} = [y_T]. \end{aligned} \tag{4.1}$$

4.1.2 Distribution Estimation

The first experiment is done on two synthetic data sets draw from a Gaussian distribution, $x \sim \mathcal{N}(\mu, \sigma)$, and a bimodal distribution, $x \sim \mathcal{N}(\mu_1, \sigma_1)$, $\mathcal{N}(\mu_2, \sigma_2)$. This is not time dependent data sets, however we want to verify the distribution estimation properties of generative adversarial networks.

For the Gaussian distribution, we sample 5000 observations from the known distribution, $\mathcal{N}(0, 0.1)$, which is used as the "real" data. This distribution is not conditional, as the time series are, so we can compute the Jensen-Shannon divergence (3.8.7) between the real and the generated data. We can also compare the estimated standard deviation $\hat{\sigma}$ to the known standard deviation of the data, $\sigma = 0.1$. Figure 4.1 shows a silhouette plot of the sampled distribution.

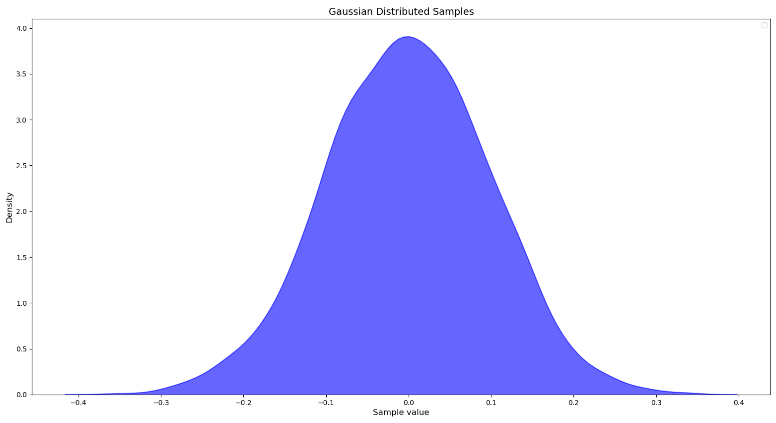


Figure 4.1: The figure shows the distribution of 5000 samples drawn from a Gaussian distribution, $\mathcal{N}(0, 0.1)$.

The bimodal distribution is defined as $f(x) = p \cdot f_1(x) + (1 - p) \cdot f_2(x)$, where f_1 and f_2 are the first and second probability distribution respectively and p is the mixing parameter. In practice, we sample from probability distribution $\mathcal{N}(\mu_1, \sigma_1)$ with probability p and from probability distribution $\mathcal{N}(\mu_2, \sigma_2)$ with probability $1 - p$. The parameters used will be

$\mu_1 = -0.5, \sigma_1 = 0.1,\ \mu_2 = 1, \sigma_2 = 0.2$, and 5000 samples from this bimodal distribution is shown in figure 4.2.

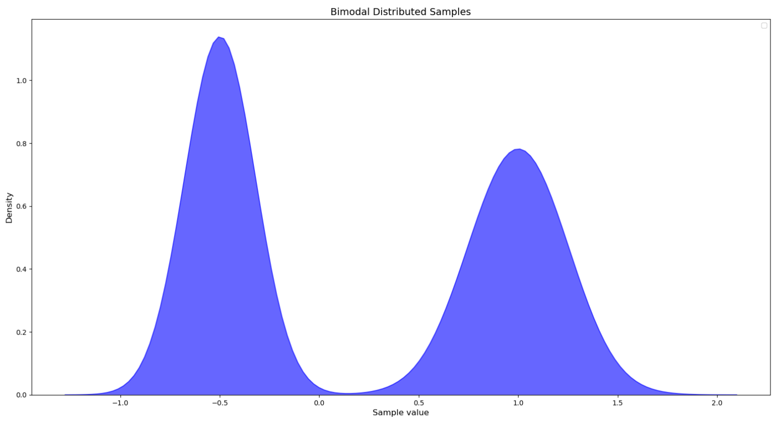


Figure 4.2: The figure shows the distribution of 5000 samples drawn from a bimodal distribution, $\mathcal{N}(-0.5, 0.1), \mathcal{N}(1, 0.2)$.

4.1.3 Sine Curve with Gaussian Noise

To investigate the performance of the GAN in the forecasting setting, we will introduce data that has some time dependency. We will still control the noise with a known probability distribution as earlier. The data will therefore consist of a seasonal component, a sine function, and a noise component, Gaussian noise. Thus, the t observation y_t will be given by:

$$y_t = \sin\left(\frac{\pi}{6}t\right) + \epsilon, \tag{4.2}$$

where $t = 1, 2, 3, ..., \epsilon \sim \mathcal{N}(0, 0.1)$ and $\pi/6$ converts the seasonality to 12, corresponding to monthly data.

As we know the underlying model, as well as the noise component, we can define the optimal forecast model as the sine component, with expected mean squared error (MSE):

$$\mathbb{E}[MSE] = \mathbb{E}\left[\frac{1}{N}\sum_{t=1}^{N}\left(A\sin\left(\frac{\pi}{6}t\right) + \epsilon - A\sin\left(\frac{\pi}{6}t\right)\right)^2\right] = \frac{1}{N}\sum_{t=1}^{N}\mathbb{E}[\epsilon^2] = \sigma_\epsilon^2 = 0.01. \tag{4.3}$$

This result can be used both to compare the MSE of the models against the optimal forecast model and compare the estimated standard deviation of the models to σ_ϵ . It is worth noting that for the estimated standard deviation $\hat{\sigma}$ should reflect both the noise, the model variance and the model bias, so we want $\hat{\sigma} \rightarrow \sigma_\epsilon^+$ as $\text{MSE}\big(f^{\hat{W}}(\cdot)\big) = \text{Var}\big[f^{\hat{W}}(\cdot)\big] + \text{Bias}\big[f^{\hat{W}}(\cdot), f^{W}(\cdot)\big]^2 + \sigma_\epsilon^2 \rightarrow \sigma_\epsilon^{2+}$.

We can also derive some theoretical quantities from the mean absolute scaled error (MASE). As shown in section 3.8.3, the MASE is scaled by the mean in-sample error of the naive forecast. For the sine curve with Gaussian noise, the expected scale can be computed as the mean absolute difference, $\mathbb{E}(MD) = \mathbb{E}[|X - Y|] = \frac{2}{\sqrt{\pi}}\sigma$, where $X, Y \sim \mathcal{N}(\mu, \sigma)$. Further we can compute the expected mean absolute deviation (MAD), of the optimal forecast model, $\mathbb{E}(MAD) = \frac{\mathbb{E}[|X|]}{\sqrt{\mathbb{E}(X^2)}} = \sqrt{\frac{2}{\pi}}\sigma$, where $X \sim \mathcal{N}(\mu, \sigma)$. Thus, we can compute the theoretical optimal obtainable mean scaled absolute error for the sine curve with Gaussian distributed noise:

$$\mathbb{E}(\text{MASE}) = \frac{\frac{\mathbb{E}[|X|]}{\sqrt{\mathbb{E}(X^2)}}}{\mathbb{E}[|X - Y|]} = \frac{\sqrt{\frac{2}{\pi}}\sigma}{\frac{2}{\sqrt{\pi}}\sigma} = \frac{\sqrt{2}}{2} \approx 0.7071. \tag{4.4}$$

This can be used as a comparison for the models when forecasting the sine time series.

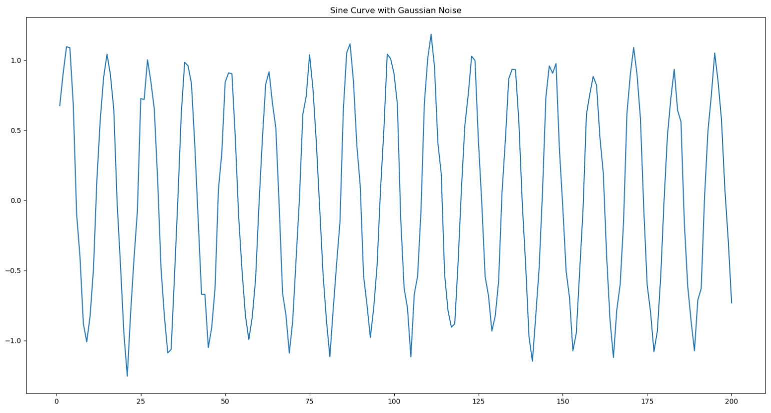


Figure 4.3: Plot that shows the first 200 of 5000 observations of sine data with added Gaussian noise.

The data set consists of 2000 observations according to the sine data (equation 4.2), $\{y_1, \dots, y_{2000}\}$, where the first 60% of the observations $\{y_1, \dots, y_{1200}\}$ will be the training set and the remaining 40% of the observations $\{y_{1201}, \dots, y_{2000}\}$ will be the test set. In addition, for the Monte Carlo dropout and the ForGAN, we want to tune hyperparameters using a validation set. The validation set will be the last 12.5% of the training observations, corresponding to $\{y_{1051}, \dots, y_{1200}\}$. In addition, using the sliding window approach for neural networks (equation 4.1), we will have $\ell = 24$ corresponding to a look-back window of two seasons.

4.1.4 Oslo Temperature Data Set

The Oslo temperature data set (Meteorologisk Institutt (MET) (2019)) contains minimum, mean and maximum air temperature for each month from February 1937 to October 2019 in Oslo. In this thesis, the mean temperature will be used as a univariate time series, and as the nature of the data is similar to the sine data with a distinct seasonality of 12, we will not diverge much from that approach.

The data set contains 993 observations, and dividing the data set into 72% training set, 8% validation set and 20% test set, we obtain training data of size 715, validation data of size 80 and test data of size 198. Further, we will use the sliding window approach (4.1), with sequence length $\ell = 24$ corresponding to two seasons, resulting in 692 training examples.

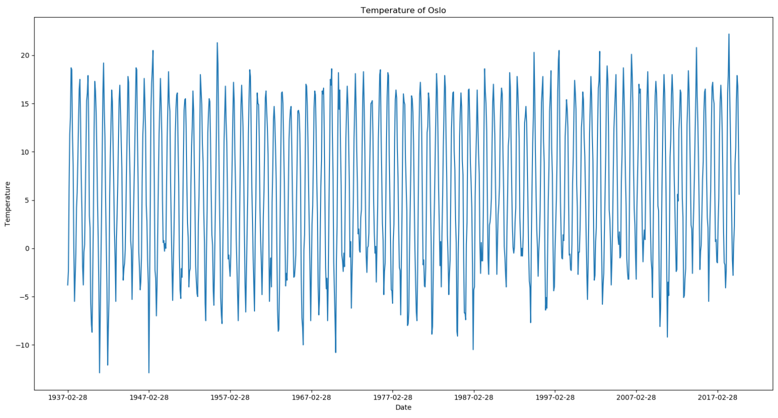


Figure 4.4: Plot that shows the mean temperature each month in Oslo from February of 1937 to October of 2019.

Figure 4.4 shows the data set and shows distinct seasonal behavior as a cyclic function corresponding with a yearly seasonality of 12 months.

As discussed in section 3.8.2, the temperature is a quantity where the sMAPE can be quite misleading. We will therefore not rely too much on this error metric when assessing the point forecast accuracy for this data set.

4.1.5 Avocado Price Data Set

The Avocado Price data set is a time series data set published through Kaggle by the Hass Avocado Board (2018). It contains the average weekly price for both conventional and organic avocado from 53 different regions in the United States (US), as well as an average price for all of the US. The prices span from the start of 2015 and through March 2018, resulting in 169 observations for each avocado type in each region. Given that there are 53 regions and an average price series, with data for both conventional and organic avocado, there are 108 different time series, resulting in $108 \cdot 169 = 18252$ observations. In addition, there are multiple regressors beyond the price available, such as sales volume, both total and for three different avocado sizes. However, as the scope of this thesis is univariate time series, we will only utilize the weekly average avocado price.

Based on domain knowledge, we would expect a yearly seasonality, as the demand and availability of avocados would depend on the season. Where the avocados are possible to grow would differ from the time of the year, and it is reasonable that this will affect the availability and the price. Further, there may also be a higher demand at certain times of the year. As the data is the weekly average price, we will implement the seasonal ARIMA model with seasonality $s = 52$.

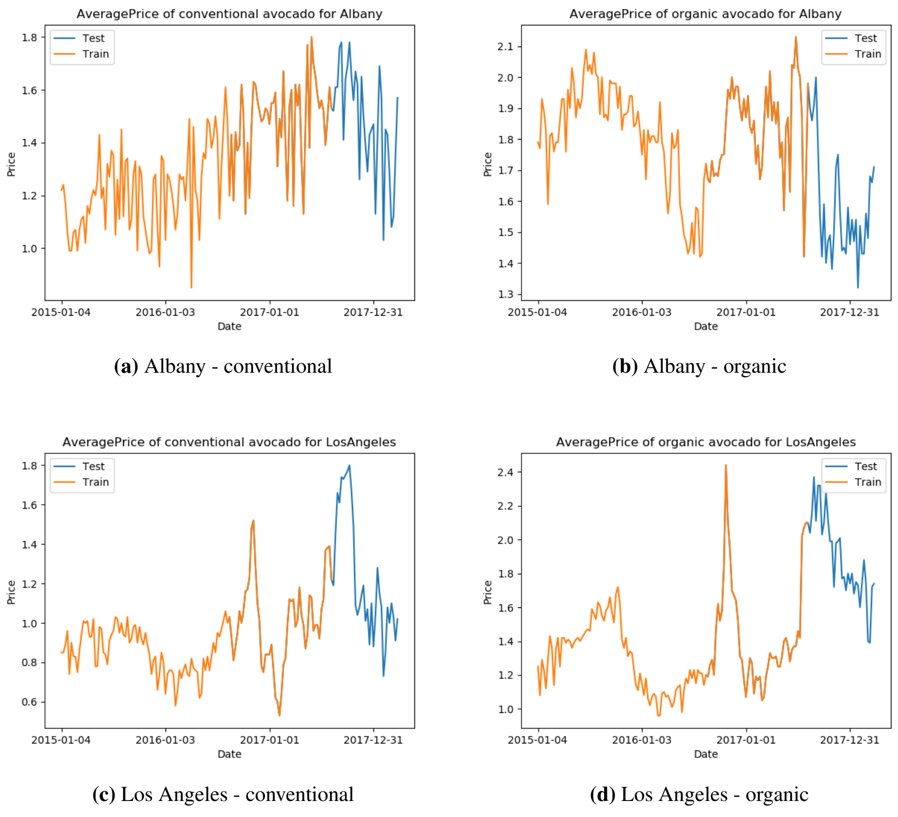


Figure 4.5: Examples of time series from the avocado price data set. Each time series is divided into a training set (orange) and a test set (blue).

Each time series will be divided into training, validation and test data as before, resulting in 123 training samples, 13 validation samples and 33 test samples per time series. In addition, we will use the sliding window technique (equation 4.1) to split the data into equal-length sequences for the neural networks. We will try some different sequence lengths, as the networks may pick up seasonal dependencies, however long window lengths will decrease the number of training samples. We will utilize a forecast horizon of $h = 13$, so we obtain enough forecasts of each forecast horizon.

Figure 4.5 shows the weekly price for conventional and organic avocado in two different regions. As argued earlier, an expectation is some sort of yearly seasonality, however, the seasonality is not as distinct as observed in the temperature data set. However, it is indeed possible to see some yearly fluctuations, especially in Los Angeles prices. Here the price seems to be at its highest during the fall, whereas after the new year, in the months of January and February the price is at is yearly lowest. In addition, we see some spikes indicating some sort of extreme event which may not be explained by past values of the price. However, these spikes are not that present in the Albany prices. In addition, looking at the price of the organic avocado in Albany (figure 4.5b), we see some seasonal behavior. If we observe where the lowest prices are positioned, we see the first in January 2015, the following during the summer of 2016 and finally late 2017. This contradicts the theory of a yearly seasonality, which may make it harder for the models to capture the cyclic behavior.

For this data set, the sMAPE can be a more suitable measure of the performance. As discussed in section 3.8.2, price is a quantity where we often denote differences as a percentage.

4.1.6 Electricity Consumption Data Set

The electricity consumption data set was found through the UCI Machine Learning Repository (Dua and Graff (2017)) and contains the electricity consumption in kW per 15 min of 370 consumers over three years. In this thesis, the data set has been resampled to contain hourly consumption in kWh. In addition, we have taken a fairly small subset of the data set: only one consumer for two months. This is due to available computational power and time, and thus we have the data set shown in figure 4.6. The time series contains 1417 observations, where the last 20%, corresponding to 283 observations, will be used for test data. This leaves us with 1134 training examples, where for neural networks we will use the last 10% of the training data as a validation set.

The data seem to have a strong seasonal effect every 24 observations, corresponding to a daily seasonality. Notably, we can observe an additional seasonal effect corresponding to weekly seasonality, where there is a higher electricity consumption two of the days in a week. This can be due to weekends, where if this is a private household, one would expect the residents to be more at home during the day. However, this characteristic is not as prominent in the last part of the time series, which in our case is the test set. However, we will omit this knowledge in order to uphold the independence of the test set.

Similarly to the avocado price, the electricity consumption is a quantity where one may be inclined to use a percent-wise error. It depends on the goal of the forecasting, where if one wants to model the spikes accurately, a non-percentage error is to be preferred. On the other hand, if the goal is to estimate the income, sMAPE may be preferred.

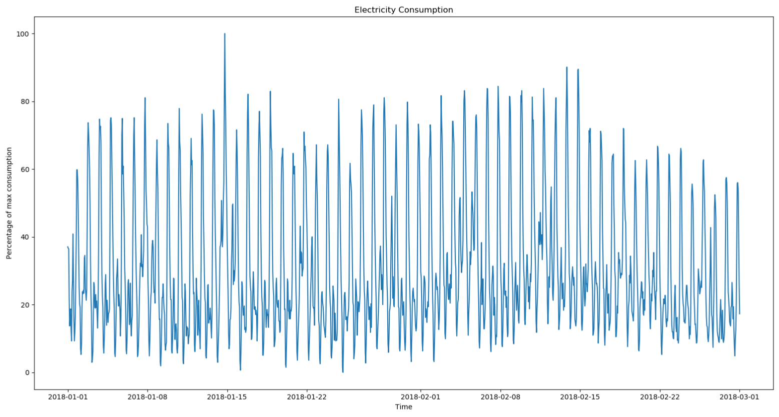


Figure 4.6: Plot that shows the hourly electricity consumption of a consumer.

4.2 Models

The models used in the thesis are based on the ForGAN (section 3.6.2), proposed by Koochali et al. (2019), which shares the architecture with the CGAN setup (Lucic et al. (2017)). We will test both the standard binary cross-entropy loss and the Wasserstein loss in the ForGAN architecture. The Monte Carlo Dropout (section 3.5.2) method (Gal and Ghahramani (2015), Zhu and Laptev (2017)) is used as a state-of-the-art model for forecast uncertainty in neural networks. In addition, the performance will be compared to well known models for time series modeling and forecasting, seasonal ARIMA (section 3.3) and Exponential smoothing (section 3.2).

4.2.1 Baseline Models

The baseline models is a SARIMA model, obtain by the `auto_arima` function from `pmdarima` (Smith et al. (2017-)), and a Holt-Winters' exponential smoothing from `statsmodels` (Seabold and Perktold (2010)). Model selection based on training data will be done for both methods. For SARIMA models using `auto_arima`, the function will fit models in a range of values for p, d, q, P, D, and Q in a seasonal ARIMA(p,d,q)x(P,D,Q)$_s$, and choose the best performing model based on the lowest corrected Akaike information criterion (AICc (3.21)). For Holt-Winters' exponential smoothing, we have the option of having the trend and/or season as a multiplicative or additive. Model selection is not done automatically by `statsmodels`, so we have therefore created a custom loop where models with each combination is fitted similarly to `auto_arima`. The best performing models is chosen based on the lowest AICc.

The Monte Carlo dropout methodology was developed as a project thesis (Opland (2020)) and contains one layer of either the Simple RNN or LSTM exclusively (and will for a further reference to be named as such), with one hidden dense layer and a dense output layer. Dropout is applied to all layers, both during training and test. Hyperparameters such as number of nodes, batch size, learning rate and training iterations are chosen based on best performance on a hold-out validation set, and are shown in the appendix for reproducibility.

4.2.2 Generative Adversarial Networks

The ForGAN (Koochali et al. (2019)) framework presented in section 3.6.2, where the generator is trained to generate the forecast distribution of y_{t+1} given $(y_t, y_{t-1}, \ldots, y_{t-\ell})$, is utilized. The first layer of both the generator $\mathcal{G}$ and the discriminator $\mathcal{D}$ is a recurrent layer, SimpleRNN or LSTM, with a given number of nodes. The default activation function(s) are used in the case of recurrent layers (more details in section 3.4.1 and 3.4.2). Batch normalization is applied to the output of the first layer (Ioffe and Szegedy (2015)) of the discriminator. For the generator, the latent code input is concatenated to the output of the recurrent layer, introducing stochasticity after the initial assessment of the temporal dependencies. For the discriminator, the either true next value or the generated forecast is concatenated before the recurrent layer, to model the temporal dynamics between the previous observed values and the next.

The second layer is a dense layer with activation function ReLU (3.34) for the generator and Leaky ReLU (3.35) for the discriminator, which is based on observation of training stability done by Radford et al. (2015). The third layer is a dense layer where there is one output node, and the activation function is linear ($g_{\text{linear}}(x) = x$) for the generator in order to forecast values from $\mathbb{R}$.

In the case of the standard GAN framework (section 3.6), the discriminator has sigmoid activation function (3.36) to classify real samples as ”1” and fake samples as ”0”. For the Wasserstein GAN (3.6.4, which is unbounded, the output function of the discriminator is linear, and represents how much the discriminator believes the sample to be either real or fake. Real samples are therefore denoted ”1” and fake samples denoted ”-1”. Either Weight clipping or gradient penalty is applied to the discriminator in order to enforce Lipschitzness, as discussed in section 3.6.4.

Training a generative adversarial network using `keras` requires a custom training loop outside the usual `model.fit`. The training algorithm is described by algorithm 1, and shows how the discriminator $\mathcal{D}$ is first trained to label real time series as real, then forecasted values from the generator $\mathcal{G}$ is trained to be labeled false. This is done D_{iter} times, which is a hyperparameter we will investigate later. After training the discriminator, the generator $\mathcal{G}$ is trained to make the discriminator label the forecast as real.

Hyperparameter Tuning

To obtain a well-performing model, hyperparameter tuning is important. In section 3.6.5 we presented typical hyperparameters that need tuning in order to achieve well-performing models. While Lucic et al. (2017) investigate the performance of different hyperparameters in the GAN setting, the task and objective are somewhat different. In the image generating setting, avoiding mode collapse is not as essential as for forecasting uncertainty. In addition, the GAN architecture is quite different, with convolutional cells in the network layers and with a bounded output function, whereas we have an unbounded output range and recurrent cells. It is strenuous to archive optimal performance, however finding a set of hyperparameters suitable for the task should be within our reach. We will present the hyperparameters to be investigated in the results.

The batch size refers to the number of samples trained simultaneously. Whereas in the standard neural network setting, common knowledge among researchers is that large batch training leads to poor generalization (Yao et al. (2018)). As we will present in the hypothesis (section 4.3), due to the distribution estimation task, we suspect that there is a conflicting interest between generalization and distribution estimation accuracy.

Further, the learning rate is a hyperparameter frequently tuned in order to achieve convergence. Too low learning rate can lead to the weights getting stuck in local minimums, whereas too large learning rates can lead to divergence. The learning rate is often highly interactive with the batch size, optimization function, number of training samples and number of training iterations, so it has to be optimized over each task.

The number of training iterations refers to how many times the neural networks will backpropagate over the batches. The goal is to stop training when the networks have trained sufficiently such that they can generalize the task, however not starting to pick up random noise in the training data, namely overfitting. We will monitor the performance in mean squared error and coverage of the prediction intervals on a hold-out validation set during training, such that we will detect when the models start overfitting the training data.

The choice of network architecture is an important task of hyperparameter tuning. For temporal data, recurrent layers such as SimpleRNN (section 3.4.1) and LSTM (section 3.4.2) are suitable. We will test out these two cells in the first layer of both the generator and the discriminator, as well as tuning the number of nodes in each of these layers. We have chosen to have more nodes in the discriminator, with the goal of having a discriminator able to model the time series as least as good as the generator to avoid mode collapse.

The latent code dimension and the number of discriminator iterations per training iteration D_{iter} are specific hyperparameters to the GAN framework. As there is not as much research on how these parameters affect the performance, we will investigate them in chapter 5. D_{iter} may be a way to avoid mode collapse, as discussed in section 3.6.5, and will therefore be especially interesting when estimating the forecast uncertainty.

The loss function is also a hyperparameter which can be tuned. We have introduced the standard GAN framework with a binary cross-entropy loss, as well as the Wasserstein loss. We will for the synthetic data present comparisons between using these losses, while for the real time series data sets this hyperparameter is tuned similarly to the others, where the best performing model with respect to the hold-out validation set is chosen.

4.2.3 Monte Carlo Forecasting

In order to estimate the forecast distribution, Monte Carlo sampling from the generator is utilized. By sampling the latent code $Z = (z_1, \ldots, z_k)$, where $z_j \sim p_z(z)$, we can sample forecast values $\hat{y}_{t+1}$ from the generator $\mathcal{G}(Z|y_t, \ldots, y_{t-\ell})$. By sampling $B = 1000$ samples from the generator $\mathcal{G}$, we estimate the point forecast by the distribution mean $\overline{\hat{y}}_{t+1}$

(3.60) and the $(1-\alpha)\cdot 100\%$ prediction intervals by $[\,\hat{q}(\frac{\alpha}{2}),\ \hat{q}(1-\frac{\alpha}{2})\,]$. The procedure is shown in algorithm 2.

The forecast is then fed back into the network in order to provide the next forecast according to the recursive multi-step framework (3.69). We will do a recursive multi-step forecast up to a specific forecast horizon, then provide the next test value and repeat. This way we can obtain the coverage as a function of each forecasting horizon, and see if the coverage changes over forecast horizon. Thus, we can investigate if the coverage of the prediction interval for forecast $\hat{y}_{t+1|t}$ differ from the coverage of the prediction interval for forecast $\hat{y}_{t+2|t}$ or $\hat{y}_{t+h|t}$.

Algorithm 2: Monte Carlo Forecast using the generator $\mathcal{G}$ to estimate the forecast distribution.

```
Input: Condition window $\{y_t, y_{t-1}, y_{t-2}, ..., y_{t-\ell}\}$
Output: Prediction mean $\bar{\hat{y}}_{t+h|t}$, $(1-\alpha)\cdot 100\%$ prediction interval
B: number of Monte Carlo forward passes
H: max forecast length
for b in B do
    for h in H do
        Sample latent code $Z^{(b)} = (z_1^{(b)}, \dots, z_k^{(b)})$, where $z_j^{(b)} \sim p_z(z)$
        $\hat{y}^{(b)}_{t+h|t} = \mathcal{G}(Z^{(b)} \,|\, \hat{y}^{(b)}_{t+h-1|t}, ..., y_{t+h-\ell})$
    end
end
Prediction mean $\bar{\hat{y}}_{t+h|t}$ is computed according to equation 3.60.
Estimated $(1-\alpha)\cdot 100\%$ prediction interval is computed according to equation 3.62.
```

4.3 Hypotheses

Before we analyze the results, we will present some hypotheses related to the experiments and what kind of results we expect. These hypotheses also motivate the tuning and interpretation of some of the hyperparameters, as we will show in the next chapter.

As the generator is trained to fool the discriminator, not minimize the forecast error, the optimization of the forecast error depends on the strength of the discriminator as well. Further, the gradients of the parameters will depend on the discriminator's ability to distinguish the samples, meaning that it may take longer before convergence. A measure to improve this performance is the amount of discriminator training iterations D_{iter}, which we will investigate. Due to not directly minimizing the forecast error, it will be natural to think that the GAN avoids, at least to some degree, overfitting. We will also expect that the GAN will be trained for more iterations than the MC dropout, which explicit minimizes the forecast error.

The standard GAN can experience a saturated discriminator, which can lead to convergence problems. The Wasserstein GAN however, should not encounter these problems. Further, Wasserstein loss should have improved stability (section 3.6.4), which will further favor the Wasserstein GAN. As reported by Arjovsky et al. (2017), they did not observe any mode collapse, which was a motivation for utilizing the Wasserstein loss for estimating the forecast uncertainty. We will therefore expect better performance with a Wasserstein loss, where it is able to asses the prediction intervals with more correct coverage.

Further, we think that large batch sizes can increase the accuracy of the distribution estimation, as the discriminator will have a larger sample size of the distribution to compare with. However, small batch sizes lead to better generalization (Yao et al. (2018)). Therefore, we might observe conflicting interests, where one wants to balance the size of the batch size to accurately estimate the forecast distribution, while still being able to generalize the forecasting task. We will therefore expect larger batch sizes for the generative adversarial network than the MC dropout.

Regarding the data sets we will be using, it is hard to make very specific hypotheses of the performance. The synthetic sine time series holds simple temporal dynamics, which the statistical models are able to model explicitly. It should therefore be expected that these models performs better than the complex neural networks with thousands of parameters, which often pick up noise in the training data, a form of overfitting. For the real time series data sets we do not know the temporal dynamics, and can therefore not disregard the statistical models. However, the neural networks can model complex time series dynamics if trained correctly. Neural networks often require a lot of training samples to distinguish noise from temporal dynamics, though ”a lot” will differ from time series to time series.

Chapter 5

Results and Discussion

This chapter will contain results using the models presented in chapter 3 and the experimental setup described in chapter 4. Each section will present results regarding a specific data set; Gaussian and bimodal distribution (section 4.1.2), sine curve with Gaussian noise (4.1.3), Oslo temperature data set (4.1.4), avocado price data set (4.1.5) and the electricity consumption data set (4.1.6). The validation results will be used for tuning and investigating the hyperparameters in the MC dropout and ForGAN models, whereas the test results are used to compare and interpret performance between the proposed ForGAN model, the MC dropout model and the baseline models. The results are discussed in each section, and a short summary of the discussion is provided at the end. Finally, we will summarize the findings and discuss how they relate to the hypotheses.

5.1 Distribution Estimation

Generative adversarial networks aim to minimize the distance between the distribution of the real data and the distribution of the data generated from the generator network (section 3.6). In order to verify those properties, we will investigate results when simulating data from a known probability distribution. The performance will be measured in the Jensen–Shannon divergence, the closeness of the estimated standard deviation and the coverage of the 80% and 95% prediction intervals. As this task is not the most computational demanding, we will run the experiments multiple times and provide the mean score along with the standard deviation.

5.1.1 Gaussian Distribution

The Gaussian samples are drawn according to $x \sim \mathcal{N}(0, 0.1)$, as described in section 4.1.2. We will confirm the GANs' ability to estimate the Gaussian distribution, and simultaneously investigate how the latent code dimension k affects the results. The remaining hyperparameters are shown in the appendix (table A1).

			Coverage	
k	D$_{\text{JS}}$	$\hat{\sigma}$	80%	95%
1	0.0313 ± 0.0159	0.1033 ± 0.0140	$77.94\% \pm 4.93\,pp$	$91.79\% \pm 4.49\,pp$
5	0.0140 ± 0.0088	0.0934 ± 0.0094	$75.14\% \pm 5.49\,pp$	$91.86\% \pm 3.09\,pp$
10	0.0070 ± 0.0044	$\mathbf{0.1010 \pm 0.0023}$	$80.02\% \pm 1.93\,pp$	$94.40\% \pm 0.92\,pp$
50	0.0056 ± 0.0019	0.0976 ± 0.0018	$78.19\% \pm 0.99\,pp$	$94.44\% \pm 0.51\,pp$
100	$\mathbf{0.0054 \pm 0.0018}$	0.0987 ± 0.0014	$78.56\% \pm 0.97\,pp$	$94.66\% \pm 0.41\,pp$

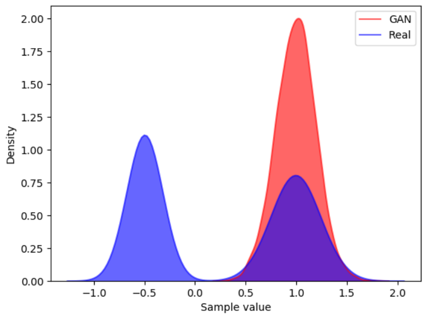
Table 5.1: GAN with different latent code dimensions estimating a Gaussian distribution with zero mean and 0.1 standard deviation, $x \sim N(0, 0.1)$. Results show the Jensen-Shannon divergence D$_{\text{JS}}$, uncertainty estimate of the distribution $\hat{\sigma}$ and coverage of the 80% and the 95% confidence interval with standard deviation averaged over 10 runs. pp denotes the percentage points. Best performing model for each metric is shown in bold.

Table 5.1 shows the results for different latent code dimensions $k \in \{1, 5, 10, 50, 100\}$. Judging by the estimated standard deviation and the prediction interval coverage, the GAN is able to estimate the Gaussian distribution pretty precisely for all values of k. The GAN with dimension $k = 100$ scores lowest in terms of the Jensen-Shannon divergence, and estimates the probability distributions closest. The GAN with latent code dimension $k = 10$ estimates the standard deviation slightly closer to the theoretical value than $k = 100$. With regard to the coverage, we observe generally good performance, where $k = 10$ has the highest coverage for the 80% prediction interval, and $k = 100$ the highest for the 95% prediction interval. Further, the stability of the models increase with larger dimension k, and the standard deviation for the metrics is at its lowest for dimension $k = 100$. Doing similar experiments with the batch size, we observe best performance for smaller batch sizes, namely 32 and 64.

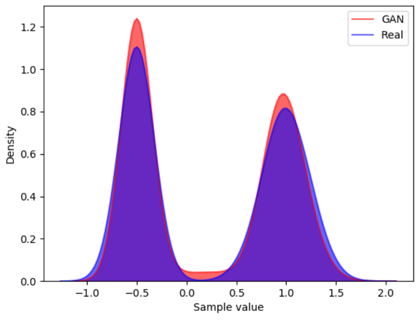
5.1.2 Bimodal Distribution

To further investigate the properties of estimating more complex distributions than the Gaussian distribution, we will sample from the bimodal distribution. We will now visually compare the performance of a standard GAN to a Wasserstein GAN (WGAN).

Figure 5.1 illustrates the estimated distributions with GAN and the WGAN using the same hyperparameters trained to generate a bimodal distribution. The GAN (5.1a) is only able to capture one of the modes, resulting in a mode collapse. On the other hand, the Wasserstein GAN (5.1b) shows convergence, where the generator learns both modes relatively close to the actual distribution. Notably, the batch size had to be large, 1024, in order to obtain convergence to a bimodal distribution, as showed by the hyperparameters in table A2.



(a) GAN



(b) WGAN

Figure 5.1: GAN and WGAN trained to generate a bimodal distribution $\mathcal{N}(-0.5, 0.1), \mathcal{N}(1, 0.2)$

5.2 Sine Curve with Gaussian Noise

We now introduce a time-dependent synthetic data set, a sine curve with Gaussian noise. We can still verify the distribution estimation properties through the known standard deviation σ_N of the noise. As the data is time-dependent, we will utilize the ForGAN model as described in section 3.6.2, which can model the forecast distribution conditioned on past observations. The 80% and 95% prediction intervals are used to investigate the distribution estimation quantities through coverage, width and mean scaled interval score (MSIS) (section 3.8.5). In addition we will investigate the point forecast accuracy through mean squared error (MSE) (section 3.8.1), symmetric mean absolute scaled error (sMAPE) (section 3.8.2) and mean absolute scaled error (MASE) (section 3.8.3). As we know that both the baseline models and the neural networks are able to model the dynamics of this synthetic time series, we expect the simpler models to perform better.

Hyperparameters	**Value**
Type of cells in the Generators first layer	SimpleRNN
Type of cells in the Discriminators first layer	SimpleRNN
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	1500 / 3000
Batch size	32
Learning rate	0.001
Discriminator iterations per generator iteration	3
Condition window length	24

Table 5.2: Hyperparameters used for ForGAN unless otherwise specified.

Table 5.2 shows the hyperparameters used unless otherwise specified, which are deduced from the results of the distribution estimation in the last section. This may not indicate that they are optimal in the forecast setting, and we will further investigate how some of the hyperparameters affect the performance of the model.

Latent Code Dimension

We will investigate how the dimension of the latent code, as earlier denoted by k, affects the results of the forecasting. The results reported are averaged over a forecast horizon of $h = 1, ..., 24$, as this will disclose hyperparameters that may lead to an unstable performance in the recursive forecast setting. The number of training iterations is 1500 for this experiment, and the results are computed on a hold-out validation set.

			MSIS		Coverage		Width	
Model	k	MSE	80%	95%	80%	95%	80%	95%
ForGAN	1	0.0163	4.443	5.805	77.44%	96.49%	0.331	0.563
ForGAN	5	0.0132	3.755	4.906	81.00%	94.21%	0.298	0.458
ForGAN	10	0.0305	6.081	9.168	66.57%	84.81%	0.345	0.536
ForGAN	50	0.0166	4.302	6.065	65.84%	86.98%	0.246	0.384
ForGAN	100	**0.0104**	**3.266**	**4.262**	82.55%	97.18%	0.285	0.437

Table 5.3: The table shows the mean squared error and 80% and 95% prediction interval metrics on the validation set averaged over the forecast horizon of 24 steps. The hyperparameters used are shown in table 5.2, where the number of training iterations is 1500. The only hyperparameter changed between the models is the latent code dimension k.

Table 5.3 shows the validation results of the MSE and uncertainty estimates, namely the 80% and 95% prediction intervals, averaged over the forecast horizon. We observe that the dimension $k = 100$ scores best in terms of MSE and MSIS. Moving further to the coverage, we can see that in terms of closeness to the coverage probability, the latent code dimension of $k = 5$ is closest with 81.00% and 94.21% coverage for the 80% and 95% prediction intervals respectively. While not being closest to the coverage probability, the model with a dimension of $k = 100$ has sufficient coverage for both prediction intervals. In addition, it has the narrowest prediction intervals among those with sufficient, or close to sufficient, coverage for both prediction intervals. Couple that with the lowest MSIS and MSE, preferring the latent code dimension $k = 100$ is well grounded in the results.

Discriminator Iterations

Another quantity to investigate is the number of training iterations for the discriminator $\mathcal{D}$ per training iteration for the generator $\mathcal{G}$, D_{iter}. As discussed in section 3.6.3, insufficient training of $\mathcal{D}$ compared to $\mathcal{G}$ may lead to mode collapse. On the other hand, a too large D_{iter} will increase training time substantially, without increased training of $\mathcal{G}$. Therefore, we will investigate how D_{iter} affects the performance on a validation set, as before averaged over a forecast horizon of $h = 24$. The experiments are run according to the hyperparameters in table 5.2, with 3000 training iterations.

			MSIS		Coverage		Width	
Model	D_{iter}	MSE	80%	95%	80%	95%	80%	95%
ForGAN	1	0.0214	5.503	8.647	50.86%	73.81%	0.224	0.344
ForGAN	3	0.0141	3.821	5.428	86.66%	97.38%	0.342	0.524
ForGAN	5	0.0135	3.794	5.108	79.44%	94.66%	0.295	0.453
ForGAN	10	**0.0113**	**3.467**	**4.651**	84.16%	97.51%	0.313	0.480
ForGAN	20	0.0137	3.739	4.866	81.40%	96.35%	0.311	0.478

Table 5.4: The table shows the validation results for the mean squared error and 80% and 95% prediction interval metrics on the validation set averaged over the forecast horizon of 24 steps. The hyperparameters used are shown in table 5.2, where the number of training iterations is 3000. The only hyperparameter changed between the models is the D_{iter}.

Table 5.4 shows the validation results of the MSE, and the 80% and 95% prediction intervals. The best performing model in regard to the MSE and MSIS is $D_{iter} = 10$. In addition, we can see that the prediction intervals have sufficient coverage. One can argue that both $D_{iter} = 5$ and $D_{iter} = 20$ is performing better, as they have coverage closer to the coverage probability and narrower prediction intervals. As discussed in section 3.8.4, too high coverage is not an unwanted feature if the intervals are not too wide. The best performing model when tuning the latent code dimension in the last subsection performs better than any model in table 5.4. This can indicate that the effect of D_{iter} is correlated with the total number for training iterations. We will thus choose the model trained for 1500 iterations and with $D_{iter} = 3$ as it performed the best.

Wasserstein GAN

For the bimodal distribution, we observed that the Wasserstein GAN (WGAN) was able to model the complex distribution without the severe mode collapse observed for the GAN. This corresponds well with the theory described in section 3.6.4, where the properties of the Wasserstein loss should lead to a more stable convergence where mode collapse is avoided. As the time series setting might differ, we will investigate the performance of the WGAN and compare the results to the standard ForGAN. The ForWGAN is train as a ForGAN with Wasserstein loss, and with the set of hyperparameters yielding best performance (table A3).

		MSIS		Coverage		Width	
Model	MSE	80%	95%	80%	95%	80%	95%
ForGAN	0.0104	**3.266**	**4.262**	82.55%	97.18%	0.285	0.437
ForWGAN	0.0104	4.523	6.823	98.32%	100.00%	0.487	0.746

Table 5.5: The table shows the validation results for the mean squared error and 80% and 95% prediction interval metrics on the validation set averaged over the forecast horizon of 24 steps. The ForWGAN are trained according to hyperparameters showed in table A3.

Table 5.5 shows the validation MSE and the performance of the 80% and 95% prediction intervals averaged over the forecast horizon of 24 steps. While the models perform equally in terms of MSE, we observe that the ForGAN has lowest MSIS for both prediction intervals, as well as coverage closest to the coverage probability. Further the ForGAN has the narrowest prediction intervals, making the binary cross-entropy loss an obvious choice.

5.2.1 Choosing Models for Comparison

The ARIMA model is fitted using `auto_arima` with seasonality $m = 12$, which will find the model with lowest AICc (3.21). This results in a seasonal ARIMA$(0, 0, 1)$x$(1, 0, 1)_{12}$ model. For forecast horizons $h \geq m$, we have that $\hat{y}_{y+h} = \hat{y}_{y+h-m}$, since $\hat{z}_{t+h-m} = 0$. Doing model selection of exponential smoothing using the lowest AICc leads to a additive error and seasonality, ETS(A,N,A). The ForGAN model utilizes the best set of hyperparameters found previously in this section and are shown in table 5.2, where the number of training iterations is 1500. The hyperparameters for the MC dropout model is found similarly to the ForGAN's, by comparing results on the validation set. The hyperparameters yielding the best performance can be found in the appendix (table A4).

5.2.2 Results

We will now compare the performance of the proposed ForGAN model with the baseline models and state-of-the-art model on the test set. Similarly to the previous results, the metrics are averaged over the forecast horizon of 24. In addition, figures will show the results as a function of the forecasting horizon, increasing insight into the performance as the forecast horizon increases.

	Point forecast		
Model	MSE	sMAPE	MASE
ARIMA	0.0099	37.91%	0.723
ETS	**0.0097**	40.32%	**0.720**
MC dropout	0.0115	36.83%	0.789
ForGAN	0.0112	**36.33%**	0.780

Table 5.6: The table shows the forecast error metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 24 steps.

Table 5.6 shows the performance of the models in terms of point forecast. The best performing model is the exponential smoothing, with an MSE of 0.0097. This is less than the argued minimum MSE (section 4.1.3), which is due to sample variance of the Gaussian noise is $\sigma_N^2 = 0.0096$. The ARIMA is performing close to the exponential smoothing, followed by the ForGAN. We will not rely too much on the sMAPE, as it may be problematic due to the time series includes values close to, and equal to 0. We observe the similar results for the MASE as the MSE. The baseline models are close to the theoretically best obtainable value for the MASE given by ≈ 0.7071, whereas the MC dropout and ForGAN performs a little worse. They are however much closer to the optimal value than to 1, which makes them considerable better than the in-sample naive last season forecast error. It is expected that less complex models perform best on this data set that has such simple dynamics, and as ARIMA and exponential smoothing are sufficient to model the data, the results correspond with Occam's razor[1].

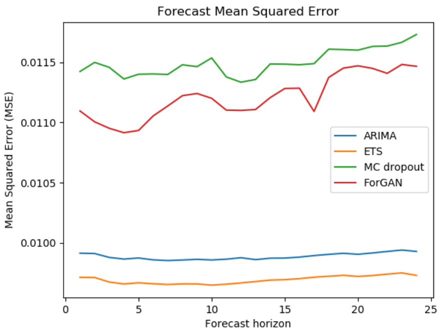
		MSIS		Coverage		Width	
Model	$\hat{\sigma}$	80%	95%	80%	95%	80%	95%
ARIMA	0.099	3.206	4.237	78.95%	94.80%	0.254	0.389
ETS	**0.098**	**3.163**	**4.088**	79.39%	94.92%	0.250	0.382
MC dropout	0.161	3.682	5.223	93.59%	99.24%	0.412	0.631
ForGAN	0.112	3.402	4.438	80.91%	95.77%	0.284	0.437

Table 5.7: The table shows the forecast 80% and 95% prediction interval metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 24 steps.

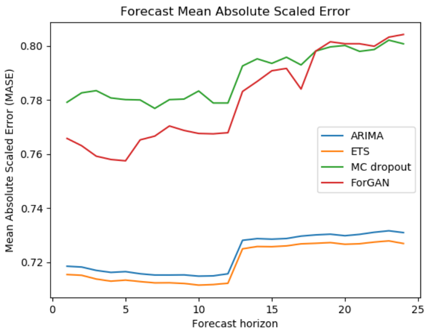
Table 5.7 shows the models compared on the uncertainty measures, with the addition of the estimated standard deviation. The standard deviation is a known metric for this data set, and is showed to be $\sigma = 0.1$ in section 4.1.3. We observed that the sample variance was slightly lower than the theoretical variance, which leads to the sampled standard deviation being $\sigma_N = 0.098$. The baseline models estimate this quantity pretty precise, whereas the MC dropout and the ForGAN overestimate the standard deviation. This is not necessarily undesirable behavior, but rather reflects the lack of forecast accuracy as showed in table 5.6. Further, the exponential smoothing has the best MSIS for both the 80% and the 95% prediction intervals. The ForGAN shows a better estimate of the standard deviation and lower MSIS than the state-of-the-art MC dropout model. The coverage showed by both the exponential smoothing and ARIMA, as well as the ForGAN approaches the coverage probability for both the 80% and the 95% prediction intervals relatively closely. The ForGAN has a coverage larger than the coverage probability, which is a desired feature if the prediction intervals are correspondingly narrow. The prediction intervals are wider than that of the baseline models, while compared to the MC dropout the ForGAN performs well. The MC dropout overestimates the forecast uncertainty largely and has too high coverage, which combined with too wide prediction intervals is undesirable.

Figure 5.2 shows the MSE (5.2a) and MASE (5.2b) as a function of the forecast horizon. The baseline's MSE stay rather constant, whereas the MC Dropout and the ForGAN has a slight increase in MSE over the forecast horizon. The neural networks have more variance in the MSE over the forecast horizon, as showed by the fluctuating plots. In figure 5.2b we observe a step-like behavior around $h = 12$, $h = 13$, which can be explained by the naive forecast scale going from one-season-ago prediction $\widehat{\text{Naive}}_{t+h}$, $h \leq m$ to two-seasons-ago prediction $\widehat{\text{Naive}}_{t+h}$, $h > m$. Knowing that the expected mean absolute difference (section 4.1.3) between observations one season apart is the same as two seasons apart, the step-like attribute can in this case be explained by randomness in the sampling of Gaussian noise. Further, we observe that the ForGAN's MASE seems to increase more over the forecast

[1]The idea of the simplest solution to a task is to be preferred until evidence proves otherwise.



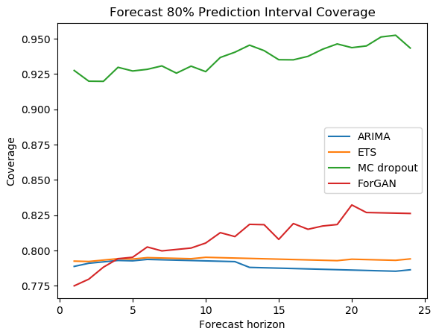
(a) Mean squared error (MSE)



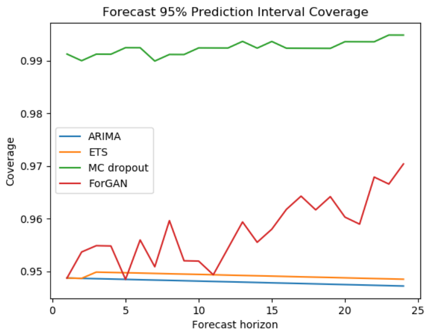
(b) Mean absolute scaled error (MASE)

Figure 5.2: The figure shows the MSE and MASE on a test set as a function of the forecast horizon.

horizon than the MSE. As discussed in section 3.8.1, an absolute error will penalize the absolute error equally, whereas the squared error will favor more consistent models. Thus we can deduct that while the absolute error of the ForGAN is equal to the MC dropout for the 18-24 steps ahead forecasts, the forecast accuracy of the ForGAN is more consistent.



(a) 80% prediction interval coverage



(b) 95% prediction interval coverage

Figure 5.3: The figure shows the coverage of 80% and 95% prediction intervals as a function of the forecast horizon, computed on the test set of the sine data set.

Figure 5.3 shows the coverage of the prediction intervals as a function of the forecast horizon. Here we can clearly see that the prediction intervals produced by the baseline models cover a consistent percentage of observations, whereas the neural networks, ForGAN especially, increase the coverage throughout the forecast horizon. This can be a consequence of the recursive forecast technique (section 3.7) used to obtain a multi-step forecast, which the model is not trained to optimize. Notably, we tried different numbers of Monte Carlo samples from the generator, both $B = 200$ and $B = 5000$ without observing significant improvement in performance as the number of samples increased. Thus, we can conclude that the accuracy of the estimation is not to fault for the drift of the coverage. These figures show that the average coverage value presented in table 5.7 may be misleading, and that changing the forecast horizon would give a different result. However, it is better that the coverage is increasing rather than decreasing over the forecast horizon.

After inspecting the results of this experiment, the baseline models perform as expected close to the optimal performance, whereas the neural networks are less accurate. Although we see an improved performance of the ForGAN compared to the MC dropout, where the coverage is close to the coverage probability and the prediction intervals are narrower. Moreover, the ForGAN has sufficient coverage for both prediction intervals, making it promising model for more complex time series. Notably, the Wasserstein GAN was not able to improve the performance over the standard GAN, scoring poor in terms of the MSIS due to wide prediction intervals.

5.3 Oslo Temperature Data Set

As time series forecasting usually applies to real data where the dynamics often are unknown and complex, we will compare the proposed ForGAN method to the baseline models on a time series created from observations of the temperature in Oslo. The data is a low-frequency time series with monthly observations, reminiscent of the sine curve with Gaussian noise.

5.3.1 Choosing Models for Comparison

Fitting the `auto_arima` model, we obtain a SARIMA(0, 0, 2)x(2, 0, 1)$_{12}$. This model is similar to the SARIMA model obtained for the sine data, with two additional parameters in MA(2) and seasonal AR(2). The best performing exponential smoothing (ETS) model with respect to the AICc is as previouisly with no trend and additive error and seasonality ETS(A,N,A). This corresponds well with observations of the time series, as there is no visual increase or decrease in temperature over time.

Hyperparameters	Value
Type of cells in the Generators first layer	SimpleRNN
Type of cells in the Discriminators first layer	SimpleRNN
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	5000
Batch size	64
Learning rate	0.001
Discriminator iterations per generator iteration	10
Condition window length	24

Table 5.8: Hyperparameters used for the ForGAN for the Oslo temperature data set.

Table 5.8 shows the hyperparameters used for the ForGAN trained on the Oslo temperature data set. They are found similarly to the experiments showed in the previous section, where the hyperparameters in the table are tuned. The model chosen is the best performing with regard to both mean squared error and prediction interval performance on the hold-out validation set. Notably, we observe better performance with increased training iterations than previously, even though it is less training samples. The hyperparameters for the MC dropout method are found similarly, and table A5 shows the hyperparameters used.

5.3.2 Results

Table 5.9 shows the point forecast error metrics for the models averaged over a forecast horizon of 24. Regarding the mean squared error (MSE), exponential smoothing is the best performing model, followed by ARIMA. Additionally, we observe that the ForGAN performs closer to the baseline models than the MC dropout model. As discussed in section 4.1.4, the sMAPE can be quite misleading for this particular data set. We will therefore not rely too much on the sMAPE results for this data set, but can note that they seem to

	Point forecast		
Model	MSE	sMAPE	MASE
ARIMA	3.8936	**43.28%**	**0.698**
ETS	**3.8699**	44.66%	0.702
MC dropout	4.3461	48.65%	0.771
ForGAN	4.0994	47.53%	0.740

Table 5.9: The table shows the forecast error metrics for the different baselines and the state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 24 steps.

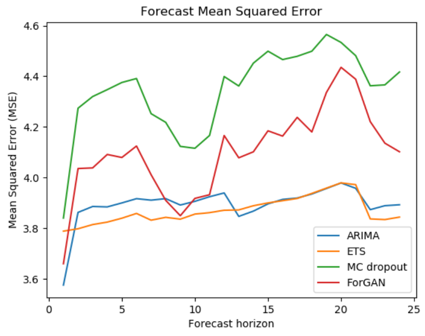
mostly concur with observed results of the MSE. Lastly, we observe slightly better values for the mean average scaled error (MASE) than for the previous data set. This strengthens the theory that the models capture a more complex dynamics.

	MSIS		Coverage		Width	
Model	80%	95%	80%	95%	80%	95%
ARIMA	3.520	4.886	82.97%	94.78%	5.385	8.236
ETS	3.512	4.840	83.18%	94.80%	5.458	8.358
MC dropout	4.057	7.069	84.53%	96.46%	6.062	9.282
ForGAN	**3.356**	**4.613**	79.52%	95.06%	4.935	7.831

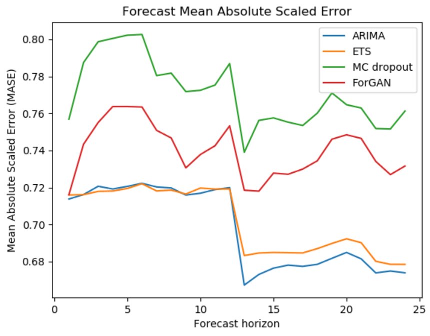
Table 5.10: The table shows the forecast 80% and 95% prediction interval metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 24 steps.

Further, we inspect the uncertainty measures in table 5.10. The mean scaled interval score (MSIS) shows that the ForGAN model is performing best for both prediction intervals, having the lowest score. The MC dropout has substantially higher MSIS than the other models, suggesting either poor coverage or wide intervals. Looking at the coverage we have relatively good performance by all models, and the models having too low coverage are very close to the coverage probability. Due to the ambiguity of prediction intervals, one might argue that it is insignificant in this case. Further, we observe that the ForGAN has the narrowest prediction intervals, while still having almost exact coverage. While not largely overestimating the uncertainty, the MC dropout has wide prediction intervals, explaining the poor MSIS. Judging by these results, the ForGAN is able to asses the narrowest prediction interval with coverage close to the coverage probability, and is arguably the best performing model in terms of forecast uncertainty estimates.

Figure 5.4 displays the MSE and MASE over the forecast horizon. Similarly to observations for the sine data, the baseline models show less variance over the forecast horizon. We also see that both the MC dropout and the ForGAN seem to have the same behavior over the horizon, reminiscent of cyclic behavior. Investigating the mean absolute scaled error (MASE) in figure 5.4b we observe a distinct drop after a season, similar to what observed for the sine time series. This indicates that the naive seasonal forecast is less accurate



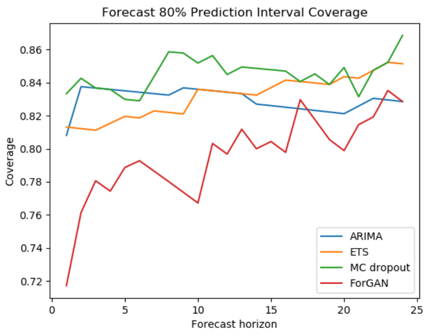
(a) Mean squared error (MSE)



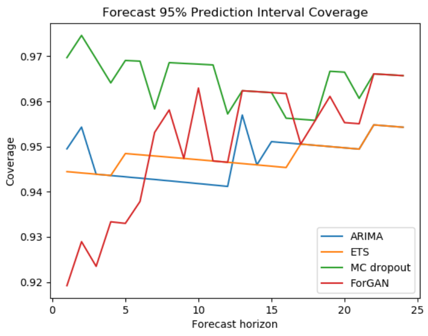
(b) Mean absolute scaled error (MASE)

Figure 5.4: The figure shows the MSE and MASE over the forecast horizon on the Oslo temperature data test set.

forecasting two seasons ahead. The models on the other hand are clearly able to model two seasons ahead more accurately, resulting in the decrease of MASE after one season. Omitting the level differences, the results appear quite similar to the MSE.



(a) 80% prediction interval coverage

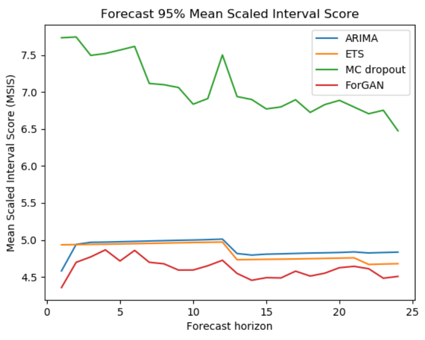


(b) 95% prediction interval coverage

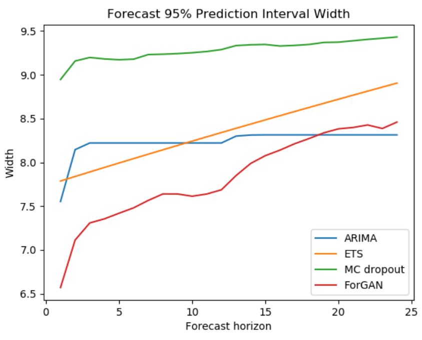
Figure 5.5: The figure shows the coverage of 80% and 95% prediction intervals over the forecast horizon on the Oslo temperature data test set.

Figure 5.5 shows tendencies conflicting with results obtained earlier in table 5.10. Here the ForGAN has poor coverage for the first 2-6 forecast horizons but increasing along with the forecast horizon. This makes the illusion of a well performing model when averaging as in table 5.10, whereas in reality, it has both poor performances for the first forecasting horizon and changing coverage over the forecast horizon. In contrast, the remaining models having much more consistent performance, which indeed is favorable.

To investigate the conflicting results obtained regarding the performance of the ForGAN



(a) Mean scaled interval score (MSIS)



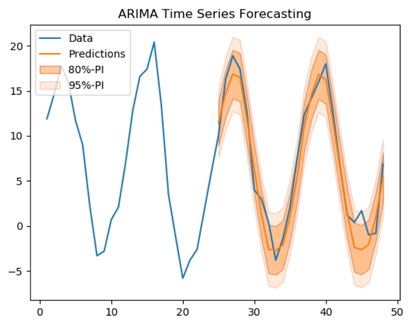
(b) Prediction interval width

Figure 5.6: The figure shows the MSIS and width of the 95% prediction interval on the test set as a function of the forecast horizon

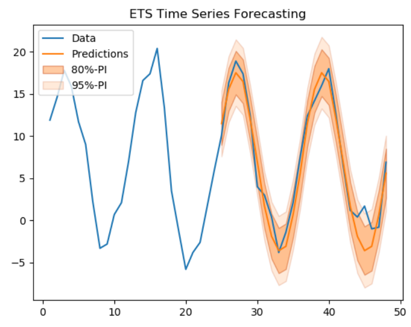
prediction intervals, figure 5.6 exhibits the mean scaled interval score (MSIS) and the width of the 95% prediction interval over the forecast horizon. We are assuming similar behavior for the 80% prediction interval MSIS and width. The ForGAN MSIS in figure 5.6a stays rather constant over the forecast horizon, whereas the coverage in figure 5.5 increases over the forecast horizon. As showed in section 3.8.5, the MSIS will heavily penalize observations outside the prediction interval, with a scaled distance from the prediction interval boundary to the observation, at the scale of 40 for the 95% prediction interval. Thus, as the MSIS does not decrease over the forecast horizon, we can assume the prediction interval boundary lies close to the observations. Figure 5.6b shows the 95% prediction interval width over the forecasting horizon. Except for the first two forecast horizons, the ForGAN prediction interval widens with the same factor as the exponential smoothing, which will not account for the constant MSIS while the coverage increases. This strengthens the theory that the observations lie close to the prediction interval boundary for the ForGAN, resulting in a low MSIS while not quite sufficient coverage.

Figure 5.7 shows examples of forecasts over the forecast horizon with the four models on the test set, along with the 80% and 95% prediction intervals. Notably, all the models perform relatively equal and capture the seasonal behavior. The ForGAN estimates the uncertainty at the lower temperatures to be slightly larger than at the upper points. We do not observe similar behavior in the other models, indicating that the ForGAN either has captured an uncertainty related to colder temperatures or it just not being able to capture the dynamics precisely. As we have seen that the ForGAN produces the narrowest and most accurate prediction intervals, it suggests the former.

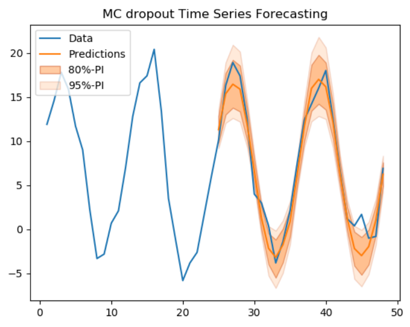
To summarise the results of the Oslo temperature data set, we have seen that while the ForGAN was not the most accurate in terms of point forecast, it scored best in terms of MSIS. It had sufficient coverage for the 95% prediction interval, and close to sufficient coverage for the 80% prediction intervals. Moreover, the prediction intervals were the narrowest, indicating precise uncertainty estimate. We observed some unsatisfactory behavior related



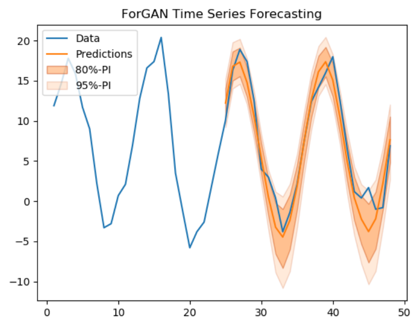
(a) ARIMA



(b) Exponential smoothing



(c) MC dropout



(d) ForGAN

Figure 5.7: Examples of the different models doing out-of-sample prediction with uncertainty estimates for a multi-step forecasting on the Oslo temperature data set. None of the test data where known during prediction.

to the coverage for the first forecast horizon. However, by inspecting the MSIS and the prediction interval width for the 95% prediction intervals, the low coverage for the first forecast horizon is not as alarming as it seems.

5.4 Avocado Price Data Set

As discussed in section 4.1.5, this data set contains multiple time series with the weekly avocado price across different regions in the US. The time series are shorter than the previously observed time series. On the other hand, neural networks can be trained across all time series, so we can gain learning between the time series. Visual inspection suggests that the time series are more complex than the previous time series encountered, and that the exogenous variables may be necessary to model the time series to its full extent. We will only use previous values of the average price, making it a univariate time series, where information about the dynamics may be lost. Nevertheless, a well performing prediction interval estimate should have sufficient coverage regardless of the model's ability to model the time series dynamics.

5.4.1 Choosing Models for Comparison

As earlier, we will use `pmdarima's auto_arima` to fit the best SARIMA/ARIMA model. An individual model is fitted for each of the time series. As there will be 108 different ARIMA models, we will not list all configurations here, although in most cases the best model does not include seasonal components. We choose the same strategy of fitting an exponential smoothing to each individual time series. When looking at the models obtaining the lowest AICc, for most regions it takes the form of an ETS(A,N,N), with the linear, trendless forecast function (3.13). However, for a small amount of the time series it takes the form of an ETS(A,A,N), which models an additive trend.

The MC dropout model is trained across all the time series simultaneously. The model performed best when inputting the previous $\ell = 26$ observations, which corresponds to half a year. Table A6 shows the hyperparameters used, and we observe a lower learning rate and some additional training iterations than for the previous data set.

Hyperparameters	**Value**
Type of cells in the Generators first layer	SimpleRNN
Type of cells in the Discriminators first layer	SimpleRNN
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	30000
Batch size	32
Learning rate	0.0001
Discriminator iterations per generator iteration	3
Condition window length	52

Table 5.11: Hyperparameters used for the ForGAN for the avocado price data set.

The ForGAN is also trained across all the time series simultaneously, and table 5.11 shows the hyperparameters used. In contrast to the MC dropout model, the ForGAN performed best with a window size of $\ell = 52$, which corresponds to a year. This will reduce the number of samples compared to the MC dropout, however, there is a higher possibility of picking up seasonal effects. In addition, the model performs better with a lower learning rate and an additional number of training iterations, which can be explained by the training data size being larger than previously.

5.4.2 Results

In this section we will analyze the results of the avocado price data set. As there is only 33 test samples per time series in this data set, the estimated coverage per time series will be rather imprecise. A 95% prediction interval covering 31 of the samples will yield a coverage of 93.93%, whereas covering 32 test samples will yield 96.93% coverage. Small variations can drastically affect the results, and we will consequently present the results averaged over the 108 time series. To further gain insight in the variability of the performance, a standard deviation of each metric is provided along with the mean value across the time series.

	Point forecast		
Model	MSE	sMAPE	MASE
ARIMA	0.259 (0.170)	17.24% (6.71 *pp*)	0.870 (0.344)
ETS	0.293 (0.210)	19.97% (7.68 *pp*)	1.036 (0.504)
MC dropout	**0.171** (0.123)	**14.46%** (6.13 *pp*)	**0.706** (0.300)
ForGAN	0.185 (0.121)	15.14% (4.31 *pp*)	0.745 (0.272)

Table 5.12: The table shows the forecast error metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 13 steps. The mean value over the time series is shown, with the standard deviation in parentheses below. *pp* denotes the percentage point.

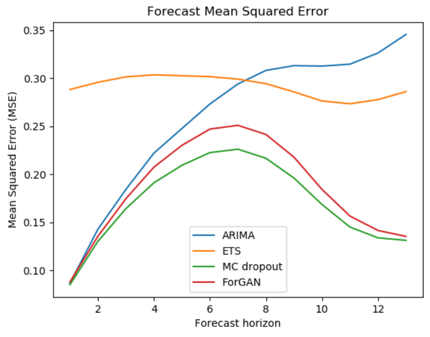
Table 5.12 shows the model performances in point forecast accuracy averaged over the time series from each region. The MC dropout performs best on all metrics, followed closely by the ForGAN. Moreover, the order of performance is identical for all metrics, indicating that the performance is so different that the nature of the metrics does not influence the order. In addition, the neural networks have the lowest standard deviation of the metrics, suggesting generally better performance across the time series. Contrary to previous results, exponential smoothing is the worst performing model. This may not come as a shock, as the time series appears to be non-stationary. ARIMA however should be able to deal with such data better, and while it does, the neural networks are performing better.

When examining the uncertainty results in table 5.13, we can see that the ForGAN scores lowest in terms of MSIS for both the 80% and 95% prediction interval. We further observe that the MC dropout does not maintain the performance seen in the point forecast accuracy, scoring worst in terms of 95% prediction interval MSIS. The ForGAN is the only model

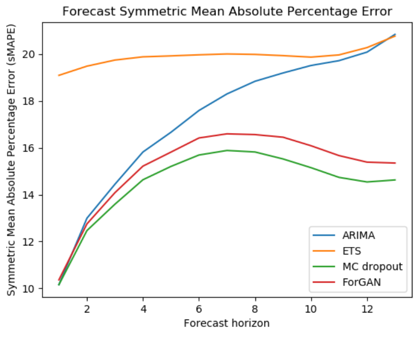
	MSIS		Coverage		Width	
Model	80%	95%	80%	95%	80%	95%
ARIMA	4.371 (1.854)	7.086 (4.164)	69.41% (17.44 *pp*)	85.59% (11.04 *pp*)	0.955 (0.176)	1.461 (0.270)
ETS	4.893 (2.470)	7.574 (4.829)	65.57% (21.95 *pp*)	84.20% (14.02 *pp*)	1.101 (0.213)	1.687 (0.326)
MC dropout	4.573 (2.362)	11.727 (7.868)	77.36% (14.61 *pp*)	90.31% (9.31 *pp*)	0.864 (0.048)	1.324 (0.074)
ForGAN	**3.600** (1.281)	**5.384** (2.485)	82.61% (10.80 *pp*)	93.81% (5.50 *pp*)	1.035 (0.071)	1.587 (0.102)

Table 5.13: The table shows the forecast 80% and 95% prediction interval metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 13 steps. The mean value over the time series is shown, with the standard deviation in parenthesise below. *pp* denotes the percentage point.

having coverage close to the coverage probability, followed by the MC dropout. This may suggest that the prediction intervals of MC dropout are quite wide despite the coverage being better than the baseline models. However, the prediction interval width reveal that this is not the case. As discussed earlier, the MSIS will penalize values outside the prediction interval by a scaled distance from the prediction interval boundary, suggesting that values not covered are far from the prediction intervals. The remaining models have similar prediction interval widths, which suggests that the ForGAN does estimate the uncertainty most accurate due to both the best MSIS and the highest prediction interval coverage.



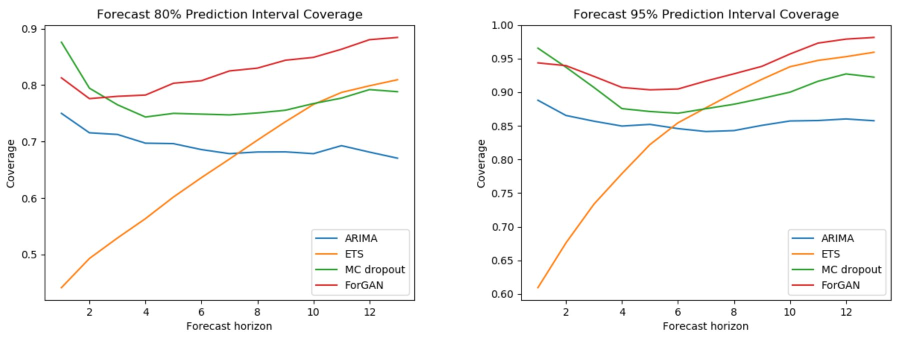
(a) Mean squared error (MSE)



(b) Symmetric mean absolute percentage error (sMAPE)

Figure 5.8: The figure shows the MSE and sMAPE over the forecast horizon of 1 to 13 on the avocado price data set.

The exponential smoothing stands out for the point forecast accuracy (figure 5.8) as having large forecast error for the one-step-ahead forecast. While it stays rather constant over the forecast horizon, we see the three other models performing substantially better for shorter forecast horizons. However, as the forecast horizon increases further, the forecast error of the neural networks decrease. The figures in section 4.1.5 showed for some of the time series large variations within the first few observations of the test data. The tendency was a substantial decrease in the avocado price within the first observations. The larger forecast horizons are not forecasted on the first observations in the test set, explaining why we might observe this behavior. It could be avoided if we have had more test data, spanning over a more extended time period. The hypothesis of decreasing error due to decrease in price over the test set is strengthened by figure 5.8b, where the tendencies are not as strong. A lower average price will penalize the absolute percentage error more heavily, as the sMAPE is scaled by the price level (section 3.8.2).

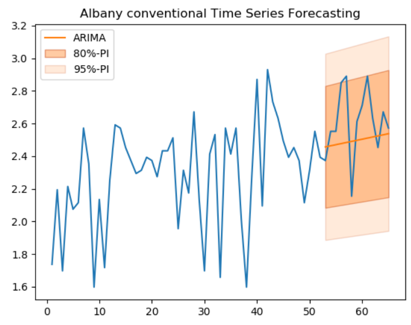


(a) 80% prediction interval coverage **(b)** 95% prediction interval coverage

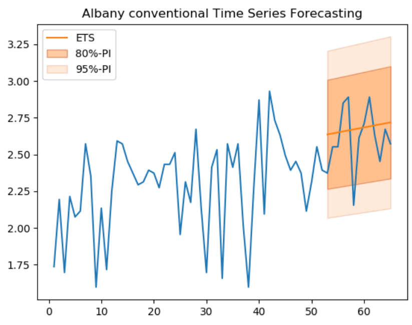
Figure 5.9: The figure shows the coverage of 80% and 95% prediction intervals over a forecast horizon of 1 to 13 on the avocado price data set.

Figure 5.9 displays the 80% and the 95% prediction interval coverage over the forecast horizon. Again, the exponential stands out in terms of different behavior than the other models and low coverage for the one-step-ahead prediction interval. Recall that the exponential smoothing was not able to model any significant dynamics of the time series either, with the forecast uncertainty calculated according to equation 3.16 in the cases of ETS(A,N,N) or in some cases ETS(A,A,N) with forecast uncertainty according to equation 3.17. This is not a sophisticated forecast uncertainty, hence it is not unexpected that the model is not able to produce more accurate prediction intervals. The other models appear to capture the uncertainty in a similar fashion, however, we have seen earlier that the ForGAN expands the prediction intervals more as the forecast horizon increases than the other models. This can further explain why it increases its coverage slightly over the forecast horizon. While neither increase nor decrease of the coverage over the forecast horizon are optimal, the figure does not weaken the interpretation that the ForGAN is able to asses the forecast uncertainty more accurately than the other models.

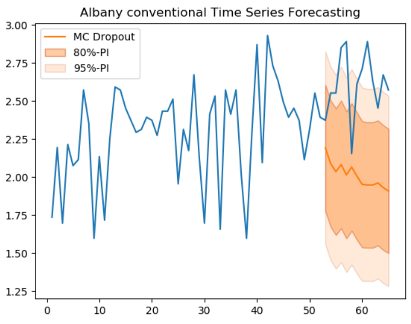
An example of the models forecasting the price of the conventional avocado in Albany 1 to 13 weeks ahead, with 80% and 95% prediction intervals, is shown in figure 5.10. The



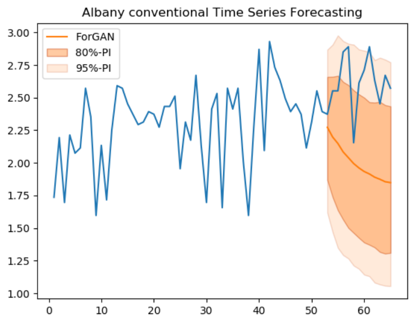
(a) ARIMA



(b) Exponential smoothing



(c) MC dropout

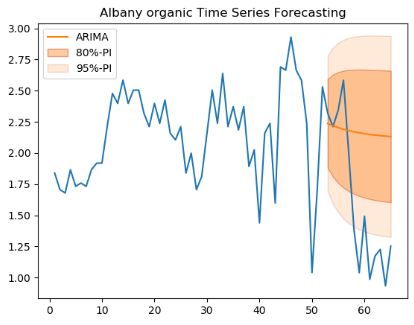


(d) ForGAN

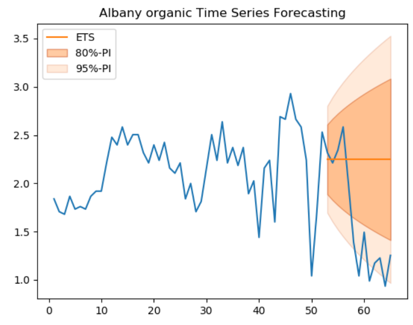
Figure 5.10: Examples of the models doing out-of-sample prediction with 80% and 95% prediction intervals, forecasting the price of the conventional avocado in Albany 1 to 13 weeks ahead. None of the test data where known during prediction.

ARIMA and exponential smoothing seem to produce more sensible forecasts, however, the prediction intervals are not widening as the forecast horizon increases. For the ARIMA model, this suggests that ϕ_j for $j > 0$ is small, as according to h-step-ahead forecast uncertainty (3.32). The same behavior in the exponential smoothing can be explained by the trend and seasonality smoothing parameters, α and β being small, as seen in equation 3.16 and 3.17. The neural networks on the other hand predict a decrease in the avocado price, however, it is not an accurate prediction in this case. The ForGAN predictions seem to be smoother than the MC dropout's, and it also has the prediction intervals widening over the forecast horizon. While this in it self is not a desired feature, generally speaking the h-step-ahead forecast is expected to be more inaccurate than the one-step-ahead forecast.

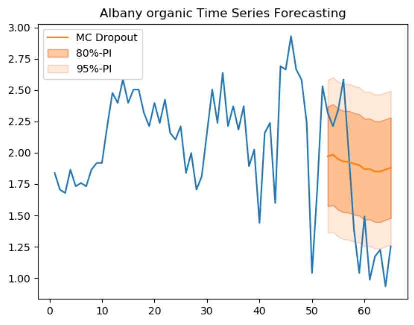
Figure 5.11 shows the same forecasts for the price of the organic avocado in Albany. Now, the ForGAN seems to make a more sensible forecast than the other models, as it is able to predict the decrease in avocado price. The other models expect the prices to stay on the level of the previous observations. Notably, the ForGAN forecasts seem to be suspiciously similar to the forecasts for the conventional avocado, thus one might be inclined to think that it makes a more general prediction, rather than model each time series separately.



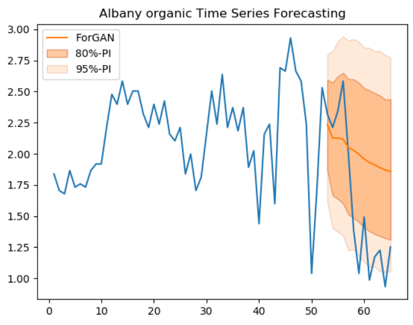
(a) ARIMA



(b) Exponential smoothing



(c) MC dropout



(d) ForGAN

Figure 5.11: Examples of the models doing out-of-sample prediction with 80% and 95% prediction intervals, forecasting the price of the organic avocado in Albany 1 to 13 weeks ahead. None of the test data where known during prediction.

Supported by the results where the ForGAN and MC dropout were able to produce both more accurate point forecasts and prediction intervals, the decreasing trend might be a concurring tendency among the majority of the time series, in which the neural networks were able to forecast. While the MC dropout was able to produce more accurate point forecasts, the ForGAN had both the best MSIS and closest coverage to the coverage probability. Judging by the results presented for this data set, we observe a clear case where the ForGAN is assessing the best forecast uncertainty estimates.

5.5 Electricity Consumption Data Set

The electricity consumption data set is a high-frequency time series with hourly observations. As discussed in section 4.1.6, we observed both daily and weekly seasonality. We choose to only select one of the time series provided in the data set due to limited availability of time and computational power. We also encountered exceeding the available random-access memory (8 GB) when fitting the `auto_arima` on a larger proportion of the time series, leading to execution error.

5.5.1 Choosing Models for Comparison

The ARIMA is as previously fitted using `auto_arima` with a seasonality of $m = 24$, resulting in a seasonal ARIMA(1, 0, 2)x(2, 0, 2)$_{24}$. We have autoregressive and moving average components, as well as seasonal autoregressive and moving average components. It is worth noting that as the model does not have the seasonal component of Φ_7, Θ_7, it can not capture the secondary seasonality we discussed in section 4.1.6. The ETS(A,N,A) has the lowest AICc, and is thus the model that will be utilized. As we could observe a distinct seasonality, but no trend in figure 4.6, this result is sensible. The ForGAN is

Hyperparameters	**Value**
Type of cells in the Generators first layer	LSTM
Type of cells in the Discriminators first layer	LSTM
Number of nodes per layer in Generator	64
Number of nodes per layer in Discriminator	256
Latent code dimension	100
Training iterations	2000
Batch size	64
Learning rate	0.0001
Discriminator iterations per generator iteration	5
Condition window length	336

Table 5.14: Hyperparameters used for the ForGAN for the electricity data set.

trained according to hyperparameters showed in table 5.14. Notably, the model performed best when utilizing LSTM layers as the recurrent layers, with additional nodes; 64 for the generator and 256 for the discriminator. This gives an idea of the time series dynamics being more complex than observed for the previous data sets, as the LSTM is able to model more long term dependencies. The ForGAN also performed best with a window size of $\ell = 336$, which corresponds to two weeks. The hyperparameters of the MC dropout is shown in table A7, and notably, the model performed best with a window size of $\ell = 168$. In contrast to the ForGAN, the MC dropout performed best utilizing SimpleRNN as a recurrent layer.

5.5.2 Results

The point forecast accuracy metrics are shown in table 5.15. The MC dropout performs best on all metrics, with an MASE at 0.731, which suggests that it models a more sophisticated dynamic than the naive seasonal forecast. We also observe that the ARIMA performs fairly well, however, the MASE at 0.976 suggests it does not perform substantially better than a naive seasonal forecast. The ForGAN does not have an MASE below 1, suggesting that it has not been able to model the seasonality to its full extent. With an MSE double of what the MC dropout has, the ForGAN appears to be performing quite bad. Notably, the sMAPE for the ForGAN is close to that of the ARIMA, which motivates the idea that the ForGAN forecasts more accurate on the time series, apart from the peaks. This is due to the sMAPE will scale errors at the peaks largely.

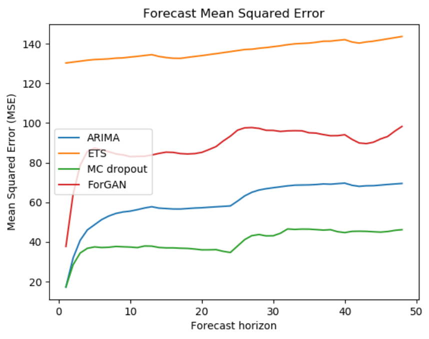
	Point forecast		
Model	MSE	sMAPE	MASE
ARIMA	60.08	26.20	0.976
ETS	136.65	32.65	1.448
MC dropout	**40.21**	**20.77**	**0.731**
ForGAN	88.58	27.56	1.103

Table 5.15: The table shows the forecast error metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 48 steps.

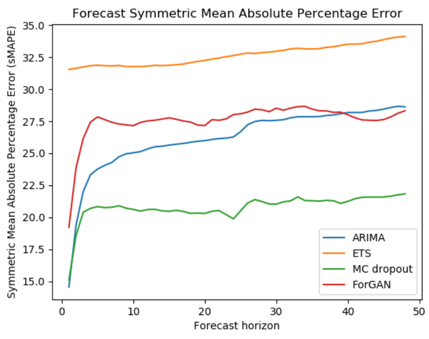
	MSIS		Coverage		Width	
Model	80%	95%	80%	95%	80%	95%
ARIMA	4.037	**5.161**	78.93%	96.60%	19.668	30.080
ETS	8.169	12.336	91.14%	96.80%	49.200	75.337
MC dropout	**3.768**	6.589	87.03%	97.18%	18.952	29.020
ForGAN	5.172	6.958	75.55%	89.91%	21.312	32.606

Table 5.16: The table shows the forecast 80% and 95% prediction interval metrics for the different baselines and state-of-the-art model, as well as the proposed ForGAN, averaged over a forecast horizon of 48 steps.

The prediction interval performance is shown in table 5.16. With regard to the MSIS, the MC dropout scores best for the 80% prediction interval, and the ARIMA for the 95% prediction interval. Looking at the coverage, we see as previous tendencies, the MC dropout overestimates the uncertainty. However, it also has the narrowest prediction intervals, making the prediction intervals more informative. Notably, the MC dropout has both highest coverage and narrowest prediction intervals for the 95% prediction interval, while still score substantially worse than the ARIMA in terms of MSIS. Accordingly, the observations not covered by the 95% prediction interval must be far from the prediction interval boundaries. The ForGAN, while not being too far off the coverage probability, underestimates the uncertainty. Further, it has wider prediction intervals than both the MC dropout and the ARIMA, which both have higher coverage. Coupling that with the fact that the ForGAN was not able to point forecast as accurate as the ARIMA and MC dropout, one would suspect some sort of mode collapse where it overestimates its ability to model the dynamics of the time series.



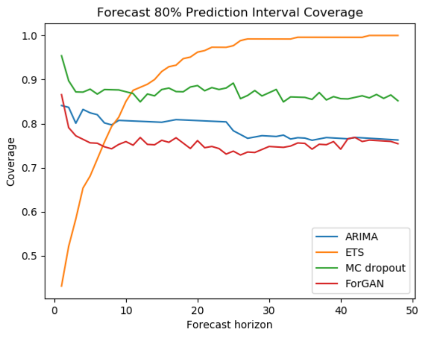
(a) Mean squared error (MSE)



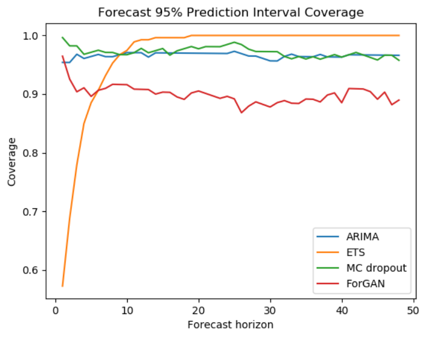
(b) Symmetric mean absolute percentage error (sMAPE)

Figure 5.12: Figures show the MSE and sMAPE as a function of the forecast horizon, computed on the test set of the electricity consumption data set.

Figure 5.12 shows the mean squared error and symmetric mean absolute scaled error over the forecast horizon. There is clearly a significant decline in forecast accuracy during the first forecast steps for all models except exponential smoothing. The MSE nearly doubles over the first two forecast horizons, whereas the sMAPE increases with about 5-7 percentage points for the three aforementioned models.



(a) 80% prediction interval coverage



(b) 95% prediction interval coverage

Figure 5.13: The coverage of 80% and 95% prediction intervals over the forecast horizon, computed on the test set of the electricity consumption data set.

Further we have the prediction intervals over the forecast horizon in figure 5.13. We observe that the ForGAN seems to have sufficient coverage for the one-step-ahead forecast, however the coverage decreases after the first forecast horizon and from there stays rather persistent. This behavior stands out from earlier observations, where the coverage has increased over the forecast horizon. Similar tendencies can be observed for the MC dropout, although at a higher coverage. This behavior can be attributed to the forecast error increases drastically during the first forecast horizon. Thus judging by the forecast horizon behavior, one can suspect that there is an instability where errors accumulate through the recursive forecast technique.

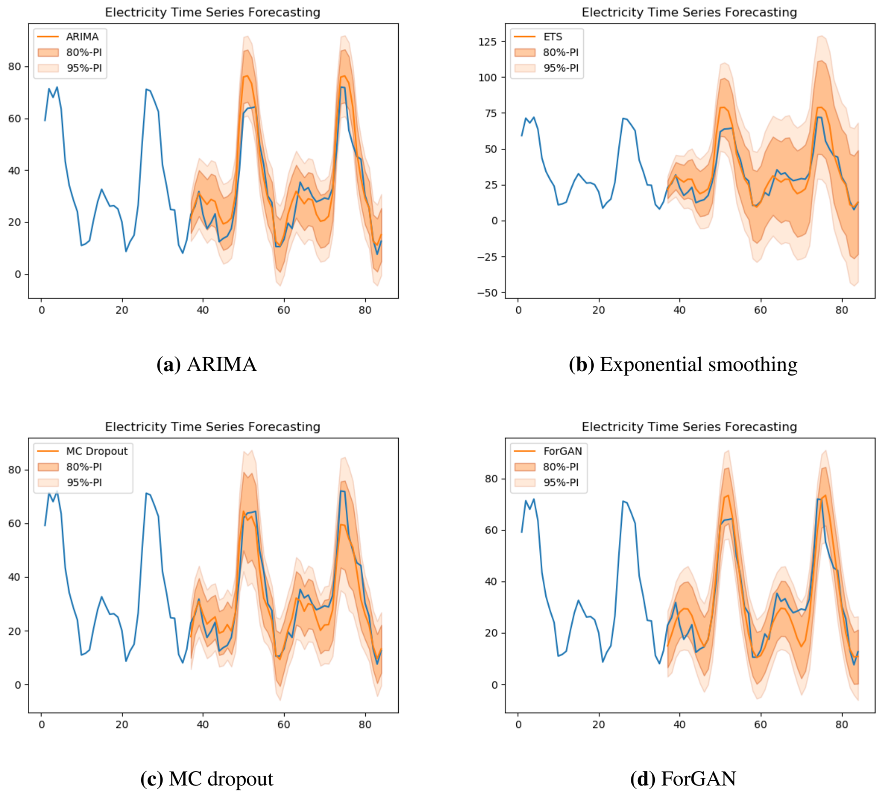


Figure 5.14: Examples of the models doing out-of-sample forecasts with 80% and 95% prediction intervals, forecasting the electricity consumption up to 48 hours ahead.

Figure 5.14 shows 1 to 48-hour forecasts on a test set, along with prediction intervals. While the MC dropout seems to somewhat underestimate the peaks of the power consumption, the other models rather overestimate it. In addition, the MC dropout has wide prediction intervals at the peaks of the electricity consumption, and narrower prediction intervals between the peaks. Conversely, the other models do not have such a noticeable change in the prediction interval width. We also observe that the ForGAN seems to forecast the smoothest function, similar to the previous data set, not modeling the small variations in the time series. However, it is hard to observe the dramatic differences in the forecast accuracy observed earlier. This can indicate that large fluctuations in the dynamics can affect the performance to a large extent.

Summarizing the results presented in this section, we observe a case where the ForGAN underestimates the uncertainty of the multi-step forecast. Conversely to the results for the previous data sets, some of the other models are able to assess the uncertainty much better, with higher coverage and narrower prediction intervals. This prompts for more research related to estimate forecast uncertainty with GAN, and further improve performance.

5.6 Discussion

In this section, we will summarize and discuss the findings in the previous sections of this chapter, as well as discuss the results against the hypotheses presented in section 4.3. We explored the properties of generative adversarial networks when estimating a Gaussian and a bimodal distribution, along with investigating how some hyperparameters influenced the convergence. We have also compared the proposed ForGAN method to three different time series models across a synthetic time series, as well as three real time series data sets with different frequencies. Both the point forecast accuracy and the forecasted prediction intervals have been compared, and we have investigated how the forecast horizon affects the aforementioned quantities.

First, we confirmed that the GAN was able to estimate a Gaussian distribution, and that the experimental setup was working correctly. The latent code dimension is a GAN specific hyperparameter, which makes for an interesting investigation of how it affects the performance of the GAN. The performance was explored for both the Gaussian distribution estimation and the synthetic sine time series with Gaussian noise. We observed for the Gaussian distribution that the performance for dimensions $k = 10$, $k = 150$, and $k = 100$ were fairly equivalent, whereas in the time series setting we observe a clear advantage of using a latent code dimension of $k = 100$.

Contradictory to the hypothesis, smaller batch sizes of 32 or 64 have yielded better performance across all the time series. When estimating the bimodal distribution with a Wasserstein GAN, the batch size had to be quite large to converge to a bimodal distribution. As we were only able to get the WGAN to converge to the bimodal distribution, these findings may be specific to the Wasserstein loss.

One of the ways to battle mode collapse is to ensure a strong discriminator $\mathcal{D}$, in the terms that it is able to distinguish real forecasts from the generated forecasts. In these experiments, we have tried updating $\mathcal{D}$ multiple times each training step, with the goal of a better trained discriminator. The optimal number of discriminator iterations have varied depending on the remaining hyperparameters, as well as the time series. However, we have seen the best results when D_{iter} has been between 3 and 10, updating the parameters considerable more than the generator.

Another measure to address mode collapse and more stable convergence is the Wasserstein loss. For the distribution estimation, we observed that the WGAN was able to generate samples from a bimodal distribution. However, in the time series forecasting setting we were not able to obtain as good results. Although research related to Wasserstein loss in the generative adversarial networks (Arjovsky et al. (2017), Gulrajani et al. (2017)) suggest that WGAN should indeed converge more stable, the findings in this thesis indicate otherwise. We should not disregard the fact that the WGAN will indeed perform better given a set of suitable hyperparameters, but for the range of hyperparameters tested in this thesis, Wasserstein loss did not yield any better results for the forecasting task. Both weight clipping and gradient penalty were applied, without success. Notably, Fu et al. (2019) were able to successfully use the Wasserstein loss with weight clipping in the forecast setting, although they did not utilize recurrent layers. There may be some incompatibility or adjustments that has to be done in order to work for recurrent layers.

After tuning the hyperparameters in the proposed ForGAN according to performance on a validation set, we compared the performance to two statistical baseline models and a state-of-the-art model for estimating forecast uncertainty using neural networks. Whereas we for the synthetic sine time series with Gaussian noise observed that the statistical models performed most accurately, we also observed quite well performance from the ForGAN. The coverage was sufficient, and it scored quite good with regard to the MSIS.

For the real time series data sets, the ForGAN was able to model low-frequency time series quite well, by scoring best in terms of MSIS for both the monthly temperature in Oslo and the weekly avocado price. It had sufficient, or close to sufficient, coverage for most prediction intervals. However, we also observed that the coverage was too low for the first forecast steps, and increasing over the forecast horizon for the Oslo temperature data set. Although this is undesirable behavior, the MSIS and prediction interval width was investigated, and revealed that it was not as problematic as we first though. For the avocado price data set, we observed a persistent coverage over the forecast horizon. In addition to scoring the best MSIS, the ForGAN had the highest and closest coverage to the coverage probability. While it did not forecast the dynamics of the time series to the full extent, none of the models were able to model it any significantly better, only the MC dropout was slightly more accurate. This may suggest that the univariate time series is not sufficient information to model the dynamics, and that external factors, such as the exogenous variables provided in the data set, is necessary in order to model the time series dynamics. Still, well-performing uncertainty estimates should have the correct coverage, independent of the model's ability to model the time series dynamics. The ForGAN was clearly able to estimate the uncertainty best for the avocado price data set, with close to sufficient coverage.

For the more high-frequency time series, the electricity consumption data set, the ForGAN had a slight decline in the coverage over the forecast horizon. The forecast accuracy nor the forecasted prediction intervals for the ForGAN performed as well on the electricity consumption data set compared to the ARIMA and the state-of-the-art MC dropout. A tendency in these observations is that the ForGAN was able to model the more low-frequency time series better, whereas the MC dropout was able to model the high-frequency time series more accurately.

Chapter 6

Conclusion and Further Work

6.1 Conclusion

The research aimed to investigate how generative adversarial networks can be used to estimate the forecast uncertainty. The approach is inspired by related work by Koochali et al. (2019) and Koochali et al. (2020), where the GAN successfully has been used to estimate the forecast distribution. Theory related to how generative adversarial networks are able to estimate distributions was presented in section 3.6, and further the GAN architecture for time series modeling was described; the ForGAN inspired by Koochali et al. (2019). Finally, we discussed some pitfalls of GANs related to mode collapse.

Further, this thesis has contributed with comparisons to a state-of-the-art model for estimating forecast uncertainty in neural networks, as well as well-known statistical models. We have confirmed that the GAN is able to estimate a simple distribution, using sampled noise without a time-dependency. We further explored the behavior when introducing a time dependency in the data, while still having a known noise distribution. We observed that the ForGAN was able to model the noise satisfying, however, the statistical baseline models were more accurate. This is expected behavior, as the statistical models have fewer parameters and are able to model the relationship of this time series exact. The more complex neural networks, with between 10^3 and 10^6 trainable parameters, will probably model some random noise as a time-dependent dynamic.

Finally, we compared the performance on three real time series data sets across both point accuracy metrics and metrics for assessing the quality of the 80% and the 95% prediction intervals. The metrics used for investigating the point forecast accuracy was mean squared error (section 3.8.1), symmetric mean absolute percentage error (section 3.8.2) and mean absolute scaled error (section 3.8.3). The quality of the prediction intervals were explored through the mean scaled interval score (section 3.8.5), prediction interval coverage (section 3.8.4) and prediction interval width. We observed that for some time series, the proposed

ForGAN produced the most accurate prediction intervals, whereas, for the high-frequency time series, the ARIMA and MC dropout were more accurate.

The neural networks are trained to forecast one step ahead, however, we wanted to investigate the uncertainty over the forecast horizon. Thus, the recursive multi-step forecasting method (section 3.7) has been utilized. We have seen the ForGAN both increasing and decreasing the coverage over the forecast horizon, which makes it hard to conclude that the recursive multi-step forecasting method has a bias when propagating the forecast uncertainty. The MC dropout also utilized the recursive multi-step forecast method, however, we did not necessarily observe the same drifts in the coverage across the forecast horizon. One might suspect that the ForGAN is sensitive to the input, and if the previous forecasted values are not exact, the coverage might drift.

6.2 Further Work

We find the proposed ForGAN to be promising, however avoiding mode collapse is essential to correctly estimate the uncertainty. We have seen behavior that was suspected to be related to mode collapse, where the ForGAN was either not able to model the time series to its full extent or the coverage was too low. As discussed in section 3.6.3, there are various ways to address this issue. In this thesis, we have investigated the Wasserstein loss (section 3.6.4) to address mode collapse, without observing any improved performance. Diversity-Sensitive (Yang et al. (2019)), Boundary Equilibrium GAN (Berthelot et al. (2017)) and mixed-batch training (Lucas et al. (2018)) are other ways to address this issue which can be adapted to the forecasting setting.

The avocado price data set contains exogenous variables, however, we omitted them to narrow the scope of the thesis. Using these variables in the multivariate forecasting setting, as seen by Koochali et al. (2020), is another way to further research the use of GAN for forecasting uncertainty. One will probably be able to model the time series dynamics, as well as the dynamics concerning the uncertainty better.

We have trained the ForGAN to forecast one step ahead, however, it is possible to train the GAN to forecast multiple steps ahead as discussed in section 3.7. Thus, one will optimize the GAN to forecast the h steps ahead, and avoid problems related to small error accumulating through the recursive multi-step forecast. The observed increase and decrease of the coverage over the forecast horizon are further issues that can be addressed and solved through a different multi-step forecasting technique.

In addition, we have only investigated architectures with one recurrent layer, but more complex architectures can be utilized. Additional layers and improved optimizers are examples of such improvements. More intricate framework architectures, such as Metropolis-Hastings GAN (Turner et al. (2018)), have shown improved results in the image generating task and can be adapted to further improve the performance of the generator in the forecasting setting.

Regularization techniques, such as dropout and weight regularizes can reduce the model's tendency to overfit. We have experienced that the generator in the ForGAN performed best when trained for fewer training steps than the MC dropout, suggesting that it is prone to overfitting. The MC dropout clearly uses dropout, which can reduce overfitting, and it is possible that the ForGAN could increase the performance when utilizing this technique. However, as the ForGAN is a stochastic model, introducing more randomness in the model during training will probably lead to an underestimation of the forecast distribution. We would therefore suggest that techniques such as dropout would be applied during forecasting as well, to avoid mode collapse.

Bibliography

Acerbi, C., Tasche, D., 2001. Expected shortfall: a natural coherent alternative to value at risk. `arXiv:cond-mat/0105191`. (last accessed on 2020-06-26).

Arjovsky, M., Chintala, S., Bottou, L., 2017. Wasserstein gan. `arXiv:1701.07875`. (last accessed on 2020-06-16).

Askanazi, R., Diebold, F.X., Schorfheide, F., Shin, M., 2018. On the Comparison of Interval Forecasts. PIER Working Paper Archive 18-013. Penn Institute for Economic Research, Department of Economics, University of Pennsylvania. URL: `https://ideas.repec.org/p/pen/papers/18-013.html`. (last accessed on 2020-06-18).

Berthelot, D., Schumm, T., Metz, L., 2017. Began: Boundary equilibrium generative adversarial networks. `arXiv:1703.10717`. (last accessed on 2020-06-02).

Brockwell, P., Davis, R., 2016. Introduction to Time Series and Forecasting. Springer Texts in Statistics, Springer International Publishing. URL: `https://books.google.no/books?id=P3fhDAAAQBAJ`. (last accessed on 2020-06-24).

Brophy, E., Wang, Z., Ward, T.E., 2019. Quick and easy time series generation with established image-based gans. CoRR abs/1902.05624. URL: `http://arxiv.org/abs/1902.05624`, `arXiv:1902.05624`. (last accessed on 2020-06-10).

Chollet, F., et al., 2015. Keras. `https://keras.io`.

Doraiswami, R., 1977. A decision theoretic approach to parameter estimation. Automatic Control, IEEE Transactions on 21, 860 – 866. doi:`10.1109/TAC.1976.1101385`. (last accessed on 2020-06-08).

Dua, D., Graff, C., 2017. UCI machine learning repository. URL: `http://archive.ics.uci.edu/ml`. (last accessed on 2020-05-10).

Esteban, C., Hyland, S.L., Rätsch, G., 2017. Real-valued (medical) time series generation with recurrent conditional gans. `arXiv:1706.02633`. (last accessed on 2020-06-10).

Fildes, R., Armstrong, J., 1979. Long-range forecasting: From crystal ball to computer. The Journal of the Operational Research Society 30, 673. doi:10.2307/3009390. (last accessed on 2020-06-02).

Frazier, P.I., 2018. A tutorial on bayesian optimization. arXiv:1807.02811. (last accessed on 2020-06-08).

Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M., Poggio, T., 2015. Learning with a wasserstein loss. arXiv:1506.05439. (last accessed on 2020-06-16).

Fu, R., Chen, J., Zeng, S., Zhuang, Y., Sudjianto, A., 2019. Time series simulation by conditional generative adversarial net. arXiv:1904.11419. (last accessed on 2020-06-10).

Gal, Y., Ghahramani, Z., 2015. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, in: ICML. (last accessed on 2020-05-30).

Glosser.ca, 2019. Artificial neural network. URL: https://commons.wikimedia.org/w/index.php?curid=24913461. (last accessed on 2020-01-10).

Gneiting, T., Raftery, A.E., 2007. Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association 102, 359–378. URL: https://doi.org/10.1198/016214506000001437, doi:10.1198/016214506000001437, arXiv:https://doi.org/10.1198/016214506000001437. (last accessed on 2020-06-14).

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial networks. arXiv:1406.2661. (last accessed on 2020-06-21).

Gorti, S.K., Ma, J., 2018. Text-to-image-to-text translation using cycle consistent adversarial networks. CoRR abs/1808.04538. URL: http://arxiv.org/abs/1808.04538, arXiv:1808.04538. (last accessed on 2020-05-14).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A., 2017. Improved training of wasserstein gans. arXiv:1704.00028. (last accessed on 2020-06-16).

Hass Avocado Board, 2018. Avocado price. URL: https://www.kaggle.com/neuromusic/avocado-prices. (last accessed on 2019-10-12).

Hewamalage, H., Bergmeir, C., Bandara, K., 2019. Recurrent neural networks for time series forecasting: Current status and future directions. arXiv:1909.00590. (last accessed on 2020-06-10).

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9, 1735–80. doi:10.1162/neco.1997.9.8.1735. (last accessed on 2020-06-27).

Holt, C.C., 2004. Forecasting seasonals and trends by exponentially weighted moving averages. International Journal of Forecasting 20, 5 – 10. URL: http://www.sciencedirect.com/science/article/pii/S0169207003001134, doi:https://doi.org/10.1016/j.ijforecast.2003.09.015. (last accessed on 2020-06-12).

Holton, G., 2014. Value-at-risk: Theory and practice second edition, e-book. URL: https://www.value-at-risk.net/. (last accessed on 2020-06-26).

Husein, A., Arsyal, M., Sinaga, S., Syahputa, H., 2019. Generative adversarial networks time series models to forecast medicine daily sales in hospital. SinkrOn 3, 112–118. URL: https://jurnal.polgan.ac.id/index.php/sinkron/article/view/10044, doi:10.33395/sinkron.v3i2.10044. (last accessed on 2020-06-10).

Hyndman, R., Athanasopoulos, G., 2018. Forecasting: principles and practice, 2nd edition. OTexts: Melbourne, Australia. URL: https://otexts.com/fpp2. (last accessed on 2020-06-02).

Hyndman, R., Koehler, A., Ord, K., Snyder, R., 2008. Forecasting with exponential smoothing. The state space approach. doi:10.1007/978-3-540-71918-2. (last accessed on 2020-06-25).

Hyndman, R.J., Koehler, A.B., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679 – 688. URL: http://www.sciencedirect.com/science/article/pii/S0169207006000239, doi:https://doi.org/10.1016/j.ijforecast.2006.03.001. (last accessed on 2020-06-02).

Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR abs/1502.03167. URL: http://arxiv.org/abs/1502.03167, arXiv:1502.03167. (last accessed on 2020-06-02).

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., Aila, T., 2019. Analyzing and improving the image quality of stylegan. arXiv:1912.04958. (last accessed on 2020-05-25).

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization. International Conference on Learning Representations arXiv:1412.6980.

Koochali, A., Dengel, A., Ahmed, S., 2020. If you like it, gan it. probabilistic multivariate times series forecast with gan. arXiv:2005.01181. (last accessed on 2020-06-15).

Koochali, A., Schichtel, P., Dengel, A., Ahmed, S., 2019. Probabilistic forecasting of sensory data with generative adversarial networks – forgan. IEEE Access PP, 1–1. doi:10.1109/ACCESS.2019.2915544. (last accessed on 2020-06-20).

Liashchynskyi, P., Liashchynskyi, P., 2019. Grid search, random search, genetic algorithm: A big comparison for nas. arXiv:1912.06059. (last accessed on 2020-06-08).

Lucas, T., Tallec, C., Verbeek, J., Ollivier, Y., 2018. Mixed batches and symmetric discriminators for gan training. arXiv:1806.07185. (last accessed on 2020-06-18).

Lucic, M., Kurach, K., Michalski, M., Gelly, S., Bousquet, O., 2017. Are gans created equal? a large-scale study. arXiv:1711.10337. (last accessed on 2020-06-08).

Makridakis, S., Hibon, M., 2000. The m3-competition: results, conclusions and implications. International Journal of Forecasting 16, 451 – 476. URL: http://www.sciencedirect.com/science/article/pii/S0169207000000571, doi:https://doi.org/10.1016/S0169-2070(00)00057-1. (last accessed on 2020-06-12).

Makridakis, S., Spiliotis, E., Assimakopoulos, V., 2020. The m4 competition: 100,000 time series and 61 forecasting methods. International Journal of Forecasting 36, 54 – 74. URL: http://www.sciencedirect.com/science/article/pii/S0169207019301128, doi:https://doi.org/10.1016/j.ijforecast.2019.04.014. (last accessed on 2020-06-10).

Mani, K., 2019a. Lstm cell. URL: https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1. (last accessed on 2020-01-10).

Mani, K., 2019b. Rnn cell. URL: https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1. (last accessed on 2020-01-10).

Meteorologisk Institutt (MET) , 2019. Oslo mean temperature. URL: https://wiki.math.ntnu.no/lib/exe/fetch.php?tok=5deb8a&media=https%3A%2F%2Fwww.math.ntnu.no%2Femner%2FTMA4285%2F2019h%2Fpdf%2Fdata.xlsx. (last accessed on 2019-10-12).

Mirza, M., Osindero, S., 2014. Conditional generative adversarial nets. arXiv:1411.1784. (last accessed on 2020-06-10).

N. Srivastava, G. E. Hinton, A.K.I.S., Salakhutdinov, R., 2014. Dropout: A simple way to prevent neural networks from overfitting URL: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf. (last accessed on 2020-05-25).

NASA Earth Observatory, 2002. Weather forecasting through the ages. URL: https://earthobservatory.nasa.gov/features/WxForecasting/wx2.php. (last accessed on 2020-05-07).

Opland, M., 2020. Forecasting uncertainty in neural networks with dropout. URL: http://folk.ntnu.no/mathiaop/. (last accessed on 2020-06-26).

Radford, A., Metz, L., Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv:1511.06434. (last accessed on 2020-06-02).

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1986. Learning Representations by Back-propagating Errors. Nature 323, 533–536. URL: http://www.nature.com/articles/323533a0, doi:10.1038/323533a0. (last accessed on 2020-06-25).

Santurkar, S., Tsipras, D., Ilyas, A., Madry, A., 2018. How does batch normalization help optimization? arXiv:1805.11604. (last accessed on 2020-06-02).

Schmidhuber, J., 1990. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. URL: http://people.idsia.ch/~juergen/FKI-126-90_(revised)bw_ocr.pdf. (last accessed on 2020-05-23).

Schmidhuber, J., 1991a. Learning factorial codes by predictability minimization. URL: http://people.idsia.ch/~juergen/FKI-126-90_(revised)bw_ocr.pdf. (last accessed on 2020-05-23).

Schmidhuber, J., 1991b. A possibility for implementing curiosity and boredom in model-building neural controllers, in: J. A. Meyer and S. W. Wilson, editors, Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pages 222–227., MIT Press/Bradford Books. (last accessed on 2020-05-23).

Schmidhuber, J., 2019. Generative adversarial networks are special cases of artificial curiosity (1990) and also closely related to predictability minimization (1991). arXiv:1906.04493. (last accessed on 2020-05-23).

Seabold, S., Perktold, J., 2010. Statsmodels: Econometric and statistical modeling with python, in: 9th Python in Science Conference.

Sezer, O.B., Gudelek, M.U., Ozbayoglu, A.M., 2019. Financial time series forecasting with deep learning : A systematic literature review: 2005-2019. arXiv:1911.13288. (last accessed on 2020-05-25).

Smith, T.G., et al., 2017-. pmdarima: Arima estimators for Python. URL: http://www.alkaline-ml.com/pmdarima.

Smyl, s., R.J., Pasqua, A., 2018. M4 forecasting competition: Introducing a new hybrid es-rnn model. URL: https://eng.uber.com/m4-forecasting-competition/. (last accessed on 2020-06-10).

Steinwart, I., Christmann, A., 2011. Estimating conditional quantiles with the help of the pinball loss. Bernoulli 17, 211–225. URL: http://dx.doi.org/10.3150/10-BEJ267, doi:10.3150/10-bej267. (last accessed on 2020-06-10).

Tolosana, R., Vera-Rodriguez, R., Fierrez, J., Morales, A., Ortega-Garcia, J., 2020. Deepfakes and beyond: A survey of face manipulation and fake detection. arXiv:2001.00179. (last accessed on 2020-05-25).

Tsay, R.S., 2000. Time series and forecasting: Brief history and future research. Journal of the American Statistical Association 95, 638–643. URL: http://www.jstor.org/stable/2669408. (last accessed on 2020-06-20).

Turner, R., Hung, J., Frank, E., Saatci, Y., Yosinski, J., 2018. Metropolis-hastings generative adversarial networks. arXiv:1811.11357. (last accessed on 2020-06-18).

Wang, P., 2019. https://thispersondoesnotexist.com/. (last accessed on 2020-05-25).

Winters, P.R., 1960. Forecasting sales by exponentially weighted moving averages. Management Science 6, 324–342. URL: https://doi.org/10.1287/mnsc.6.3.324, doi:10.1287/mnsc.6.3.324, arXiv:https://doi.org/10.1287/mnsc.6.3.324. (last accessed on 2020-06-18).

Yang, D., Hong, S., Jang, Y., Zhao, T., Lee, H., 2019. Diversity-sensitive conditional generative adversarial networks. arXiv:1901.09024. (last accessed on 2020-05-10).

Yao, Z., Gholami, A., Arfeen, D., Liaw, R., Gonzalez, J., Keutzer, K., Mahoney, M., 2018. Large batch size training of neural networks with adversarial training and second-order information. arXiv:1810.01021. (last accessed on 2020-06-18).

Yoon, J., Jarrett, D., van der Schaar, M., 2019. Time-series generative adversarial networks, in: Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., Garnett, R. (Eds.), Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 5508–5518. URL: http://papers.nips.cc/paper/8789-time-series-generative-adversarial-networks.pdf. (last accessed on 2020-06-10).

Zhang, K., Zhong, G., Dong, J., Wang, S., Wang, Y., 2019. Stock market prediction based on generative adversarial network. Procedia Computer Science 147, 400 – 406. URL: http://www.sciencedirect.com/science/article/pii/S1877050919302789, doi:https://doi.org/10.1016/j.procs.2019.01.256. 2018 International Conference on Identification, Information and Knowledge in the Internet of Things. (last accessed on 2020-06-10).

Zhou, X., Pan, Z., Hu, G., Tang, S., Zhao, C., 2018. Stock market prediction on high-frequency data using generative adversarial nets. Mathematical Problems in Engineering 2018, 1–11. doi:10.1155/2018/4907423. (last accessed on 2020-06-10).

Zhu, J., Park, T., Isola, P., Efros, A.A., 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. CoRR abs/1703.10593. URL: http://arxiv.org/abs/1703.10593, arXiv:1703.10593. (last accessed on 2020-05-14).

Zhu, L., Laptev, N., 2017. Deep and confident prediction for time series at uber. 2017 IEEE International Conference on Data Mining Workshops (ICDMW) URL: http://dx.doi.org/10.1109/ICDMW.2017.19, doi:10.1109/icdmw.2017.19. (last accessed on 2020-06-10).

Appendix

Link to GitHub repository containing code used to run the experiments in this thesis `https://github.com/mattaop/ForGan/tree/Release/1.0`.

Hyperparameters	**Value**
Type of cells in the Generators first layer	Dense
Type of cells in the Discriminators first layer	Dense
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	2000
Batch size	32
Learning rate	0.0005
Discriminator iterations per training iteration	3

Table A1: Hyperparameters used for distribution estimation with GAN unless otherwise specified.

Hyperparameters	**Value**
Type of cells in the Generators first layer	Dense
Type of cells in the Discriminators first layer	Dense
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	3000
Batch size	1024
Learning rate	0.002
Discriminator iterations per generator epoch	3

Table A2: Hyperparameters used for bimodal distribution estimation with GAN and WGAN.

Hyperparameters	**Value**
Type of cells in the Generators first layer	SimpleRNN
Type of cells in the Discriminators first layer	SimpleRNN
Number of nodes per layer in Generator	16
Number of nodes per layer in Discriminator	64
Latent code dimension	100
Training iterations	5000
Batch size	32
Learning rate	0.0001
Discriminator iterations per generator iteration	5
Condition window length	24

Table A3: Hyperparameters used for the ForWGAN estimating the sine data.

Hyperparameters	**Value**
Type of cells in the first layer	RNN
Number of nodes per layer	64
Number of layers	3
Training iterations	32062 (2000 epochs)
Batch size	64
Learning rate	0.001
Optimizer	Adam
Dropout rate	0.4
Activation function first layer	tanh
Activation function second layer	ReLU
Activation function last layer	Linear
Loss	MSE

Table A4: Hyperparameters used for MC dropout model for forecasting on sine data.

Hyperparameters	**Value**
Type of cells in the first layer	RNN
Number of nodes per layer	64
Number of layers	3
Training iterations	5406 (500 epochs)
Batch size	64
Learning rate	0.001
Optimizer	Adam
Dropout rate	0.4
Activation function first layer	tanh
Activation function second layer	ReLU
Activation function last layer	Linear
Condition window length	24

Table A5: Hyperparameters used for MC dropout model for forecasting on Oslo temperature data set.

Hyperparameters	Value
Type of cells in the first layer	RNN
Number of nodes per layer	64
Number of layers	3
Training iterations	8184 (50 epochs)
Batch size	64
Learning rate	0.0001
Optimizer	Adam
Dropout rate	0.4
Activation function first layer	tanh
Activation function second layer	ReLU
Activation function last layer	Linear
Condition window length	26

Table A6: Hyperparameters used for MC dropout model for forecasting on avocado price data set.

Hyperparameters	Value
Type of cells in the first layer	RNN
Number of nodes per layer	64
Number of layers	3
Training iterations	8859 (500 epochs)
Batch size	64
Learning rate	0.0001
Optimizer	Adam
Dropout rate	0.4
Activation function first layer	tanh
Activation function second layer	ReLU
Activation function last layer	Linear
Condition window length	168

Table A7: Hyperparameters used for MC dropout model for forecasting on electricity consumption data set.