Generation of multivariate time series using neural networks

Jan Tijink

Technische Universiteit Delft

TU Delft Delft University of Technology

Challenge the future

Generation of multivariate time series using neural networks

by

Jan Tijink

to obtain the degree

Master of Science
in Applied Mathematics

at the Delft University of Technology,
to be defended publicly on Thursday November 22, 2018 at 10:00.

Student number:	4203240	
Project duration:	December 1, 2017 – November 22, 2018	
Thesis committee:	Prof. dr. C. W. Oosterlee,	TU Delft, supervisor
	Dr. C. Kraaikamp,	TU Delft
	Dr. Ir. W. den Dunnen,	EY, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft Delft University of Technology

Contents

1

Introduction

In this thesis scenarios are generated for time series by modelling the future based on the past events of this time series. This is done by using neural networks to estimate the distribution at the next time step. Before delving into the details of this approach, the necessity and usefulness of an approach to generate (possibly multivariate) scenarios is justified.

When looking at the global financial crisis around 2007-2009, it is often considered that lack of market consistent valuation within banking and insurance was a contributing factor to this crisis [1]. After this crisis the Solvency II directive and IFRS 9 framework were implemented to take future economic scenarios into account for banks and insurers to ensure that they keep enough capital on hand to handle different economic scenarios. So, in essence, economic scenario generation is a component of preventing a potential future crisis and thus a part of the stability of the financial system. Under the IFRS 9 framework banks and insurers are required to compute their Expected Credit Loss (ECL) based on the current state of the economy but it should be forward looking. This means that they have to base their estimates for the probability of default of customers on different future scenarios. These different scenarios are now commonly generated based on expert judgement, which obviously has flaws such as bias and incompleteness.
Not only economic scenarios are being generated, there are many other fields in which scenario generation is useful. Scenario generation is a broad term: trying to sketch scenarios for Brexit is also generating scenarios and it is dependent on time. However, in this thesis the focus will be on mathematical time series: measurable quantities that are ordered in time.

Scenario generation is clearly linked to uncertainty, if there were no uncertainty the result is deterministic and although this does not guarantee that the outcome is known, it makes modelling it much easier in general. Consider the situation where the temperature has been 15 degrees Celcius for the last three days and the goal is to generate scenarios for what happens to the temperature tomorrow. If we were to predict a 40% probability of 14 degrees, 40% probability of 15 degrees and a 20% probability of 16 degrees and it turns out to be 16 degrees tomorrow.

An analogy to the problem at hand is trying to predict tomorrow's weather based on the information of today and the past few days. If it has been hot in the past few days the probability that it is hot tomorrow is larger than usual. It is evident that the current state of the atmosphere and weather influences what the weather tomorrow will be, which can be observed in the measured temperatures.

Finally, from a mathematical perspective scenario generation is interesting. First let us focus on the difference between forecasting and generation.

Now that the base of reasoning as to why scenario generation is useful is given, let us consider how to generate these scenarios. To this end there are several approaches which will be given in this section. By giving examples the value of using distributions to forecast will be shown.

Before the elaboration on these methods there is a focus on scenario generation and future prediction in general. For example consider the simple random walk X_t with $X_{t+1} = X_t + D_{t+1}$ where $D_t = \pm 1$ each with probability $\frac{1}{2}$. Suppose $X_t = 0$ and we try to predict the value at the next time step X_{t+1}. Clearly at the next time step we will have $X_{t+1} = \pm 1$. If we were to forecast the expectation we would predict 0 at the next time step. This is obviously a prediction that always will be wrong. So using the expectation as a prediction can result in values that can never materialize. Furthermore, the mean does not have to be a typeical situation which we would expect. This can also be seen in the example given below.

One possible solution to ensure we do not predict a value that can never occur is to take the most likely event as the prediction. This has its flaws as will be described in the following example.

Consider the asymmetric random walk Y_t with $Y_{t+1} = Y_t + D_{t+1}$ where

$$D_t = \begin{cases} 99 & \text{with probability } \frac{1}{100}, \\ -1 & \text{with probability } \frac{99}{100} \end{cases}.$$

Suppose $Y_t = 0$ and we try to predict the value at the next time step Y_{t+1}. If we were to forecast the expectation we would predict 0 at the next time step, just like in the example before. If we take the most likely event as a prediction we will get a rising Y_t for every time step. In the long run we can see that this is clearly not desirable since the time series would forever increase, while its expectation is 0.

Note also that the two methods of taking the mean or mode at every time step only give the same result every time instead of generating different possible scenarios. Doing this over and over gives the same result, which is not what we are looking for when generating scenarios.

Another approach would be to give a confidence interval for the forecast. While this might work and give some extra insight next to the point estimation, it does not necessarily tell us much about the shape or symmetry. Therefore, a method which shows what could happen in the future is desirable. This is also the reason that forecasting distributions makes sense. We are not certain what is going to happen, thus we try to predict with what probability each event can occur. This gives the possibility for better understanding than a point estimator. We can see how spread out the possibilities are, which values are much more likely than others and what is the inherent uncertainty in the process. Now this is all for considering prediction. Note that point estimators are inherently bad at generation of scenarios, since they give little understanding of the uncertainty of the process. Note that from the distribution we can still compute the expected value and the most likely event, thus this is a more informative prediction, in fact it contains all the information that can be used to compute properties of the random variable.

So how do we generate a time series? A logical approach is to consider all possible events that could happen at the next time step and take this into account for our prediction. If we do this at each (future) time step we can see all possible scenarios. To put this in a more mathematical formulation: if we estimate the distribution at the next time step we can generate a scenario by simply taking a draw from the distribution. Essentially, we are doing Monte Carlo simulation by repeatedly predicting distributions based on the past and sampling. This allows us to generate scenarios, but also to indirectly approximate the distribution several time steps ahead.

Many methods currently used in forecasting as described in the previous section are based on distributions. The only notable exception is expert judgment. This boils down to asking an expert question on what (s)he expects to happen. Note that the phrasing plays a role on this from a psychological perspective already and it is hard to extract the views of several experts and reach a clear consensus. Expert judgement has another drawback; experts can be just as wrong as models. If you asked experts on interest rates around 15 years ago whether the interest rate between banks and for the central banks could become negative they would have said this is not going to happen. This has however happened in the past few years.

Another aspect to consider is that scenarios should not only depend on the empirical historical distribution. In almost all fields where scenario generation is used it is possible for more extreme events to take place. Think of a new warmest day record, highest amount of precipitation but also new highs/lows in the stock market. So, the scenario generation method must be able to produce extreme scenarios.

However, it also needs to be realistic. For example, if we are predicting temperatures in some physics experiments the temperatures can approach 0 degrees Kelvin. If a scenario is generated which goes below the 0 degrees Kelvin, it defies the laws of physics and therefore is unrealistic. This balance is clear in the physics sense but is less clear in economics. How does one judge whether a scenario is realistic?

Note that the approach used in this thesis could be used for distributional forecasting in general. This has the advantage that it is more widely applicable and more interesting for future research. Having the information about all possible events at the next time step gives you much more information than solely a confidence interval and the mean and can be used in practice in situations such as financial trading and energy load forecasting.

Thus predicting a distribution is useful both for generation of scenarios and insight into inferences. The method of forecasting distributions is not present in all areas where predictions are based since most rely on point estimates. However the fields in which most research is done with regard to distributional forecasting are numerical weather prediction [2], Natural Language Processing [3], reinforcement learning [4], image classification [5] and energy load forecasting [6].

Scenario generation is a hard task in general, but when considering multivariate time series it becomes especially involved. Consider the example of macroeconomic variables: there is surely a dependence between the interest rate, unemployment rate, GDP and other variables. In which way these variables exactly depend on each other is not entirely clear. Therefore choosing a flexible model is required, that accommodates the unknown dependence structure. Not only should the model thus be able to incorporate different dependence structures and allow for a multivariate approach, it should also be able to be dependend on historic values in several different ways and incorporate randomness. Due to this flexible requirements it is chosen not to pursue a parametric model for this, as further explained in Chapter 3.

An investigation into different scores for probabilistic forecasts is done and it is also based on another distinction. If we would have five events A, B, C, D and E that can happen and they are equally likely it is straightforward to forecast the probability for all of them. However, does it make a difference if you care about being close to the result as well. So do we only care about predicting the right probability for each part? The answer to this question depends on the context of the events meaning in which way are they linked. If we are talking about the temperature as in case of the weather earlier and we predict event B while the realization is event E, we would prefer to be closer to the realization than further away, so D would have been a better prediction even though not being correct. A thorough analysis is presented in chapter 3.

To investigate time series we use neural networks to learn to predict a distribution instead of the more common point estimate. From the estimated distribution we draw a sample to generate a possible future event. This is done repeatedly to obtain a possible future realization. Furthermore, the issue of generating in higher dimensions is addressed.

In the recent years advances in computing and neural networks have reached a new height. After computers started outperforming humans on chess, the game of Go, which has now been fully dominated by computers. The best performing project on Go is made by Google and named AlphaGoZero. It is a machine which makes its decisions based on a neural network. This is not the only field in which neural networks shine, they also achieve state-of-the-art performance in areas such as image recognition, computer games [4] and natural language processing. However, in the field of time series they do not always have the best performance. Great progress was made by Google Deepmind on the generation of audio using WaveNet [7].

Artificial Neural Networks (mostly referred to as neural networks) represent a machine learning technique. They are modelled after the human brain, hence the term neural network. The idea is that as input we have certain neurons that activate with an energy level. This information is then passed onto other neurons, and based on the combination of the previous input it activates in a certain way. This chain continues until eventually a conclusion is drawn. Think of recognizing an image of a cat. How does one recognize a cat? In the image you see color, context, edges, etc. The combination of these things makes one confirm: this is a cat. A neural network tries to make inference by recognizing patterns in the data. The neural network learns the weights and functions from examples, data for which the realization of the prediction is known. By training on these examples they learn to identify patterns which will allow them to distinguish between different objects. In practice, they serve in many applications. The recommendations seen online are often based on predictions from neural networks, such as when listening to Spotify recommended music or similar items in the webshops that are shown often come from predictions by neural networks.

Historically, predictions were often made using analytical methods. These analytical methods have the benefit of producing mathematically consistent solutions to these problems and can often be used for a range of problems due to generalization. With the decrease in the cost of computing power and advances in modelling and machine learning approaches, simulation has become a standard tool in finance.
Statistical models such as ARIMA are able to generate distributions and are used often in time series analysis. A downside is that they assume a linear stochastic model which is also Gaussian. Note that in reality this might not be the case, so we would be restricting the model for ease of computation and expressibility. "Literature in time series forecasting is rich and has a long history in the field of econometrics which makes extensive use of stochastic models such as AR, ARIMA and GARCH ... . In practice, one can notice that these models 'over-fit' on financial time series: their parameters are unstable and out-of-sample performance is poor."[8]. This is a problem since the goal is to use the predictions out of the sample. Note also that in reality the stochasticity might not be linear or Gaussian. The ARIMA model can capture this but then we have to estimate its parameters, which is another problem to which there is not a clear solution.

Neural networks have achieved state-of-the-art performance in classification tasks. Predicting a distribution is also a classification task but with a (perhaps) more complicated dependence structure between the classes. Since neural networks are good at classifying it is interesting to see if they are also good at predicting distributions and there has been not much research into this as stated before.

It is important to understand that a neural network is only trained on data. Although the network architecture (the amount of layers, activation functions, amount of neurons, etc.) is usually chosen specifically for the problem, the neural network boils down to a function approximator. There are theoretical results that certain setups of neural networks are able to approximate arbitrary non-linear functions [9]. However, if the data is not representative or of bad quality or there is a change in the underlying dynamics over time, the neural network will not be able to perform well. Where a model is (usually) based on a theoretical principle, the predictions of a neural network could lead to unrealistic results at times if not properly trained or supplied with bad data. This is important to keep into account.

When finally time series are generated and we have a framework to forecast distributions, the issue of measuring the performance of this rises. It is clear that it is a hard task to assess the goodness of the forecasted distribution for a single realization. If the underlying distribution of the data is known it is workable to evaluate the forecasted distribution. When the underlying distribution is not known, this is much harder. To this end a new method is introduced to assess how uniformly distributed the realizations are with regards to the expected distribution.

In Chapter 2, the more elaborate problem description and outline of the thesis are given as well as a theoretical background on the most common time series modelling approaches, such as ARIMA, Bayesian methods and machine learning approaches. In chapter 3 the methodology of this thesis to approach the problem is explained and motivated based on the information given in Chapter 2. In chapter 4 the set-up for the numerical experiments is given and explained, whereas the results from these are presented and discussed in chapter 5. In chapter 6 the conclusions are drawn and finally in chapter 7

further research recommendations are discussed and drawbacks of the approach used are highlighted.

2

Problem description

This chapter will give a more in depth view into the problem of generating multivariate time series. The main question to answer is: How can multivariate time series be generated without explicitly knowing how these different series depend on each other or how they are distributed? This question will be answered by first considering which well known time series forecasting approaches there are to choose from. Furthermore, their advantages and disadvantages will be discussed including how they handle multiple dimensions.

First, a brief overview of time series is given. A time series is a collection of data points ordered in time. In most cases the size of the time step is constant, for example a day, or an hour or a week. A simple example of a time series is the daily temperature outside at noon in Amsterdam. The temporal structure is what makes the difference between most other parts of statistics since it is often the idea that the future is in some way dependent on the past.

Time series analysis is applied in many different research areas due to the broadness of the applicability of temporal datapoints. Examples are: statistics, pattern recognition, econometrics, finance, weather forecasting, earthquake prediction. Usually a model is not readily available or known to accurately describe the behaviour.

It was notable in the literature that the score or error metrics were always given, but not compared to naive methods or tested for statistical significance. Often it seemed that a time series modelling approach would work well on one problem but perform terrible on other problems or datasets. This of course makes sense, some methods are excellent at one specific task but underperforming in other tasks. Since it is unclear what the underlying dynamics are in economic time series this thesis will seek a method which works well on different types of datasets and assess performance on different series to assess robustness.

The three main classes to time series analysis and forecasting are given below and will be disucssed in their own section:

1. ARIMA models
2. Bayesian time series models
3. Machine learning approaches.

2.1. ARIMA models overview

In this thesis only discrete time series will be discussed, this means that the time steps are discrete. The definitions and notation used here are similar to those used in general books on time series such as [10–12]. Mathematically we have the collection

$$\{X(t,\omega) : t \in \mathbb{Z}, \omega \in \Omega\}.$$

We denote the stochastic process by $X_t := X(t, \omega)$ for ease of notation and since it is clear we are dealing with a random variable we can omit the ω. When working with time series, a criteria that is often used is stationarity.

Definition 2.1.1 (Strongly stationary process)**.** A stochastic process $\{X_t : t \in \mathbb{Z}\}$ is strongly stationary if the distribution of

$$(X_t, X_{t+1}, \dots, X_{t+h})$$

is independent of t for any $h \in \mathbb{N}$.

Simply said, a time series is strongly stationary (sometimes called strictly stationary) if for all sets of indexed time series, so for any length, the distribution stays the same when applying any size of time shift. This means that no matter at what time we look at the time series, the distribution behaves the same. Furthermore, if the mean exists it is also constant over time. This definition is too strict for most useful applications. Moreover it is difficult to assess from a dataset whether or not the time series is strictly stationary. Therefore a slightly weaker but in practice more useful definition of stationarity will be used. Before this is introduced some measures of dependence have to be explained.
The mean of a time series X_t with density f_{X_t} is given by $\mu_t = \mathbb{E}(X_t) = \int_{-\infty}^{\infty} x f_{X_t}(x) \mathrm{d}x$. Another aspect of time series that is often used is the autocovariance. It is defined as

$$\gamma_X(s,t) = \mathrm{Cov}(X_s, X_t) = \mathbb{E}\left[(X_s - \mu_s)(X_t - \mu_t)\right] \tag{2.1.1}$$

for all $s, t \in \mathbb{Z}$. If it is clear which time series we are referring to we will notate $\gamma(s,t)$ instead of $\gamma_X(s,t)$. Note that covariance is symmetric and by the definition so is the autocovariance.
These are used in the definition of a weakly stationary process:

Definition 2.1.2 (Weakly stationary process)**.** A stochastic process $\{X_t : t \in \mathbb{Z}\}$ is weakly stationary if the following conditions are met.

1. The mean $\mathbb{E}(X_t) = \mu_t$ does not depend on t and is thus constant over time.
2. The autocovariance function $\gamma(t, t+h) = Cov(X_t, X_{t+h})$ may depend on time solely by the absolute difference $|h|$.
3. The variance of X_t is finite for all $t \in \mathbb{Z}$.

When in this thesis a series is referred to as (non) stationary, the term weakly stationary is meant.
Note that any strongly stationary process that has a finite variance automatically is also stationary in the weak sense.
Several examples of stationary processes will now be given that are often used. A series $\{X_t\}$ is white noise if the random variables are uncorrelated with zero mean and finite and constant variance σ^2. This means that $\mu_t = 0$ for all $t \in \mathbb{Z}$ and

$$\gamma(s,t) = \begin{cases} \sigma^2 & \text{if } s = t \\ 0 & \text{else} \end{cases}$$

Clearly this is a stationary process. If we have a series of i.i.d. normally distributed variables with zero mean and variance σ^2 this is a white noise series.

2.1.1. AutoRegressive model

In this section the AutoRegressive (AR) model will be introduced. The autoregressive model has an associated order p with it. Autoregressive means that the future is a function of the past value; it regresses on itself. Mathematically this means

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-i} + \epsilon_t.$$

Here ϵ_t is white noise and the φ_i are parameters with $\varphi_p \neq 0$.
This assumes that the mean μ of X_t is 0. If this is not the case we can use a transformation of $X_t - \mu$ and achieve a similar representation. For ease of notation this is now omitted.

Consider the AR(1) model. The model equation is given by

$$X_t = \varphi_1 X_{t-1} + \epsilon_t.$$

This series can be rewritten as

$$\begin{aligned} X_t &= \varphi_1 X_{t-1} + \epsilon_t = \varphi_1 \left(\varphi_1 X_{t-2} + \epsilon_{t-1}\right) + \epsilon_t \\ &= \varphi_1^k X_{t-k} + \sum_{i=0}^{k-1} \varphi_1^i \epsilon_{t-i} \end{aligned}$$

By letting k go to infinity, we can see, if $|\varphi_1| < 1$ and $\sup_t Var(X_t) < \infty$, that

$$X_t = \sum_{i=0}^{\infty} \varphi_1^i \epsilon_{t-i}.$$

Clearly,

$$\mathbb{E}\left(X_t\right) = \varphi \mathbb{E}(X_{t-1}) = \varphi_1^t \mathbb{E}(X_0) = 0.$$

Furthermore,

$$\begin{aligned} \mathrm{Cov}(X_t, X_{t-h}) &= \mathbb{E}(\varphi_1 X_{t-1} \cdot X_{t-h}) + \mathbb{E}(\epsilon_{t-1} \cdot X_{t-h}) \\ &= \mathbb{E}(\varphi_1^h X_{t-h} \cdot X_{t-h}) = \varphi_1^h \mathbb{E}\left(X_{t-h}^2\right) \\ &= \varphi_1^h \mathrm{Var}(X_{t-h}) = \varphi_1^h \mathbb{E}\left(\left[\sum_{i=0}^{\infty} \varphi_1^i \epsilon_{t-h-i}\right]^2\right) \\ &= \varphi_1^h \mathbb{E}\left(\sum_{i=0}^{\infty} \sigma^2 \varphi_1^j \varphi_1^j\right) = \varphi_1^h \sigma^2 \sum_{j=0}^{\infty} \varphi_1^{2j} \\ &= \varphi_1^h \sigma^2 \frac{1}{1-\varphi_1^2} \end{aligned}$$

The last equality holds since $|\varphi_1| < 1$ thus we have convergence. Clearly this shows that this is a stationary time series since the mean is constant zero and the autocovariance is a function of the difference in time.
Consider the AR(1) process with φ_1=1. This process is also known as a random walk or the discrete Brownian motion. This is not a stationary as can be seen from the autocovariance.

2.1.2. Moving Average model

In this section the Moving Average (MA) model will be introduced. The moving average model has an associated order q with it. The moving average model is based on a weighted sum of previous values of random variables. Mathematically, this means

$$X_t = \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}.$$

Here ϵ_t is white noise and the θ_i are parameters with $\theta_q \neq 0$.
This is similar to the AR process but here it is based on a weighted sum of random previous values and not on the value of X_{t-1}. Taking the MA(1) process we get

$$X_t = \epsilon_t + \theta_1 \epsilon_{t-1},$$

which has

$$\gamma(t, t+h) = \begin{cases} (1+\theta_1^2)\sigma^2 & h = 0, \\ \theta_1 \sigma^2 & h = \pm 1, \\ 0 & |h| > 1. \end{cases}$$

2.1.3. ARMA model

An obvious combination of the previous two models is the AutoRegressive Moving Average (ARMA) model. A process is called an ARMA(p,q) process, if we can write

$$X_t = \epsilon_t + \sum_{i=1}^{p} \varphi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}.$$

This is a combination of the previous two models and allows for a greater flexibility of processes to be modelled.

2.1.4. ARIMA model

The ARIMA model can be seen as an ARMA model on the d times differenced original series. So. an ARIMA(p,d,q) model has

$$Y_t = X_t - \sum_{i=1}^{d} X_{t-i}.$$

Then the ARMA component becomes

$$Y_t = \epsilon_t + \sum_{i=1}^{p} \varphi_i Y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i}.$$

The main strength of the ARIMA is that even if the original time series is not stationary, it can often become stationary by differencing a number of times. This allows an even greater class of possible models.

2.1.5. Box-Jenkins methodology

The ARIMA model was popularised for time series by Box and Jenkins in the 1970s for their flexibility. A problem with this was the choice of the orders p, d and q and estimating the corresponding parameters. For this purpose they developed the Box-Jenkins methodology which gives guidelines on how to fit an ARIMA type model to a time series. It follows these steps:

- Determine stationarity. If non-stationary, then take a one step time difference of the series to obtain the new series $X_t - X_{t-1}$ and repeat this step until a stationary series is obtained. The stationarity is determined by looking at the data for seasonal patterns and observing the AutoCorrelation Function (ACF) at different lags as well as the Partial AutoCorrelation Function (PACF) at different steps.
- Propose a model for the (possibly differenced) time series based on the ACF and PACF. Estimate the parameters corresponding to these models using regression or maximum likelihood.
- Check for goodness of fit of the model by testing whether the residuals are white noise.

The maximum likelihood techniques used as well as the (non-linear) least squares approach are often used for parameter estimation. This is one of the challenges using these types of models due to the wide range of possible order choices and parameter choices to be made.

The ARIMA model is often used to predict the mean and under the assumption of weak stationarity will provide the best linear predictor in terms of the Mean Squared Error (MSE), which is an attractive property. However, in this thesis we are not interested in solely predicting the mean of the next time step but instead interested in the full distribution. This is a problem since only knowing that a series is white noise is not enough to know the distribution at the next time step. Therefore, a choice has to be made into modeling the error term which might not be desirable. Moreover, according to [8] they over-fit on financial time series and their out-of-sample performance is poor.

The ARIMA model is a type of model that works well in a single dimension but when expanding to multiple dimensions we arrive at Vector AutoRegression (VAR) type models for which we also have to specify the dependence structure. More on the dependence structure and multivariate forecasting will be stated later.

2.2. Bayesian Time series forecasting

The Bayesian approach to statistics in general is based on the following global steps

1. Formulate a prior distribution using the available information;
2. Obtain new data;
3. Find posterior distribution based on the new data;
4. Update the prior distribution;
5. Repeat from step 2.

This is opposed to the frequentist approach where these parameters are estimated (for example using maximum likelihood). These parameters are assumed to be fixed but not known. In opposite of the frequentist approach, in the Bayesian approach parameters are assumed to have some distribution and the beliefs about these parameters are updated when more information becomes available.
Mathematically, this means

$$p(\theta|y_{1:t}) \propto p(y_{1:t}|\theta)p(\theta).$$

Now applying this to a time series scenario is more involved and is most often done in Hidden Markov models, see [13]. Before the Hidden Markov Model is explained, some more information about Markov Chains and Markov Chain Monte Carlo (MCMC) methods have to be given.

2.2.1. Markov Chains

A Markov chain is a discrete-time stochastic process $X_1, X_2, \ldots$ taking values in an arbitrary state space that has the Markov property of order m: the future is independent of the past given the information in the last m states. In mathematical terms: the distribution of X_{t+1} depends on $X_t, X_{t-1}, \ldots, X_{t-m+1}$ but not on anything of the time before that. A Markov chain has two properties that fully describe its behaviour: an initial state and the transition probabilities. Here we will assume that the distribution of X_{t+1} depends on $X_t, X_{t-1}, \ldots, X_{t-m+1}$ in the same way for all t. With induction it is now clear that we have the full distribution for all future time steps.
Applying Markov chains in practice means that we do not know the transition probability beforehand. A stationary Markov chain $X_1, X_2, \ldots$ is a stationary stochastic process, but it does not need to be real-valued. If g is a real-valued function on the state space of the Markov chain, then $g(X_1), g(X_2), \ldots$ forms a stationary real-valued stochastic process. Note that it is not necessarily a Markov chain, since conditioning on $g(X_t)$ instead of X_t may not give the Markov property. However, the process $g(X_1), g(X_2), \ldots$ has many attractive properties. It is called a "functional" of the original chain.
The general notion of a Markov Chain Monte Carlo (MCMC) method is to estimate probabilities or expectations by simulating a Markov chain and averaging over the simulations. The probabilities or expectations calculated are those for functionals $g(X_i)$ of the stationary chain, thus they are probabilities or expectations with respect to the invariant distribution.
The most often used MCMC algorithm is the Metropolis-Hastings algorithm which is very flexible. The Metropolis–Hastings algorithm works by generating a sequence of sample values in such a way that, as more and more sample values are produced, the distribution of values more closely approximates the desired distribution $P(x)$. These sample values are produced iteratively, with the distribution of the next sample being dependent only on the current sample value (thus making the sequence of samples into a Markov chain). Specifically, at each iteration, the algorithm picks a candidate for the next sample value based on the current sample value. Then, with some probability, the candidate is either accepted (in which case the candidate value is used in the next iteration) or rejected (in which case the candidate value is discarded, and the current value is reused in the next iteration)—the probability of acceptance is determined by comparing the values of the function $g(x)$ of the current and candidate sample values with respect to the desired distribution $P(x)$. This acceptance probability is chosen in such a way that the limiting distribution of the Markov Chain is the desired distribution $P(x)$. An algorithmic overview is given below.
Now in the Hidden Markov Model the states of the Markov chain are not observable (hence the name hidden). This makes it difficult to assess in which state the model currently is and therefore inference becomes harder. Using a Bayesian approach ensures that based on the information available the belief

Algorithm 1 Metropolis - Hastings algorithm

procedure Initialization
  Pick an initial state X_0.
  Have the transition probabilities as $P(x'|X_t)$.
  $t = 0$
procedure Iterate
  Generate candidate x' according to $g(x'|X_t)$.
  Compute the acceptance probability $A(x'|X_t) = \min\left(1, \frac{P(x'|X_t)}{P(X_t|x')} \frac{g(X_t|x')}{g(x'|X_t)}\right)$.
  Generate u uniformly from $[0, 1]$.
  if $u \leq A(x', X_t)$ **then**
    New state accepted!
    $X_{t+1} = x'$
  else
    New state rejected!
    $X_{t+1} = X_t$
  $t = t + 1$

about the current state can be updated and this can then be used to compute the posterior and thus drawing from this the next state. The MCMC algorithm provides a way of finding the correct parameters for the model.

2.3. Machine learning approaches

Machine learning is an approach of not specifying a specific parameteric assumption on the model or data, such as priors or normal distributions, which is often done in other types of models. These machine learning methods extract information about the data in order to use on new data.

Before delving into the machine learning approaches for time series we will focus on neural networks and their basic approaches.

First, the basics are introduced with the most simple network. Second, the main neural network approaches are discussed. Thirdly, an investigation is made on the different loss functions. Finally, a roundup of other important features and considerations is given.

Artificial Neural Networks (ANNs) are computer systems that resemble the biological workings of our brains. These computer systems are said to learn tasks from examples, often without specifically being told what to look for. A research area where these networks are often used is image recognition and a task might be recognizing if an image contains a cat or does not contain a cat. The network learns from images which contain cats and images where there are no cats. This is all without any prior knowledge of what a cat looks like or other properties of cats.

The term artificial will be omitted for the rest of this thesis as it refers to the fact that we are not talking about a biological neural network, the original basis for the computing structure now referred to as neural networks. A neural network works by taking a combination of inputs, giving them an adjustable weight and applying a function to this as an output signal. Neural networks work by specifying a structure from the input to the output with neurons and connections between these neurons. Usually this is done by grouping the neurons in layers. The weights in the connections between these neurons are then learned by examples to produce the desired output.

2.3.1. Feedforward Neural Network

One of the most simple version of a neural network is the Feedforward Neural Network. This neural network starts with the input and a layer of K neurons. These K neurons are all connected to the input and all these connections have their own weights. Every neuron also has its own bias. An activation is applied to the weighted sum combined with the bias to produce the output for that neuron. It is possible to stack several layers by simply using the output just described as input for the next layer. Eventually, the output of the network is computed.

To formulate this mathematically we get an input $\mathbf{x} = (x_1, \ldots x_O))$ and have a network with L layers and M_l neurons for layer $l \in \{1, \ldots, L\}$. We want to predict Y based on this. The activation a in the first

layer for neuron i becomes:

$$a_1(i) = \sum_{j=1}^{t} w_1(i,j)x_j + b_1(i)$$

where w_1 is the weight matrix of dimension $\mathbb{R}^{M_1 \times t}$ with bias $b_1 \in \mathbb{R}^{M_1}$. This is visualized in Figure 2.3.1. Now the activation function comes in. These outputs are transformed using the activation function g:

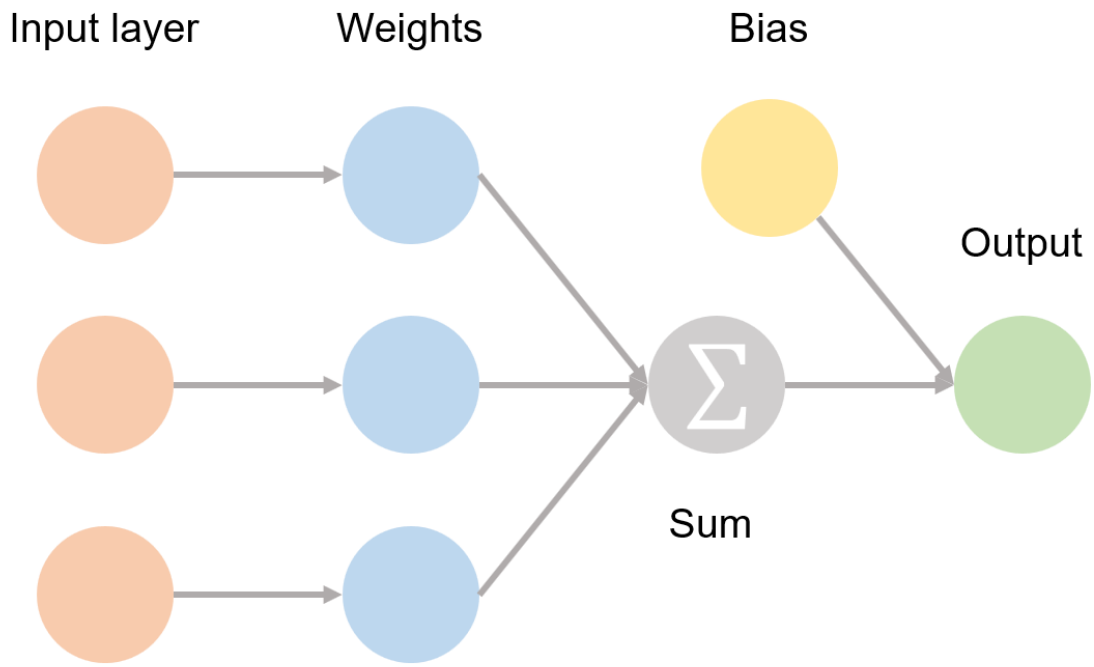

Figure 2.3.1: Visualization of the basic workings of neurons in neural networks.

$$f_1(i) = g(a_1(i))$$

Now all these different outputs of the first layer are again inputs for the second layer. So, in layer $l \in \{2, \dots, L-1\}$ we have

$$f_l(i) = g\left(\sum_{j=1}^{M_{l-1}} w_l(i,j)f_{l-1}(j) + b_l(j)\right)$$

This is done for every layer and the result of this from the final layer is simply the output of the network. So $\hat{x}(t+1) = f_L = g\left(\sum_{j=1}^{M_{L-1}} w_L(j)f_{L-1}(j) + b_L(j)\right)$. This means that every neuron in a layer is connected to all the neurons in the previous layer and to the next layer, often called a fully connected layer. Stacking these together yields a network that is visualized in 2.3.2.

The architecture of a neural network refers to the setup of the neural network and comprises the input size, output size, internal connections from the inputs to internal nodes and to the outputs as well as the functions used in these parts.

A mathematical definition of the network architecture will now be given. Let $f : \mathbb{R} \to \mathbb{R}$ be the activation function. Let the input $I \subseteq \mathbb{R}^{d_1}$ where d_1 is the dimension of the input. Let the output $O \subseteq \mathbb{R}^{d_2}$ where d_2 is the dimension of the output. Let V be the nodes of a graph, usually referred to as the hidden units. Let E be the directed edges corresponding to the graph with nodes V, I and O. Each edge in E has a corresponding activation function with its own parameters.

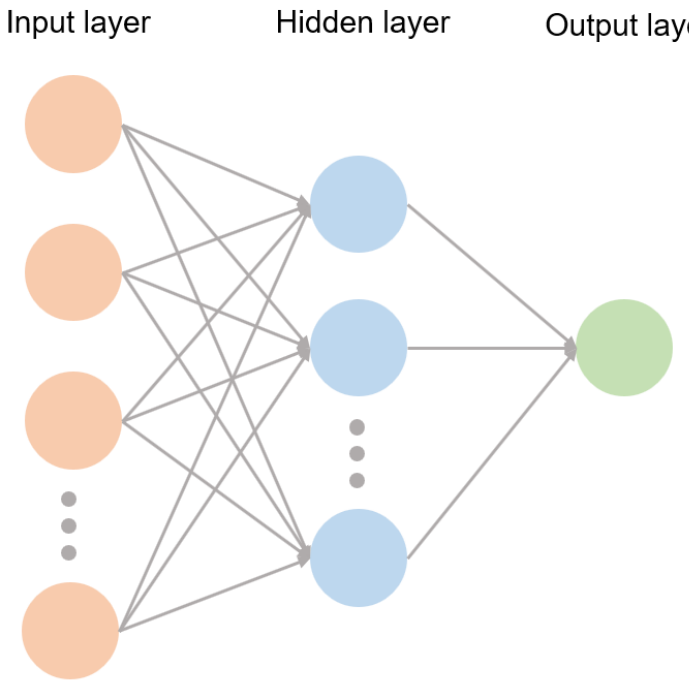

Figure 2.3.2: Image overview of a feedforward neural network with a single hidden layer.

In practice all of the hidden nodes N are divided into several layers N_1, N_2, etc. where I is only connected to the inputs in N_1, the edges from N_1 go to N_2 up to the last N_M which connects to O. This ensures that the network is a so called feed-forward network and does not contain any cycles. In recurrent neural networks this is not the case.
By stacking several layers such as in Figure 2.3.2 a so called deeper neural network is created. This has multiple layers of neurons before the output is reached.

2.3.2. Activation functions

The architecture just described only shows how the neurons are connected. Up to this point no activation functions were introduced. Hence we can in that case only use sums and weights of the inputs (and sums and weights of these) we can essentially reduce this via a linearity argument to a weighted sum of the input. Thus we get a linear function as an end result. Since, in general, it is desirable to be able to model more complex non-linear behaviour activation functions are required. In the construction of a neural network the use of non-linear functions is essential to allow the network to learn arbitrary non-linear functions.
There is a myriad of available activation functions. The ones that are most used in practice are the Rectified Linear Unit (ReLU), softmax and the hyperbolic tangent function. Note that these functions are differentiable (almost everywhere), making it easy to as to compute the gradients of the loss function with respect to the parameters.
An activation function in a neural network is any function $g : \mathbb{R} \to \mathbb{R}$. To ensure that networks are capable of being trained, we require the activation functions to be continuous and differentiable almost anywhere.

ReLU

The Rectified Linear Unit (ReLU) is a function $g : \mathbb{R} \to \mathbb{R}$ given by:

$$g(x) = \max(0, x).$$

The ReLU is a very simple function that has an intuitive explanation in machine learning. Only if the input passes a certain threshold does it activate, this can be used as a trigger for certain events and can thus be very useful. Note that it is not differentiable at $x = 0$ but the derivative is very simple at all other points, as it is 0 when $x < 0$ and 1 if $x > 0$. This makes computation of the gradients during the training fast and ensures that the network does not suffer from vanishing or exploding gradients. This will be explained in more detail in Chapter 3.

Softmax

The softmax is a function $g : \mathbb{R}^K \to \mathbb{R}^K$ given for $j = 1, \dots, K$ by:

$$g_j(\mathbf{x}) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}.$$

The softmax function has the property that the sum of the entire output adds up to one, which is very useful for probabilities and classification tasks.
Its derivative with respect to x_m is

$$\begin{aligned}\frac{\partial g_j(\mathbf{x})}{\partial x_m} &= \frac{\partial}{\partial x_m}\left(\frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}}\right) \\ &= \frac{e^{x_j}\left(-e^{x_m} + \delta_{j,m} e^{x_m} \sum_{k=1}^{K} e^{x_k}\right)}{\left(\sum_{k=1}^{K} e^{x_k}\right)^2}.\end{aligned}$$

Since we are dealing with exponentials here it is easy to see that this is infinitely often differentiable. Before the ReLU was widely used the most used activation functions were the softmax function and the hyperbolic tangent, which is now given.

Hyperbolic tangent - tanh

The hyperbolic tangent is a function $g : \mathbb{R} \to \mathbb{R}$ given by:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The hyperbolic tangent rescales the input to the range of -1 and 1 and is often used in LSTM networks which will be introduced in section 2.3.4.

2.3.3. Time series forecasting using machine learning

There are two main approaches to time series forecasting in machine learning:

- Recurrent Neural Networks
- Convolutional Neural networks

Although there are more approaches available such as feed-forward neural networks [14], copula networks [15] and Generative Adversarial Networks (GANs) [16], the recurrent neural networks and convolutional neural networks are the most prevalent and have achieved the best results to date. For more approaches or details on neural networks one may consult books such as [17].

2.3.4. Recurrent Neural Network

The idea of the recurrent neural network is that we have a state (comparable to a Markov chain) and we plug in the output of the previous timestep. This is of course why it is called recurrent. Essentially, the past inputs and activations of the network have an impact on the new input. This greatly increases the training complexity since when you update your parameters the output for the previous step would also change. An example of a RNN is the Long-Short Term Memory (LSTM) approach introduced by Hochreiter and Schmidhuber [18]. The main difference between Feedforward-Neural Networks and Recurrent Neural Networks is that in the latter some part of the activations or output are passed on to the next time step. So the input at time t has an effect from the previous input at time $t - 1$. This is clearly demonstrated in Figure 2.3.3.

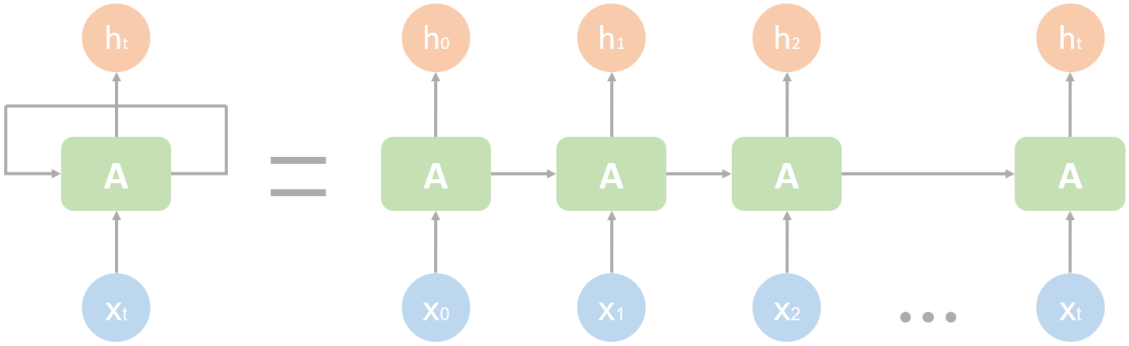

Figure 2.3.3: An unrolled recurrent neural network.

One of the appeals of RNNs is the idea that they are able to connect previous information to the present task. The downside of this approach is that sometimes we may need more information then just the previous state. Consider trying to predict the last word in the text "I grew up in France so I speak fluent _____." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. Sometimes, the only information necessary was just given. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the _____," no further context is needed; it's clear that the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information. For larger gaps this is harder due to vanishing gradients over time.

LSTM

Long Short Term Memory networks (LSTMs) are a type of Recurrent Neural Network that are capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber in 1997 [18]. They work tremendously well on a large variety of problems, and are now widely used in practice.
Recurrent neural networks are based on a chain of repeating modules of a neural networks. In the standard RNN this repeating system has a simple structure such as a single activation layer, e.g. a ReLU or tanh layer. The main idea behind the LSTM is to have a certain state that can be updated based on every new input. There are so called gates that influence what information goes in and out.

A multidimensional approach for the cells, the input and output of an LSTM is possible, but for ease of understanding a one-dimensional version will be presented. The first step in developping an LSTM is deciding which information will be deleted from the cell state and is decided by the first gate. This is a softmax function σ from the output of the previous time step h_{t-1} and the new input x_t.

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

where b_f is the bias and w_f is the weight matrix, which here only has two entries. Note that the term $w_f \cdot [h_{t-1}, x_t]$ is simply the inner product. A higher activation implies more information is kept. The next step is choosing which information goes into the state C_t and is composed of two parts. Another softmax σ is used to decide which values to update. This is

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

Now a tanh layer is used to give potential values to add to the state and this is $\tilde{C}_t$:

$$\tilde{C}_t = \tanh(w_C \cdot [h_{t-1}, x_t] + b_C).$$

Now the state is updated using what came before

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

Finally, the output of the state is generated and then multiplied by the tanh of the state. This gives us the final two equations

$$o_t = \sigma(w_0 \cdot [h_{t-1}, x_t] + b_0)$$
$$h_t = o_t \cdot \tanh(C_t).$$

So altogether the following equations are used iteratively.

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f),$$
$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i),$$
$$\tilde{C}_t = \tanh(w_C \cdot [h_{t-1}, x_t] + b_C),$$
$$C_t = f_t * C_{t-1} + i_t \cdot \tilde{C}_t,$$
$$o_t = \sigma(w_0 \cdot [h_{t-1}, x_t] + b_0),$$
$$h_t = o_t * \tanh(C_t).$$

2.3.5. Convolutional Neural Networks

The convolutional neural network (CNN) is a different approach to the previous networks. The easiest way to explain the technique behind a convolutional neural network is to look at images. An image has pixels, each of which has a color that can be represented by three color channels: Red, Green and Blue (RGB). The image is thus specified by many pixels each of which have three values corresponding to the RGB colors and these values range between 0 and 255. When looking at only a single pixel it is hard, if not impossible, to say that this pixel is part of a tree. It is important to look at the context. The idea behind convolutional networks is to extract so called features from the image. This is done by taking a small square of the image and using a filter on this producing an output. This filter is often referred to as the kernel or feature detector. The filter takes an area around a certain pixel and uses weights to compute a value. This filter is then slided over the entire image to give every part a value. These filters are constructed in such a way as to detect certain features. As an example, when considering images take a filter that gives a high value for edges, which are useful in detecting shapes. On a higher level, we can use a combination of edges to get the contours of a shape, and use this to classify what is actually in the image. Note that like in the previously described networks a non-linear function is applied to the outcome of the layers. An example of this can be seen in Figure 2.3.4.

These convolutions are learnable, the weights they apply to each of their inputs is optimized during training. A choice for the filter size has to be made.
When talking about a discrete convolution $*$ between $u = (u_0, \dots, u_{N-1})$ and $v = (v_0, \dots, v_{N-1})$ we define

$$(u * v)(i) = \sum_{j=0}^{N-1} u_j v_{i-j}$$

for $i \in \{0, 1, \dots, N-1\}$.
This leaves the problem of the values that were not defined. Taking $i = 1$ will leave us with values that are not defined such as v_{-5}. In practice this is solved by filling these values in with zeros, a technique often called padding.
Convolutional layers have a filter size and an amount of filters. Suppose we have the same input $x = (x_0, \dots, x_{N-1})$ again. Let us have the ReLU activation function $g(x) = \max(x, 0)$ and weight vector $w \in \mathbb{R}^{1\times k\times 1}$ with k the filter size. By convolving every weight of the filter with the input we obtain:

$$(w_1 * x)(i) = \sum_{j=0}^{N-1} w_i x_{i-j}$$

Afterwards a non-linear function is applied and this becomes the output of the convolutional layer. In practice there are multiple filters, ensuring that w becomes a matrix instead of a vector.
The convolutional layer has learnable filters. We decide on a filter size beforehand and slide this filter over the image. The output becomes the inner product between the input and the weights. This is

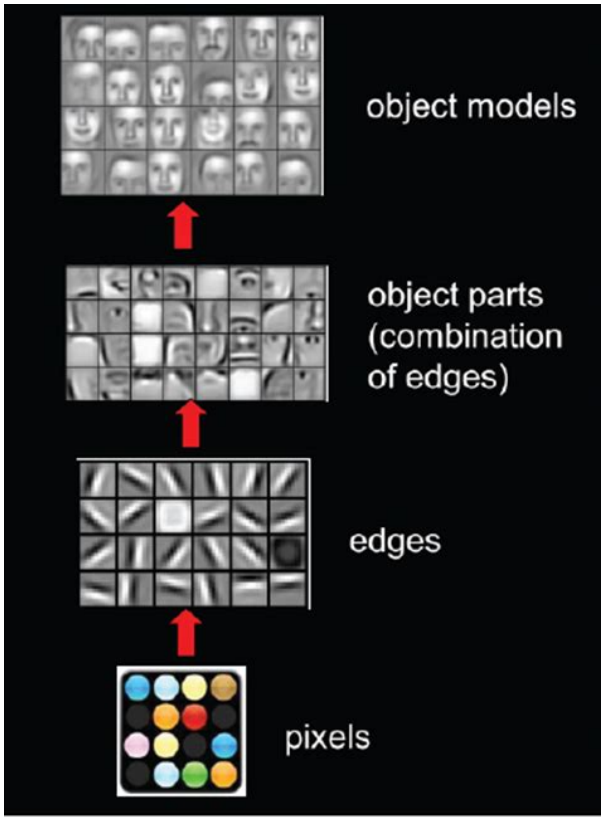

Figure 2.3.4: Overview of the way a convolutional neural network recognizes images by [19].

usually described as learning a certain feature, because in an image we have many edges. To detect them we do not always need the entire image, a small part will do. The same can hold for time series, we do not always need the global picture, the combining of local effects may be sufficient. There are significantly less parameters when using convolutional layers, this is usually an advantage over the fully-connected layers. Furthermore due to the reduced dimensionality convolutional neural networks suffer less from overfitting [20].

Dilated convolutions and WaveNet

In this section the WaveNet approach used in [7, 21] is introduced that uses dilated convolutions to process a long input sequence with relatively little computational cost.
Take a one-dimensional time series $(X_t)_{t=1}^N$. Suppose there is a model with parameters θ and the goal is to predict the density at the next time step. By Bayes Theorem, we can rewrite this to:

$$\begin{aligned} p(X_N, X_{N-1}, \dots, X_1|\theta) &= p(X_N, X_{N-1}, \dots, X_2|X_1, \theta)p(X_1|\theta) \\ &= p(X_N, X_{N-1}, \dots, X_3|X_2, X_1, \theta)p(X_2|X_1, \theta)p(X_1|\theta) \\ &= \prod_{t=1}^{N} p(X_t|X_1, \dots X_{t-1}, \theta) \end{aligned}$$

Since the data has a time component and it is undesirable to have a model which depends on future realizations causal convolutions are used. A benefit of this method as opposed to a RNN is that for a longer sequence these models are much faster to train due to the lack of the recurrent connection. A problem when using causal convolutions is that many layers are required, or large filters, to increase the receptive field size.
To this end, dilated convolutions are used to increase the receptive field without significantly increasing the computational cost. A dilated convolution is a convolution where the filter is applied over an area larger than its filter length by skipping input values with a certain step size. Its equivalent is a convolution with a larger filter with zeros for the steps skipped, but in practice this is less efficient. Dilated

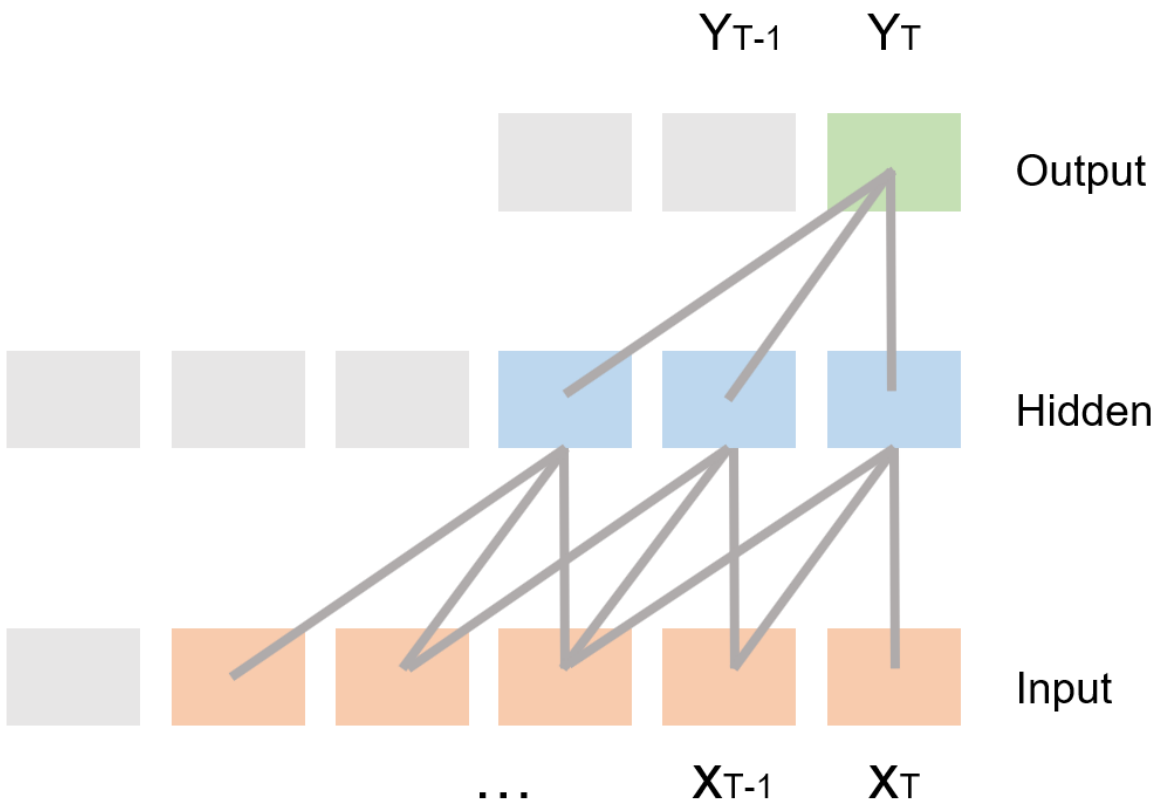

Figure 2.3.5: Visualization of a causal convolution with a kernel/filter size of three.

convolutions allow a neural network to have a larger receptive field than with normal convolutions. The dilated convolutions were introduced by Yu in [22]. Note that a dilation of 1 produces the standard convolution. When increasing the dilation per layer an interesting pattern can be seen. Let us take 4 layers where the first layer has dilation 1, the second layer has dilation 2, the third layer has dilation 4 and the fourth layer has dilation 8.
It is clear that when we have L layers with filter size k we obtain a receptive field r with the size of $r = k^L$. In almost all cases a filter size of 2 is used. In the WaveNet paper [7] they take a filter size of 2 and double this dilation of every layer up to 512. This gives a receptive field of 1024 with only 10 layers. This can clearly be seen in Figure 2.3.6.
Mathematically we can represent the dilated filter as

$$(w *_d x)(i) = \sum_{j=0}^{N-1} w_j x_{i-d \cdot j}.$$

2.3.6. Data dependence

Since neural networks are fully trained on the input data and loss corresponding to their output, it is very important to have good data quality that is representative of what is happening. A shift in the underlying behaviour in the dataset will likely have disastrous results for the outcome of the network. This is represented in the popular statement: ”garbage in is garbage out”.
The goal is to eventually generate economic time series. To generate a time series using neural networks we must train the network on data. But if the network is not able to learn the actual underlying distribution we do not want to generate a scenario from this. So, t we test the network on several types of artificially crafted data sets.
The idea is to see what kind of dependence structures the network is able to capture. We will look at deterministic non-linear functions, independent random variables, dependent random variables and a

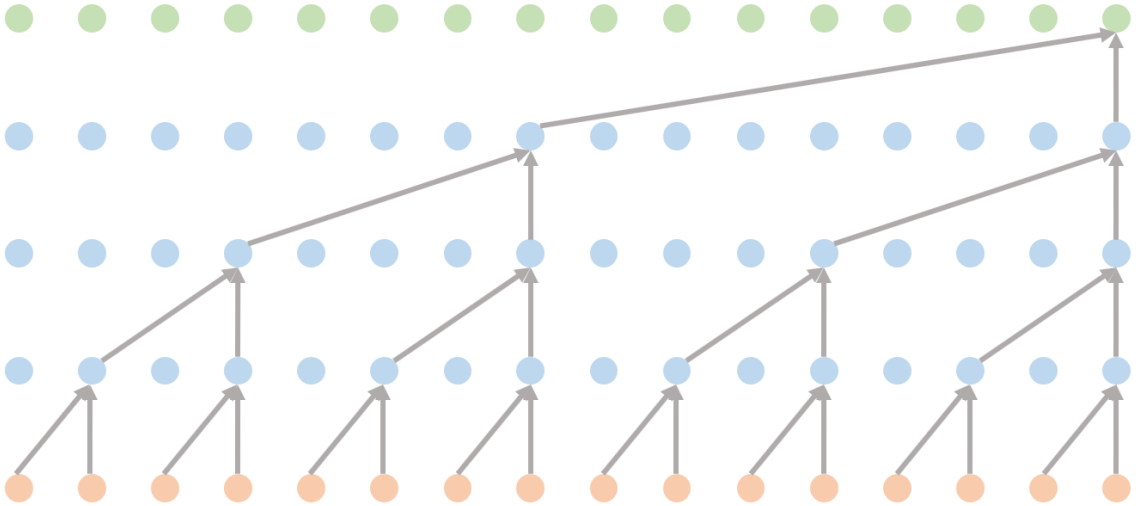

Figure 2.3.6: Visualization of stacked dilated convolutions with dilations $1, 2, 4, 8$ and filter size 2.

combination of these. The goal of this is twofold. Firstly, it is desirable to see in which situations the neural network is able to learn the structure and perform well. Currently neural networks are known as "black-box" methods. This could gain more insight as to how and why neural networks perform. Secondly, if the network is not able to perform well on these datasets for which we know the dependence structure, we cannot expect it to do well on the economic time series, since we have even less knowledge and information about these. Also note that it will be interesting and important to check whether the generated economic time series are realistic and not take impossible values.

2.3.7. Forecasting and error metrics

Note that the distribution we forecast has only one realization for each time step. It is desirable to assess the quality of the forecast based on a score or metric. This is also necessary in the neural network, it requires a loss function which is to be minimized. The idea is that the lowest loss is given for the best fitting distribution. This is similar to the point forecast. For point forecasts we usually define the loss function as the Mean Square Error (MSE). For an unbiased estimator, minimizing the MSE is equivalent to minimizing the variance of the estimate. There are many other loss metrics available and they are applicable or desired in certain cases. The loss or error function which is to be minimized depends on the purpose of the optimization. Sometimes the variance can be the most important part, but in other cases (for example when not all observations have equal variance) other measures can be more suitable.

The same holds for distributions. When considering a prediction in which category an image falls it is important to get the class exactly right. Suppose we have an image of a cat. If our method categorizes this as a cat with 90% probability and 10% as a tree, or as a cat with 90% probability and 10% as a bicycle it makes little difference to us. But when predicting the stock market returns split into different intervals, there is a significant difference between predicting a gain of 0.5 with 90% probability and a loss of 20 with 10% probability, or a gain of 0.5 with 90% probability and a gain of 20 with 10% probability. In the first case distance is not important, while in the second case it is.

The main loss function used in neural networks for distributions , primarily used for classification tasks is the cross-entropy loss. Since this thesis is about economic scenarios it is clear that distance is important. Therefore we will focus on a measure that takes this into account: the Ranked Probability Score. This was first introduced in numerical weather prediction, precisely because a way to incorporate distance into the evaluation of their forecasts was needed. More details and derivations will be given in chapter 3.

2.4. Multivariate generation

When multivariate generation of a random variable or time series is required it becomes more involved than for the univariate case. Note that we cannot simply see two time series as independent and generate from them disjointly. This can only be done if they are independent, which is a major assumption. We would need the joint distribution function. Since in the one-dimensional case we were trying to estimate this, this means we would have to simulate a two-dimensional distribution. If we have N bins, in one dimension, this would mean we get N^2 bins in two dimensions. It is clear that this implies that for larger dimensions the amount of bins blows up. Since we need to estimate the probability of it being in this bin, we would require sufficient data for this, which means that the data required also grows exponentially. This is also known as the curse of dimensionality.

There are two main approaches to solving this problem. The first is using copulas. This consists of two parts, the first part is that we can transform a continuous random variable to a uniform distribution with the inverse of its cumulative distribution function. The copula approach works by taking univariate uniform random variables and describing the dependence structure in a copula. This is essentially the multivariate distribution function. An attractive property about this is that the dependence structure and the marginals can be estimated separately. However, in practice, the copula has a parametrization and should be chosen beforehand and fitted correspondingly.

The second method is predicting the univariate distributions and sampling from this for one variable at a time. This means that if we have 2 time series X_t, Y_t and estimate the distribution from X_{t+1} from $X_{t-M}, \dots, X_t$ and estimate the distribution of Y_{t+1} from $Y_{t-M}, \dots, Y_t$. We now take a sample from the predicted distribution and call this sample $\widetilde{X}_{t+1}$. TWe then obtain the sample for Y_{t+1} by predicting the distribution based on $X_{t-M+1}, \dots, X_t, \widetilde{X}_{t+1}$ and $Y_{t-M}, \dots, Y_t$. Thus we iteratively generate a value. Note that this means that if there is a dependence between X_{t+1} and Y_{t+1} this dependence is preserved here. This is commonly known as Gibbs sampling, a special variant of the Markov Chain Monte Carlo framework. The downside of this method is that it is not possible to directly find the joint distribution.

When using copulas the shape of the dependence structure is already prescribed. Since we do not know exactly how the economic variables interact or are dependent, this is not desirable. Furthermore we would have to perform an extra optimization routine apart from the neural network optimization to determine the parameters. Note also that we specifically choose neural networks as a method because of their flexibility, choosing copulas would counter this somewhat. These downsides of copulas and the attractive alternative of Gibbs sampling leads to the decision to use Gibbs sampling to generate paths in this thesis. Note that the downside of not being able to directly find the joint distribution is not a problem, since that was not the goal of this thesis. If desired, this could still be found using Monte Carlo simulation, but this would be computationally intensive.

3

Methodology

Based on the methods described in Chapter 2, the method used in this thesis to generate scenarios is presented as well as a motivation behind the choices made. As with any model, assumptions must be made. First an overview is presented on the methodology. In the second section the assumptions made will be described as well as their advantages and disadvantages.
The goal is to have an approach that can generate scenarios based on a given time series. This approach must be able to incorporate randomness, effects over long and short term and multidimensional interaction without knowing these effects beforehand. Therefore a flexible approach is required that adapts the model based on the given time series.

3.1. Overview of model approach and assumptions

In this section a total overview of the methodology used in generating scenarios and the motivation for the choices made is given.

This approach will depend on the following four components:

1. Time series model approach that works based on distributions for each time step.
2. Procedure for fitting/optimizing this model approach to a given, possibly multivariate, time series.
3. Generation procedure to produce scenarios and that can handle multivariate problems, this is interlinked with the previous time series model.
4. Method of assessing whether the produced model output and scenarios make sense.

The need for a distribution based approach is clear due to the purpose of generation instead of only predicting the mean, as described before. Once a certain model type is chosen, it has to be calibrated/trained/optimized to fit the data set at hand. This approach should thus be broad since the underlying dynamics of the data set are not known nor are the dependence relations available. Then a procedure should be available to eventually produce different scenarios from this model, which is the final goal. Finally, the goodness of fit of the model has to be tested by comparison to known properties of data sets and other methods.
Note that these components are not completely standalone. The optimization or fitting procedure that has to be done clearly is dependent on the choice made for the time series model. Likewise, the generation procedure has to work with the time series model. Therefore it is important that the choices made are able to perform all tasks and satisfy the requirements.

3.1.1. Time series model

First of all a time series $\mathbf{X}_t \in \mathbb{R}^d$ is required. The method of forecasting the distribution is linked to the type of time series model being used. In Chapter 2, the most common approaches to time series forecasting have been discussed.

Since the underlying dynamics of the time series are not considered to be known beforehand or clearly visible from inspection, the ARIMA type models are not chosen. Furthermore, these models would not properly be able to forecast a distribution without assuming a certain shape on the distribution which is not the chosen way, since the underlying dynamics and distributions were assumed to be unknown. Moreover when forecasting in the multivariate sense, we would either need to fit multivariate versions of the ARIMA model which have other disadvantages or use copulas. This essentially means that a choice for the dependence would have to be made, which is opposite to the flexible approach sought in this thesis.

Therefore, a machine learning method is chosen. Due to the need of both a short term as well as a long term dependence on the history, a choice is made between the LSTM and the WaveNet approach. Currently the state-of-the-art approach is of the latter and is able to handle conditional input well. Therefore a WaveNet type approach is used.
The biggest assumption we make is that the distribution of the future X_t, conditionally on the last N realizations, is fully specified as a function of these realizations and independent of the time t. This can be univariate or multivariate. In mathematical terms:

$$F_{X_t}(X_t|X_{t-1}, \dots X_{t-N}) = g(X_{t-1}, \dots X_{t-N}),$$

for all t.
The assumption that is made here allows the flexibility to allow all kinds of distributions and does not describe a parametric approach while still allowing some structure. Note that this approach allows us to model any polynomial up to order N but also allows advanced random structures.

The model will not be effective if the underlying dynamics will change over time. Note that if the underlying dynamics change over time, and this is known, there is often already information available about this change of dynamics thus this could perhaps be accounted for.
The goal in the used approach is to estimate the function g with a neural network by training on data where the realization is known.

Now there are several methods of producing a distribution from the data, but the method used here is to approximate the distribution in certain fixed bins. The easiest way to look at this is a histogram. Fix the locations of the bins and estimate the probability that the realization will fall into that bin. We are taking M bins $[x_0, x_1), [x_1, x_2), \dots, [x_{M-1}, x_M)$ and aim to predict the categorical distribution.

Now suppose that the neural network (or any other method for this matter) is able to correctly establish the probabilities for each bin. We can see this as having a set of points $\{x_1, x_2, \dots, x_M\}$ for which we have the value of the cumulative distribution function $F_{X_t}(x_1)$, $F_{X_t}(x_2)$, ..., $F_{X_t}(x_M)$.

In this thesis the goal is to generate scenarios. This is done by approximating the underlying distribution using a neural network. The neural network gives the probability of being in a certain bin. MWe choose a set $x_0, \dots, x_N$ such that $P(Z \in [x_{i-1}, x_i)) = p_i$ for $i \in \{1, \dots, N\}$. These p_i are what the neural network produces as a function of the past. We can also describe this in terms of the Cumulative Distribution Function (CDF) for the points x_i and see that $F_Z(x_i) = \sum_{j=0}^{i} p_j$.

This methodology does not give us information about the distribution within these bins. So we know that $P(x_{i-1} \leq Z < x_i) = p_i$ but we do not know $F_Z(y)$ for $y \in (x_{i-1}, x_i)$ When generating scenarios this will be a problem.

We can choose between assuming that the distribution in the bin $[x_1, x_2)$ is a certain shape and work with this or assuming that we do not know the distribution on this interval and approximate it. If we approximate the distribution in any bin $[x_i, x_{i+1})$ with the uniform distribution, the result is a linear interpolation between $(x_i, F_{X_t}(x_i))$ and $(x_{i+1}, F_{X_t}(x_i))$. Since the cumulative distribution function is a monotonically increasing function some convergence results are available.

It is chosen to set the distribution in these bins to uniform, especially since the edges are flexible in the sense that they can be chosen by hand if desired. When looking at the CDF it could be seen as that

we have the points $F(x_i)$ and linearly interpolate between these points. Since the CDF is a monotonic increasing function adding points will ensure that if predicting p_i in N bins is as good quality as in $2N$ bins, the error will decrease.

When the distribution of X_t is discrete, with a finite amount of possibilities, it is straightforward to see that if we set the amount of bins to the amount of possibilities this approach should work. If however, the distribution is continuous or countably infinite, we can split any bin in 2 and since the cumulative distribution function is monotonic, we can have the same or a better result. If we let $N \to \infty$ the approximation will converge in distribution to F_{X_t}. Thus, using the uniform distribution on these bins is a valid approach.

If we let $N \to \infty$ we can approximate F_Z arbitrarily close, however, in practice this is unfeasible. Since the neural network trains on these bins we would have an ever increasing amount of bins which would decrease the performance of the prediction for each bin since there would be so many. Therefore, care should be taken when deciding on the amount of bins used in the approximation within the neural network. Too few bins and the approximation of F_Z will be bad, too many and the approximation of the p_i will suffer.

Another assumption currently in use is the boundedness of the time series, e.g. there will not be a more extreme value then the values that have already been observed and used for the training. Note that this is not a restriction of this model, but rather a choice to deal with the more extreme scenarios easily.

This is not necessarily a model restriction, as this model could be adapted to incorporate the more extreme events by taking the bins at the two bounds and defining a distribution such as the pareto or exponential distribution between the bounds and positive and negative infinity. One could choose to fit a distribution for this or a different model for the extremes and this would be fine with all other assumptions. In practice extreme scenario generation is already done with separate models and this approach allows for this flexibility.

As described in Chapter 2 the architecture consists of stacked dilated convolutions. This means the receptive field grows exponentially when more layers are used, allowing a large receptive field for relatively few parameters. The filter size in the original paper [7] is chosen to be two, but in this thesis also filter sizes k of 3 and 4 are investigated for the convolution to see if there is an impact on the performance. If we have L layers the dilation in layer i is f^i.

With this time series a neural network is used to forecast the distribution at the next time step. The neural network is based on stacked layers of dilated convolutions ensuring a rapidly expanding receptive field. On an intuitive level, this means that filters are applied to detect patterns over several time distances. So first on a local level with the time series entries next to it and then combining these patterns over bigger distances. In the end, a score is produced for each bin, one can see this as the likelihood of being in this bin. This is transformed to a probability using the softmax function, which corresponds to multinomial regression. The output of the network is thus the probability of being in each bin. From these probabilities we essentially have the CDF at several points. This means that we can sample from the distribution by inverse transform sampling: generating a standard uniform sample and translating this back to the inverse of this CDF by using linear interpolation. This linear interpolation means that the distribution on each bin is uniform. From this the sample is obtained. This means that based on the realizations on $(t - N, t - N + 1, \dots, t)$ we have generated a sample at $t + 1$. This sample is then used to forecast the distribution for $t + 2$ based on $(t - N + 1, \dots t)$ and the generated sample at $t + 1$.

The architecture similar to [7] will be used as described in Chapter 2. This means stacks of dilated convolutions will be used to process the input. Each of these layers will have ReLU activation functions. The final layer consists of the softmax function which produces a distribution. The softmax function σ transforms a $d-$dimensional input into a $d-$dimensional output, where for dimension $j \in \{1, \dots, d\}$m we have

$$\sigma_j(\mathbf{x}) = \frac{e^{x_j}}{\sum_{i=1}^{d} e^{x_i}}.$$

This ensures that we have $\sigma_j(\mathbf{x}) \geq 0 \quad \forall x \in \mathbb{R}$ and

$$\sum_{j=1}^{d} \sigma_j(\mathbf{x}) = \sum_{j=1}^{d} \frac{e^{x_j}}{\sum_{i=1}^{d} e^{x_i}} = \frac{\sum_{j=1}^{d} e^{x_j}}{\sum_{i=1}^{d} e^{x_i}} = 1.$$

Thus the total of the outputs sums up to one and is non-negative which is sufficient to state that it is a (discrete) probability distribution. Essentially this produces a categorical distribution where the probabilities here are dependent on the history and on the parameters in the neural network. The network is trained by optimizing the parameters such that the probability corresponds to the underlying distribution. More details on how this distribution is optimized will be given in section 3.3.
A filter size of 2 is chosen as in the literature. This is also compared to larger filter sizes of 3 and 4 in the results chapter to see if this has an impact.
The method can be seen in Figure 3.1.1 This approach allows the network to distinguish between

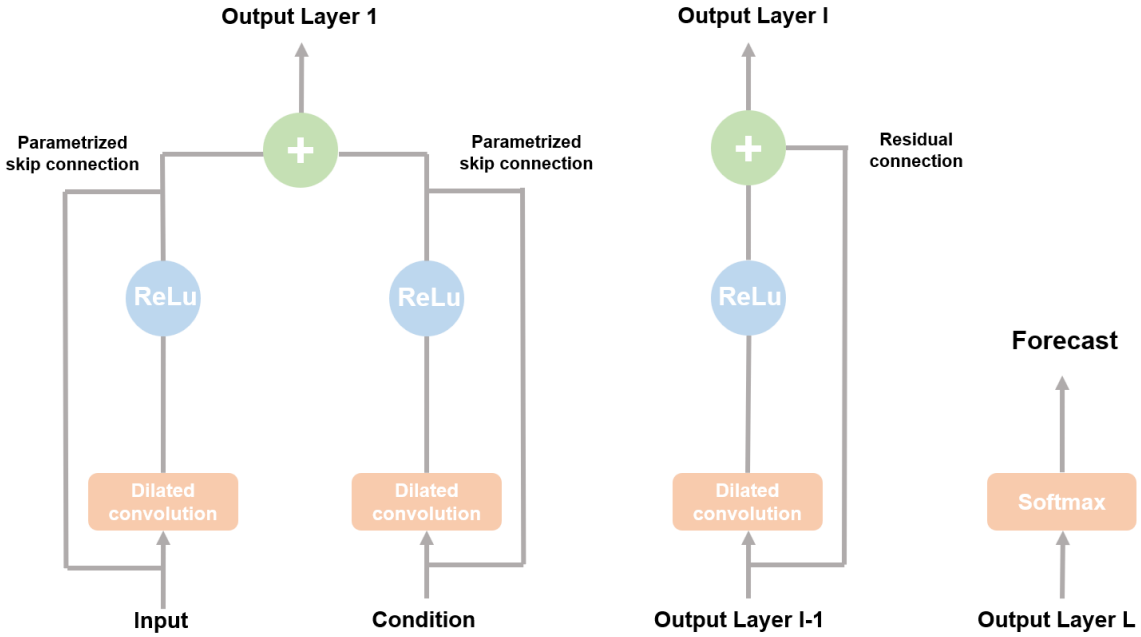

Figure 3.1.1: Overview of the neural network architecture used in this thesis.

useful and irrelevant information, is efficient to train and outputs a distribution. The parametrized skip connections were first introduced for neural networks by He [23]. These parametrized skip connections are a very simple structure, they are simply a 1×1 convolution, yet they are able to improve performance in many networks by learning identity mappings. In this thesis when working with multivariate series, the input length of the multivariate series will be the same as the other input, while this is not strictly necessary for this set-up. More details on the multivariate approach are given in section 3.4.

3.2. Scoring rules and error measures

In this section the difficulties of evaluating the performance of a forecast are discussed and theoretical properties of forecasts and scoring rules are introduced. A difficulty with the method of estimating a distribution is that the underlying distribution is not the same at each point in time, otherwise it would be identically distributed and we could use simpler methods such as the empirical distribution to estimate the actual distribution. Since we estimate a full distribution at each point in time and we have only one realization for this predicted distribution, it is not trivial to asses how good the forecast is. For example, suppose we try to predict the probability that it will rain tomorrow. If we say it is 70%

and it rains, how good is our estimate? How do we measure this? The difficulty of this question will have an impact on training the network, but also makes comparing this method with point forecasts harder. This was only for the binary event of rain or no rain, but in this thesis we will focus on a wider range. The realization will be a numerical value, but since it is hard to numerically approximate a continuous distribution, we will make the estimated distribution discrete by only considering a finite amount of bins for which to predict the probability.

Scoring rules provide a measure of the performance of a probabilistic forecast by giving a numerical score based on the forecast and the realization. In order to train the neural network we must choose a loss function. This choice has a large influence on the result since there is a link between what we are trying to optimize and which loss function is used. Note that during training we will attempt to minimize the loss function, thus we must understand what this means in terms of the data and output of the network. Think of the MSE and the variance for unbiased estimators. We define scoring rules as negatively oriented values that are given for a certain forecast and we want to minimize, similar to Gneiting [24] who provides a thorough overview of the field of probabilistic forecast evaluation and scoring rules.

For a distributional forecast $\mathbf{p}$ for the next time step with realization x the score becomes $S(\mathbf{p}, x) \in \mathbb{R} \cup \{-\infty\} \cup \{\infty\}$. Consider the distribution $\mathbf{q}$ (possibly different from $\mathbf{p}$) for the value at the next time step. Then we write $\mathbb{E}_{\mathbf{q}}(S(\mathbf{p}, x))$ for the expected value of $S(\mathbf{p}, \cdot)$ under the distribution $\mathbf{q}$. Now assume that $\mathbf{q}$ is the true underlying distribution. Clearly we would like to have that the lowest score is achieved if the forecast is identical to the true distribution. A scoring rule for which $\mathbb{E}_{\mathbf{q}}(S(\mathbf{q}, x)) \leq \mathbb{E}_{\mathbf{q}}(S(\mathbf{p}, x))$ for all $\mathbf{p}$ and $\mathbf{q}$ is called a proper scoring rule. If the equality only holds when $\mathbf{p} = \mathbf{q}$ the scoring rule is strictly proper. Note that this means that if we want to minimize the expected score, the best way to do this is to forecast the true distribution.

Since we are trying to generate time series based on the distribution, we wish to sample from the true distribution. Therefore a strictly proper scoring rule is desired. In this thesis we will focus on two scoring rules, the cross-entropy loss and the Ranked Probability Score.
Choosing a proper scoring rule for the evaluation of a forecast is like choosing the error function for a deterministic forecast. The loss function should be dependent on the desired goal.

3.2.1. Cross-entropy loss

The cross entropy loss, also known as Shannon entropy, or logarithmic score, for a distribution $\mathbf{p} = (p_1, p_2, \dots, p_N)$ is defined as:

$$S(\mathbf{p}, j) = -\sum_{i=1}^{N} \delta_{ij} \log p_j$$

where δ_{ij} is the Kronecker delta ensuring that it is 1 if $i = j$ and 0 otherwise. Under the actual distribution $\mathbf{q} = (q_1, q_2, \dots, q_N)$ the expectation is $\mathbb{E}[S(\mathbf{p}, \cdot)] = -\sum_{i=1}^{N} q_i \log(p_i)$. This is closely related to the Kullback-Leibler divergence which is sometimes used in statistics to describe the difference between two distributions. It is quite easy to see that this score is a strictly proper scoring rule. The score has the possibility of being infinite, which can lead to numerical problems in computing if there is zero probability assigned to a forecast bin where the realization lies.

The cross-entropy loss is without doubt the most widely used loss function in neural networks when working with distributions, especially in categorical forecasts.
Note that the cross-entropy loss is not symmetric, this is not a required property for a proper scoring rule. It can be shown that minimizing this loss function is equivalent to maximizing the likelihood, which is an attractive property since maximum likelihood estimation is done often in statistics.

As is shown in [25] this scoring rule behaves differently for likely and unlikely events. If we have events that are unlikely, it assigns a higher score to a forecast which gives a higher probability than a lower probability. If p is small and equal to the true probability and $\epsilon << p$ we have a higher score for $p + \epsilon$ then $p - \epsilon$. This works the other way around for the more likely events.

The reason this score might not be the best for the goal of forecasting time series is that location can matter. To illustrate this, consider the following example.

Let $\mathbf{q} = (q_1, \dots, q_{10})$ be the true distribution for the next time step. Let $\mathbf{f}$ and $\mathbf{g}$ be two different forecasts for the next time steps, or equivalently estimators of $\mathbf{q}$. Let $\mathbf{f} = (0.1, \dots, 0, 0.8, 0.1)$ and $\mathbf{g} = (0, \dots, 0.1, 0.8, 0.1)$. The forecasts only differ in the first and 8^{th} bin. Now suppose the realization x^* falls into bin 9.

$$S(\mathbf{f}, x^*) = -\log(0.8) = S(\mathbf{g}, x^*)$$

Clearly, the cross entropy loss assigns the same score to both forecasts. If the distribution is for categories, this can make sense, the category next to the realization might is likely not to be similar to the realization. If we consider weather forecasts or stock returns this is an entirely different story. Therefore we might want to consider a scoring rule which takes the location into account.

3.2.2. Ranked Probability Score

The ranked probability score was first introduced by Epstein [26] in 1969. The score or loss functions used at that moment would not take the distance of the forecast with respect to the observation into account but only the forecasted probability and whether or not this event happened. This makes sense if the forecast is made on images, we do not (necessarily) want to distinguish between how close a cat is to a dog. When forecasting the weather, being off 1 degree is better than being off 15 degrees. The cross-entropy loss does not take this into account. This is the reason this forecast score was introduced and is still in use today for numerical weather forecasting.

We define the Ranked Probability Score for the forecast $\mathbf{p} = (p_1, p_2, \dots p_N)$ with realization in bin j as:

$$\mathbf{RPS}_j = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\sum_{k=1}^{i} p_k - \delta_{kj} \right)^2$$

where δ_{kj} is the Kronecker delta which is 1 if $k = j$ and 0 for all other values of k and j. Note that the score is zero when $q_i = p_i$ for all i. The proof that the RPS is a proper scoring rule was not directly given for the definition we use in this thesis, but only for a different form by [27]. Therefore a proof for the strict properness of the Ranked Probability Score will be derived here, where we follow the line of the original proof but first rewrite many of the sums.

Consider the probabilistic forecast as $\mathbf{p} = (p_1, \dots, p_N)$ where N is the amount of bins used for the forecast. Furthermore let q_i denote the actual probability of the event being in bin i, resulting in the vector $\mathbf{q} = (q_1, \dots, q_N)$. Define the Ranked Probability Score (RPS) as

$$\mathbb{E}_{\mathbf{q}}(S(\mathbf{p}, x)) = \frac{1}{N-1} \sum_{i=1}^{N} \left(\sum_{j=1}^{i} p_j - \sum_{j=1}^{i} q_j \right)^2,$$

Notice that the terms inside the first sum are the cumulative distribution functions of respectively the forecast and the actual distribution.

We will rewrite the expressions to obtain a form for which we can determine the derivative of the expectation more easily. Given a realization in bin j we get (with dirac delta function for q):

$$S_j = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\sum_{k=1}^{i} p_k - \delta_{kj} \right)^2$$

Note that

$$\left(\sum_{k=1}^{i}(p_k-\delta_{kj})\right)^2=\begin{cases}\left(\sum_{k=1}^{i}p_k\right)^2 & i<j\\ \left(1-\sum_{k=1}^{i}p_k\right)^2 & i\geq j\end{cases}=\begin{cases}\left(\sum_{k=1}^{i}p_k\right)^2 & i<j\\ \left(\sum_{k=i+1}^{N}p_k\right)^2 & i\geq j\end{cases}.$$

This gives us

$$S_j=\frac{1}{N-1}\sum_{i=1}^{N-1}\left(\sum_{k=1}^{i}p_k-\delta_{kj}\right)^2$$
$$=\frac{1}{N-1}\left[\sum_{i=1}^{j-1}\left(\sum_{k=1}^{i}p_k\right)^2+\sum_{i=j}^{N-1}\left(\sum_{k=i+1}^{N}p_k\right)^2\right].$$

Using the equalities of the sums

$$\sum_{i=1}^{j-1}\sum_{k=1}^{i}p_k=\sum_{i=1}^{j-1}(j-i)p_i\qquad\text{and}\qquad\sum_{i=j}^{N-1}\sum_{k=i+1}^{N}p_k=\sum_{i=j+1}^{N}(i-j)p_i,$$

we obtain

$$S_j=\frac{1}{N-1}\left[\sum_{i=1}^{j-1}\left(\sum_{k=1}^{i}p_k\right)^2+\sum_{i=j}^{N-1}\left(\sum_{k=i+1}^{N}p_k\right)^2\right]$$
$$=\frac{1}{N-1}\left[\sum_{i=1}^{j-1}\left(\sum_{k=1}^{i}p_k\right)^2+\sum_{i=j}^{N-1}\left(\sum_{k=i+1}^{N}p_k\right)^2\right]+\frac{1}{N-1}\sum_{i=1}^{j-1}(j-i)p_i-\frac{1}{N-1}\sum_{i=1}^{j-1}\sum_{k=1}^{i}p_k$$
$$+\frac{1}{N-1}\sum_{i=j+1}^{N}(i-j)p_i-\frac{1}{N-1}\sum_{i=j}^{N-1}\sum_{k=i+1}^{N}p_k$$
$$=\frac{1}{N-1}\sum_{i=1}^{j-1}\left[\left(\sum_{k=1}^{i}p_k\right)^2-\sum_{k=1}^{i}p_k\right]+\frac{1}{N-1}\sum_{i=j}^{N-1}\left[\left(\sum_{k=i+1}^{N}p_k\right)^2-\sum_{k=i+1}^{N}p_k\right]$$
$$+\frac{1}{N-1}\sum_{i=1}^{j-1}(j-i)p_i+\frac{1}{N-1}\sum_{i=j+1}^{N}(i-j)p_i.$$

Since we are dealing with probabilities we can rewrite the sums again:

$$\frac{1}{N-1}\sum_{i=1}^{j-1}\left[\left(\sum_{k=1}^{i}p_k\right)^2-\sum_{k=1}^{i}p_k\right]+\frac{1}{N-1}\sum_{i=j}^{N-1}\left[\left(\sum_{k=i+1}^{N}p_k\right)^2-\sum_{k=i+1}^{N}p_k\right]$$
$$=\frac{1}{N-1}\sum_{i=1}^{j-1}\left[\frac{1}{2}\left(\sum_{k=1}^{i}p_k\right)^2+\frac{1}{2}\left(1-\sum_{k=1}^{i}p_k\right)^2-\frac{1}{2}\right]+\frac{1}{N-1}\sum_{i=j}^{N-1}\left[\frac{1}{2}\left(\sum_{k=i+1}^{N}p_k\right)^2+\frac{1}{2}\left(1-\sum_{k=i+1}^{N}p_k\right)^2-\frac{1}{2}\right].$$

So, we get

$$S_j = -\frac{1}{2} + \frac{1}{2(N-1)} \sum_{i=1}^{j-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(1 - \sum_{k=1}^{i} p_k \right)^2 \right] + \frac{1}{2(N-1)} \sum_{i=j}^{N-1} \left[\left(\sum_{k=i+1}^{N} p_k \right)^2 + \left(1 - \sum_{k=i+1}^{N} p_k \right)^2 \right]$$
$$+ \frac{1}{N-1} \sum_{i=1}^{N} |i-j| p_i$$
$$= -\frac{1}{2} + \frac{1}{2(N-1)} \sum_{i=1}^{j-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(\sum_{k=i+1}^{N} p_k \right)^2 \right] + \frac{1}{2(N-1)} \sum_{i=j}^{N-1} \left[\left(\sum_{k=i+1}^{N} p_k \right)^2 + \left(\sum_{k=1}^{i} p_k \right)^2 \right]$$
$$+ \frac{1}{N-1} \sum_{i=1}^{N} |i-j| p_i$$
$$= -\frac{1}{2} + \frac{1}{2(N-1)} \sum_{i=1}^{N-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(\sum_{k=i+1}^{N} p_k \right)^2 \right] + \frac{1}{N-1} \sum_{i=1}^{N} |i-j| p_i.$$

We want to minimize the expected score. Thus, we write

$$\mathbb{E}(S) = \sum_{j=1}^{N} q_j S_j$$
$$= \sum_{j=1}^{N} q_j \left\{ -\frac{1}{2} + \frac{1}{2(N-1)} \sum_{i=1}^{N-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(\sum_{k=i+1}^{N} p_k \right)^2 \right] + \frac{1}{N-1} \sum_{i=1}^{N} |i-j| p_i \right\}$$
$$= -\frac{1}{2} + \frac{1}{2(N-1)} \sum_{i=1}^{N-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(\sum_{k=i+1}^{N} p_k \right)^2 \right] + \sum_{j=1}^{N} q_j \frac{1}{N-1} \sum_{i=1}^{N} |i-j| p_i.$$

Finally, we can also rewrite the sum, looking at all pairs

$$\sum_{i=1}^{N-1} \left[\left(\sum_{k=1}^{i} p_k \right)^2 + \left(\sum_{k=i+1}^{N} p_k \right)^2 \right]$$
$$= \sum_{i=1}^{N} \sum_{k=1}^{N} \left((N-1) - |i-k| \right) p_i p_k.$$

So we obtain the expression

$$\mathbb{E}(S) = -\frac{1}{2} + \frac{1}{N-1} \sum_{i=1}^{N} \sum_{k=1}^{N} \left((N-1) - |i-k| \right) p_i p_k + \frac{1}{2(N-1)} \sum_{j=1}^{N} q_j \sum_{i=1}^{N} |i-j| p_i.$$

We want to minimize the score. Directly taking the derivative and setting it to zero will not work here, so the Lagrange multiplier is used. To do this the constraint function $G(p_1, \dots, p_N) = \sum_{i=1^N} p_i - 1$ is introduced. It is clear that for any viable set of probabilities we must have $G = 0$. The function to minimize will be

$$H(p_1, \dots, p_N) = \mathbb{E}(S) - \lambda G(p_1, \dots, p_N)$$

We take the derivative with respect to p_n :

$$\begin{aligned}\frac{\partial H}{\partial p_n} &= \frac{1}{2(N-1)}\frac{\partial}{\partial p_n}\left(\sum_{i=1}^{N}\sum_{k=1}^{N}\left((N-1)-|i-k|\right)p_i p_k\right)+\frac{1}{N-1}\frac{\partial}{\partial p_n}\left(\sum_{j=1}^{N}q_j\sum_{i=1}^{N}|i-j|p_i\right)-\lambda \\ &= \frac{1}{N-1}\sum_{i=1}^{N}\left((N-1)-|n-i|\right)p_i+\frac{1}{N-1}\sum_{j=1}^{N}q_j|n-j|-\lambda \\ &= \frac{1}{N-1}\sum_{i=1}^{N}(N-1)p_i-\frac{1}{N-1}\sum_{i=1}^{N}|n-i|p_i+\frac{1}{N-1}\sum_{i=1}^{N}|n-i|q_i-\lambda \\ &= -\frac{1}{N-1}\sum_{i=1}^{N}|n-i|(p_i-q_i)+1-\lambda \qquad \text{for } \; n=\{1,2,\dots,N\}.\end{aligned}$$

Since we are minimizing we set this equal to zero and obtain:

$$-\frac{1}{N-1}\sum_{i=1}^{N}|n-i|(p_i-q_i)+1-\lambda=0 \qquad \text{for } \; n=\{1,2,\dots,N\}.$$

This means that we have the equality

$$\sum_{i=1}^{N}|n-i|(p_i-q_i)=(1-\lambda)(N-1).$$

Now let $r_i=p_i-q_i$ and $C=(1-\lambda)(N-1)$ and clearly we must have that for $n\in\{1,2,\dots,N\}$

$$\sum_{i=1}^{N}|n-i|r_i=C.$$

We now prove that the above equation is satisfied if, and only if, $r_i=0$ for all $i\in\{1,2,\dots,N\}$. To do this we take $n_1\in\{1,2,\dots,N-1\}$ and $n_2=n_1+1$. From the previous equation we must have that

$$\sum_{i=1}^{N}|n_1-i|r_i=C=\sum_{i=1}^{N}|n_2-i|r_i=\sum_{i=1}^{N}|n_1+1-i|r_i.$$

Rewriting the sums:

$$\sum_{i=1}^{N}|n_1-i|r_i=\sum_{i=1}^{n_1}(n_1-i)r_i+\sum_{i=n_1+1}^{N}(i-n_1)r_i,$$

and

$$\begin{aligned}\sum_{i=1}^{N}|n_2-i|r_i &= \sum_{i=1}^{N}|n_1+1-i|r_i \\ &= \sum_{i=1}^{n_1}(n_1-i+1)r_i+\sum_{i=n_1+1}^{N}(i-n_1-1)r_i.\end{aligned}$$

Taking the difference between these sums yields the equality

$$\sum_{i=1}^{n_1}r_i=\sum_{i=n_1+1}^{N}r_i.$$

Now again using the property that $\sum_{i=1}^{N} r_i = \sum_{i=1}^{N} p_i - q_i = 1 - 1 = 0$ we get

$$\sum_{i=1}^{N} r_i = 2 \sum_{i=1}^{n_1} r_i = 0$$

for all $n_1 \in \{1, 2, \ldots, N-1\}$ thus we must have that $r_i = 0$ for all i. We can conclude that the expected score is minimized if and only if the forecast is unbiased. Note that we know that this is the minimum and not the maximum by noting that the second derivative with respect to p_n is positive. Therefore we have now shown that the Ranked Probability Score is a strictly proper scoring rule.

Note that a continuous equivalent of the RPS exists and is actually the limit of the RPS when taking the limit of the bin size to 0.
Another attractive property of the RPS is that we can decompose the score in three parts which all have an interpretation. This is comparable to the MSE. We can decompose the MSE into the sum of the squared bias and the variance. We can rewrite the RPS as a sum of Brier scores [28]. The Brier score is a scoring rule for binary events and is decomposable into the uncertainty, the resolution and the reliability. Since we are talking about sums of these parts we can simply write the sum of the uncertainties as the uncertainty of the RPS to achieve an interpretable result. This is not done in this thesis since this is not within the scope of the research.
The original goal was to compare the results of the cross-entropy loss and the RPS. However, when experimenting it seemed that the RPS as a loss function did not perform well. After thorough checking of the implementation, it became apparent that even for simple cases it did not perform well. There were several possible explanations for this. The first explanation is that taking a sample and taking the gradients from this sample leads to biased gradients. This is a known problem for the Wasserstein divergence, as described in [29] and it leads to problems in the optimization routine due to steps in the wrong direction or size with the gradients. This is however not the case for the RPS as it is a discrete version of the Cramèr distance. Another possibility is that the objective function for the minimization becomes too complex for the optimization algorithm to achieve good results on. A final possibility is that while technically the RPS is a strictly proper scoring rule, in practice there are many local minima which have scoring values similar to the true minimum and therefore it is very hard to find the true minimum.

3.3. Parameter optimization

In order to achieve a good result with the network the parameters are trained to minimize the loss function, which has the goal of achieving the best outcome of the network. Training the network is done by updating its parameters.
The simplest way of training parameters is by considering the gradients of the loss function with respect to the parameters. This is called backpropagation.
Almost all methods that are used in practice when training a neural network are based upon computing the gradients for the weights with backpropagation. With these gradients, the parameters such as weights and biases are updated by going into the direction of that gradient multiplied by a factor, which is called the learning rate. Note that all methods here are described for a single input sample and output sample, but in practice we use more than one input sample. To account for this we simply take the loss (usually the sum or average) of the multiple inputs and compute the gradient of this with regard to the parameters. An algorithmic overview is given in 2.

Algorithm 2 Backpropagation

1: **procedure** Training with learning rate η
2: Let θ_t be the (possibly multi-dimensional) parameter values at training iteration t. Furthermore let f be the cost function for a given input and desired output. This cost depends on θ_t.
3: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
4: $\theta_t \leftarrow \theta_{t-1} - \eta g_t$
5: $t \leftarrow t + 1$

This method is simple and quite effective in simple cases. When the dimensions get large and the sign

of gradients switches a lot this is less effective. A solution to the switching of gradients and small steps is to use momentum.

The momentum method was introduced by Polyak in 1964 [30]. The idea is that this speeds up backprogagation by gradient descent by taking a momentum or velocity in a certain direction. Note

Algorithm 3 Momentum gradient descent

ht

1: **procedure** Momentum
2: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
3: $m_t \leftarrow \mu m_{t-1} + g_t$
4: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$
5: $t \leftarrow t + 1$

the momentum parameter μ.

A more sophisticated version of momentum was introduced by Nesterov. It was shown by [31] that this is useful in a machine learning setting. It has theoretically superior properties in non-random optimization problems but performs better than backpropagation and regular momentum in practice for stochastic cases.

Both momentum methods compute a velocity by applying a correction to the previous velocity vector based on a gradient. Regular momentum does this by computing the gradient update from the current value of θ while Nesterov momentum first produces an estimate for θ_t and computes the gradient from this to correct this estimate.

Algorithm 4 Nesterov momentum gradient descent

1: **procedure** Nesterov Momentum
2: $g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1} - \eta\mu m_{t-1})$
3: $m_t \leftarrow \mu m_{t-1} + g_t$
4: $\theta_t \leftarrow \theta_{t-1} - \eta m_t$
5: $t \leftarrow t + 1$

So while the regular momentum updates the parameters by moving in the direction of the momentum and taking the gradient of the reached point, the Nesterov method makes a move in the direction of the momentum (since we are going in that direction regardless) and computes a correction from that point.

There is a different method called Adam [32]. This method uses both momentum and adaptive step sizes. It computes the first and second moment and corrects for variance. Whereas the momentum methods discussed before work with a decaying sum of the gradients, the Adam method estimates the exponentially moving average of the gradient and the squared gradient. The squared gradient is used as in RMSProp [33] to adapt the learning rate for a specific parameter based on the size of the previous gradients. Furthermore it incorporates bias correction to ensure that the intial steps are not biased towards the starting guess 0 for the first and second moment estimates. The exponentially moving average of the gradient and squared gradients both have a momentum parameter. As before we have μ, but we now also have the ν parameter for the squared gradients.

So, Adam combines regular momentum with adaptive learning rates based on past gradient sizes. Its name Adam is derived from adaptive moment estimation, which is essentially what it does by combining the previously mentioned components. The algorithm is given below.

Since Adam uses regular momentum it is an obvious attempt to improve it by using Nesterov momentum. The algorithm stays almost the same, the only change is that we use updated momentum for the computation of the next parameter. So we replace $\hat{m}_t$ with $(1-\mu_t)\frac{g_t}{1-\prod_{i=1}^{t}\mu_i} + \mu_{t+1}\hat{m}_t$ to obtain Nadam.

This method was introduced in [34] and similar to the regular and Nesterov momentum, the Nadam outperforms the Adam method in practice.

Algorithm 5 Adam optimizer

procedure Adam given a batch. Note that t is the amount of training steps already taken.
$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$.
$\hat{m}_t \leftarrow \frac{m_t}{1-\prod_{i=1}^{t+1} \mu_i}$.
$n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
$\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$
$t \leftarrow t+1$

Algorithm 6 Nadam optimizer

procedure Nesterov Momentum with Adam given a batch. Note that t is the amount of training steps already taken.
$g_t \leftarrow \nabla_{\theta_{t-1}} f(\theta_{t-1})$
$\hat{g}_t \leftarrow \frac{g_t}{1-\prod_{i=1}^{t} \mu_i}$
$m_t \leftarrow \mu m_{t-1} + (1-\mu) g_t$.
$\hat{m}_t \leftarrow \frac{m_t}{1-\prod_{i=1}^{t+1} \mu_i}$.
$n_t \leftarrow \nu n_{t-1} + (1-\nu) g_t^2$
$\hat{n}_t \leftarrow \frac{n_t}{1-\nu^t}$
$\bar{m}_t \leftarrow (1-\mu_t)\hat{g}_t + \mu_{t+1}\hat{m}_t$
$\theta_t \leftarrow \theta_{t-1} - \eta \frac{\bar{m}_t}{\sqrt{\hat{n}_t}+\epsilon}$
$t \leftarrow t+1$

3.4. Gibbs sampling and multivariate generation

To generate multivariate scenarios we will use the technique known as Gibbs sampling. In essence it comes down to sampling from several conditional distributions instead of a multivariate one which give the same distribution for the variables in the end. This will now be shown.

Suppose we have a 2-dimensional time series $\{Y_t\}$ and $\{Z_t\}$. Now we want to generate a realization Y_{t+1}, Z_{t+1}. For the reasons explained in this thesis we will only focus on whether the variable is in a certain bin rather than the actual value. Suppose we have bins $y_0, \ldots, y_N$ and $z_0, \ldots, z_M$. Note that the model uses that the time series is autoregressive.
First the samples will be generated using inverse transform sampling. If we have a random variable X with Cumulative Distribution Function (CDF) F_X which is invertible and a standard uniformly distributed variable on $[0,1]$, it can be seen that $F_X^{-1}(U)$ is distributed as F_X. So generating a sample of X is done in the following steps:

1. Generate a random number u from the standard uniform distribution in the interval $[0,1]$.
2. Compute the value x such that $F_X(x) = u$.
3. Take x to be the random number drawn from the distribution described by F_X.

Note that the correctness is shown by taking the procedure, applying F_X to both sides in the probability and using the property of the uniform distribution.

$$\begin{aligned}\mathbb{P}\left(F_X^{-1}(U) \leq x\right) &= \mathbb{P}\left(U \leq F_X(x)\right) \\ &= F_X(x)\end{aligned}$$

This will now be applied to the multivariate generation problem by combining it with Gibbs sampling. Let

$$\tilde{Y}_{t+1} = F^{-1}_{Y_{t+1}|Y_t,\ldots Y_{t-N},Z_t,\ldots,Z_{t-N}}(U)$$

be a realization for Y_{t+1}. Now it is assumed that Y_{t+1} is a categorical distribution so we are only interested in what bin our realization lies. Suppose $\tilde{Y}_{t+1} \in y_i$. Next, a realization for Z_{t+1} is generated by generating conditional on $Y_t, \dots Y_{t-N}, Z_t, \dots, Z_{t-N}$ and on $\tilde{Y}_{t+1}$. This realization will be $\tilde{Z}_{t+1}$ and lies in bin z_j. So what is the probability of this happening?

$$\begin{aligned}&\mathbb{P}\left(Y_{t+1} \in y_i | Y_t, \dots Y_{t-N}, Z_t, \dots, Z_{t-N}\right) \mathbb{P}\left(Z_{t+1} \in z_j | Y_t, \dots Y_{t-N}, Z_t, \dots, Z_{t-N}, Y_{t+1} \in y_i\right) \\ =&\mathbb{P}\left(Y_{t+1} \in y_i, Z_{t+1} \in z_j | Y_t, \dots Y_{t-N}, Z_t, \dots, Z_{t-N}\right)\end{aligned}$$

This is due to Bayes' theorem and it shows that sampling from the multivariate time series in this way is equivalent to taking a draw from the multivariate conditional distribution, which is more involved. Thus the dependence structure is kept intact. This procedure is now repeated by increasing t by 1. An algorithmic overview of this is shown in algorithm 7.

Algorithm 7 Generation

procedure Network workings from input $\mathbf{x} = (x_{t_1}, \dots x_{t_N})$ to output $\mathbf{y} = (y_1, \dots y_M)$.
    It is assumed that the neural network(s) is/are trained. It outputs a probability for being in each bin for the next realization, based on the time series history. This is represented in the function f_d. Its output is multidimensional and corresponds to the probability of X_d^t being in that specific bin. So the output is M dimensional where M is the number of bins.
    while $m \leq M$ **do**
        while $t \leq T$ **do**
            while $d \leq D$ **do**
                $u \leftarrow$Draw from random uniform $[0, 1]$ distribution.
                Input is D dimensions, last N timepoints and extra points: $x[t, 1], \dots x[t, d]$.
                Produce $f_d(x[t-N, 1], \dots, x[t-1, 1], x[t-N, 2], \dots, x[t-1, 2], \dots x[t-N, D]$
                Take cumulative values of the above.
                Interpolate this to find the value that corresponds to u. This is $x[t, d]$
                $d = d + 1$
            $t = t + 1$
            $d = 1$
        $m = m + 1$
        $t = 1$

In Algorithm 7 the in depth explanation of the workings of the neural network is not explained. This is done for clarity reasons. Furthermore, the generation approach only depends on the outputted distribution. If we have another procedure that outputs probabilities in the same way and takes the same input the generation approach is identical.

The neural network is chosen to model the time series and the multivariate generation procedure has been described in general. However, the multivariate modelling approach with the neural network has not been presented. In the simplest sense the multivariate approach is to condition on the other series to take the possible dependence into account and these series will thus be added as input. Note that this means that a network has to be configured and trained for every dimension conditional on the others, together with an ordering of the conditioning as just described. However this makes our one-dimensional time series problem more complex for the neural network so the architecture has to be revised. This is done by augmenting the first layer.
This approach is similar to the ones in [7, 21]. Suppose we have two time series $x = (X_t)_{t=1}^N$ and $y = (Y_t)_{t=1}^N$. The conditioning is now done by computing the activation function f in the first layer:

$$f(w_h^1 *_d x + b) + f(v_h^1 *_d y + b)$$

for the filters $h = 1, \dots, M_1$. The idea is that if conditioning or part of the extra input does not give more information, this will be ignored in the end due to the optimization. The skip connections and the original convolution should have the same dimension, so if more than one filter is used, a 1×1 convolution is used.
IMAGE OF DIFFERENT ARCHITECTURE WITH CONDITIONING

3.5. Distribution goodness-of-fit testing

In order to test if a model can be used to study a given data set, it is wise to check if this data satisfies the assumptions of the model. One can argue that one of the most important assumptions to test is whether the data is really distributed as the model specifies. This can be verified by conducting a goodness of fit test with null- and alternative hypothesis given by:

- H_0 : The sample was generated by the assumed distribution F_0.
- H_1 : The sample was not generated by the assumed distribution.

When testing a hypothesis we either reject the null hypothesis or we do not reject it. Not rejecting the null hypothesis is not the same as accepting the null hypothesis, there is merely a lack of evidence to reject it.
When testing the hypothesis there are two types of errors that can be made:

- Type I error: Rejecting the null hypothesis while the null hypothesis is true.
- Type II error: Not rejecting the null hypothesis while the null hypothesis is false.

Note that there is a link between these two effects. If we were to always accept the null hypothesis the Type I errors would never happen, but our statistical test would not be useful, since the Type II error would be large.

In this section the methodology is presented on measuring the goodness of fit of the predicted distribution against the actual distribution. This is an essential piece of the picture, since a bad fit is sure to give bad results. In testing the goodness of fit of distributions there are several approaches. There are tests available for certain types of distributions: e.g. whether a random variable is normally or uniformly distributed. There are also tests available against more general families of distributions, these are usually based on the Empirical Cumulative Distribution Function (ECDF). These tests assume a random variable that is i.i.d. which is not necessarily the case in this thesis. It is however vital to check whether the approach taken in this thesis works for i.i.d. variables before moving on to more involved series. First distribution specific tests will be described, followed by distribution-free tests and finally a novel approach for non-i.i.d variables.

3.5.1. Distribution goodness-of-fit testing for known distributions

Hypothesis testing is done by choosing a null hypothesis and an alternative hypothesis. These hypotheses are compared by computing how likely the observation is given that the null hypothesis is true. When the probability of this event or a more extreme realization happening is very low, the null hypothesis is rejected. Before the test takes place, a threshold is chosen at which certainty level, or significance level, the null hypothesis will be rejected. Often this is chosen at 5%.

Jarque-Bera test

The Jarque-Bera test [35] is a test which takes a random sample $x_1, \dots, x_N$ and uses the skewness and kurtosis of the sample to see if this matches a normal distribution. The test statistic is based on the skewness S and kurtosis C of the sample:

$$S = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^3}{\left(\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2\right)^{\frac{3}{2}}}$$

$$C = \frac{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^4}{\left(\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2\right)^{2}}$$

When the sample is drawn from a normal distribution it can be shown that the test statistic JB asymptotically converges to a chi-squared distribution with two degrees of freedom. The statistic can now be used to test the hypothesis that the data is generated by a normal distribution. The statistic is computed as:

$$JB = \frac{S^2}{c_1} + \frac{(C - c_2)^2}{c_3}$$

where the c_1, c_2 and c_3 are correction factors for small sample size. In this thesis we do not consider such small sample sizes and can neglect the factors to constants and get the statistic

$$JB = \frac{N}{6}\left(S^2 + \frac{1}{4}(C - 3)^2\right).$$

Essentially this tests jointly whether the sample data has skewness 0 and excess kurtosis of 0 as does the normal distribution.

Anderson-Darling test

The Anderson-Darling test is a statistical test to assess whether a given sample was drawn from a given, specified distribution. One of the properties of this test statistic is that its critical values are distribution free and no parameters have to be estimated when computing the statistic. For the normal distribution this means that it assesses whether or not the empirical distribution function is similar enough to the hypothesized distribution without estimating the mean or standard deviation. The main idea is that when using the CDF of the random variable the distribution follows a uniform distribution on $[0, 1]$. The test statistic A^2 is usually given in terms of the empirical and hypothesized distribution:

$$A^2 = n\int_{-\infty}^{\infty} \frac{(F_n(x) - F_0(x))^2}{F(x)(1 - F(x))}.$$

The idea is based on a property of continuous distributions. When the data comes from the hypothesized distribution, the CDF of the data follows a uniform distribution. The Anderson-Darling test assesses whether or not this is true by ordering the data as described in [36]. The the data is ordered $\{Y_1 < \cdots < Y_N\}$ to compute

$$S = \sum_{i=1}^{n} \frac{2i - 1}{n}\left(\log \Phi(Y_i) + \log\left(1 - \Phi(Y_{n+1-i})\right)\right),$$

where Φ is the CDF under the null hypothesis.
Then the test statistic is

$$A^2 = -n - S.$$

The critical values for this depend on the CDF Φ. For the normal distribution at the 95% confidence level this becomes 0.787.

3.5.2. Goodness of fit when dynamics unknown

To be able to assess the performance of this methodology on a dataset where the underlying dynamics are not known the dataset is split in two parts: the training set and test set. The training set will be used to train the neural network on, whereas the test set is used to measure the performance and check if the network has learned to recognize patterns and perform on data it has not yet seen instead of overfitting on the training set.

An important problem in this thesis is how to assess the goodness of a probabilistic forecast when only a single realization is available. If the underlying distribution is known or the distribution is the same at every point in time the problem simplifies but is still non-trivial. When this is not the case the realizations can be aggregated and based on the assumptions, there are still ways to quantify the performance of the probabilistic forecasts.

Here, a new method to assess a probabilistic forecast is introduced. Suppose for each realization we want to know where every 10% percentile is. So we want to find the z such that $F_{X_t}(z) = 0.1$. We not only do this for 0.1 but also for $0.2, 0.3, \ldots, 0.9$. This might span several of the bins we defined, and might fall in the middle of a bin. For each realization we count in which of these areas it lies. If we look at all the 10 different areas we would expect them to all get approximately 10% of the realizations.

Therefore we can use a statistical test whether or not this is distributed uniformly over these bins.
Note that the values of z can be different for every single time step, since the shape of the distribution is a function of past values of the time series. For the uniform distribution there are several tests available to assess whether or not a sample comes from the uniform distribution. In this thesis the Pearson χ^2 test will be used.

Suppose that $Z \sim N(0, 1)$ is a standard normal random variable. Let us define a new random variable $Y = Z^2$. Y is a chi-squared distributed random variable with one degree of freedom χ_1^2. If the amount of samples increases, so do the degrees of freedom. The χ^2 distribution is most often used for hypothesis testing.

The primary reason that the chi-squared distribution is used extensively in hypothesis testing is its relationship to the normal distribution. Most hypothesis tests use a test statistic. For these hypothesis tests, by the Central Limit Theorem, the sampling distribution of the test statistic approaches the normal distribution. Because the test statistic is asymptotically normally distributed the distribution used for hypothesis testing may be approximated by a normal distribution. Testing hypotheses using a normal distribution is relatively easy. Once again, the simplest chi-squared distribution is the square of a standard normal distribution. So wherever a normal distribution could be used for a hypothesis test, a chi-squared distribution could also be used.

Another reason that chi-squared tests are widely used is the fact that they belong to the class of Likelihood Ratio Tests. These have the attractive property that they provide the highest power to reject the null hypotheis (by the Neyman-Pearson lemma) asymptotically. This means that a large sample size is required.
The method presented below should be used with care, there was no method found in literature that uses the same approach on random variables that are non-i.i.d. distributed. This method is about hypothesis testing of the predictions over the entire dataset and thus not about the soundness of the forecast on a single time step basis. Since the distribution is not known for a single observation, a higher level approach must be taken to make an informed and useful judgment.

The test statistic is

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed frequency of the variable in the cell and E_i the expected frequency. In our example above $n = 10$ and $E_i = \frac{N}{10}$. Now the null hypothesis can be tested whether or not the frequencies

correspond to the uniform distribution. We test

H0: The observed frequency for all bins is distributed as a categorical distribution with $p_i = \frac{1}{n}$ for all i.

This asymptotically becomes a chi-squared distribution with $n-1$ degrees of freedom. Note that this does not necessarily prove that the predicted distribution is correct, but if this is done at several quantile levels it should give a good indication whether or not the method is sensible overall. A counterexample when the forecast is not perfect but the chi-squared test does not help is now given.
For example suppose we have a distribution with fixed $a, b \in \mathbb{R}$ and $b > a$ and U denoting the uniform distribution such that

$$X_t = \begin{cases} U([a, \frac{b}{2}]) & \text{if } t \text{ odd} \\ U([\frac{b}{2}, b]) & \text{if } t \text{ even} \end{cases}$$

Suppose that the forecast $\hat{Y}_t = U[a, b]$ is made at every time step. Clearly this forecast is not very good since it assigns half of the probability to an event that will certainly not happen at that time step. With our uniform forecast $\hat{Y}_t$ we expect to find as many observations in each quantile (ignoring the possibility of one more even or odd realizations) at every time step t. However, it is clear that this prediction is not that good at all. The methodology described above basically checks whether the realizations are biased towards a certain side of the distribution. This is worthwhile to investigate but it is important to know that the outcome does not imply that the predicted distribution is the actual underlying distribution.
The null-hypothesis cannot be rejected. This shows that the chi-squared test described in this section tests whether the aggregated probabilistic forecasts are skewed towards a certain side. The chi-squared approach does not take the historical dependence into account. When looking at the approach this might appear to be a problem. However, during the training/optimization procedure assures that these very uninformative predictions get a higher error than better predictions and thus minimizing the cross-entropy loss in practice prevents this problem from happening since its expected loss is minimized for the true underlying distribution.
A combination of the informativeness of the forecast and this test could be a promising way to evaluate probabilistic forecasts on series where the dynamics are unknown.

A practical problem in using the binned approach to estimate the distribution is that when a certain bin has a large probability it can span several quantiles. For example, if the prediction for events A, B, C, D and E is $[0.1, 0.1, 0.6, 0.1, 0.1]$ and we are interested in estimating the quantiles with the above method where we take the percentile by halves we see that this falls for two quantiles we are interested in. This poses a problem for the counting since it is unclear in which quantile the realization lies and how it should be counted.

This method can work on problems where the underlying distribution is not known but has no other literary background and has some flaws. The flaws are that this method determines the uniformness of the realizations with regard to the prediction as aggregated, thus allowing a switching approach as with the uniform counterexample. In practice this should not be a problem since the optimization routine ensures that not such a broad forecast is made. Another issue is predictions in bins with a high probability. There will be several of the splits made in that bin, which makes it a challenge to be solved. This method can however provide an indication of the spread on a dataset where we do not know the underlying dynamics, which is better then no method at all.

3.6. Comparison Feedforward Network

In this section we compare the previous results of the WaveNet against a Feedforward Neural Network. This is to show the difference in the architecture of the network on the performance.
When going to random variables the difference becomes very clear. Looking at the normal distribution as in Figure 3.6.1 it is easy to see that the Feedforward Network performs worse. We train on the normal distribution.
What is also interesting to see is that the Feedforward network suffers from overfitting whereas the WaveNet based model does not. This can be seen in Figure 3.6.2.

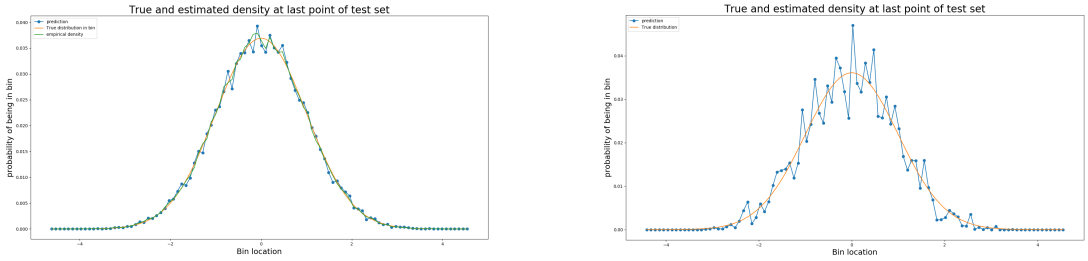

Figure 3.6.1: Density forecast for the normal distribution for WaveNet (left) and the Feedforward Neural Network (right).

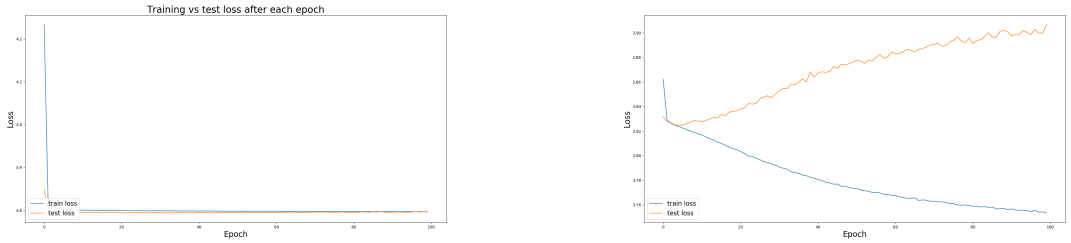

Figure 3.6.2: The loss function value on the Y-axis per training epoch on the X-axisfor both training (in blue) and test set (in orange) for WaveNet (left) and a FeedForward Neural Network (right).

4

Numerical Experiments Setup

In this chapter the setup for testing how the methodology in this thesis compares against different classical methods is described as well as the performance with respect to different parameter choices. A motivation is given to understand why each of the tests and datasets used to asses this performance is useful, how they work and what insights we can gain from each dataset or test. The actual numerical results of these tests will be presented afterwards in chapter 5.

When predicting a distribution it is hard to assess how well this estimates the actual underlying distribution, since only a single realization is known for an entire distribution. There are several cases that make this generally hard problem easier. If the underlying distribution is known beforehand there are several statistical tests we can perform. Likewise, if the distribution at each time step is independent and identically distributed, naive estimators can perform very well. However, if the underlying distribution is not known it can be much harder to assess the performance. Since it is hard to assess the goodness of fit of the distribution in the more complex cases, it is logical to first look at simpler cases and build on these to gain insight into the more involved problem. Therefore the approach of this thesis is to first generate artificial time series for which the underlying dynamics are known, which ensures that we are able to assess the performance of the methodology.

The simpler patterns are subdivided into deterministic time series in which the ability of learning non-linear dynamics as well as time dependence is tested. These will be further subdivided into deterministic series which show a certain pattern over time but no randomness and stochastic series where no time pattern is shown but randomness is prevalent.

Later on, these two test sets are combined to assess the performance of the combination of those series. This means that there can be a time effect as well as randomness and possible non-linearity. Finally, conditional forecasting is considered which is used for multivariate forecasting of time series.
In the following sections more detail will be provided on which series are used in their respective category, what are properties of these series and which metrics are used to measure the performance.

4.1. Deterministic time series

The deterministic series are used to assess whether the WaveNet approach can pick up patterns based on the time series history. Furthermore, since we are dealing with a deterministic series, there is no randomness involved and the realization on the next time step should be perfectly predictable. Therefore, we know the value of the event beforehand with probability 1 and expect to see a single peaked value at one of the bins for the probability, so all bins having almost zero probability except for one bin with a probability of almost one.

If the network is not able to predict the next time step well this should be seen in either one of the assessment metrics or the spread of the distribution over several bins. The cross-entropy loss, which is used for training the neural network, gives a better score to more peaked distributions thus we should

be able to see this. The ranked probability score on the other hand does not only look at which bin the realization was in and with what probability this was predicted, it also takes into account how close it was in predicting. For a continuous series, such as the sine wave, this is an important aspect as well.

The one step ahead forecast based on the true underlying series will be investigated as well as the iterative mean forecast. This entails taking the mean of the predicted distribution and plugging this in as part of the input for the next time step.

The error metric used to assess the performance of the neural network prediction on the paths of these series is the Mean Squared Error (MSE) between the mean of the predicted distribution and the realization.

These metrics will be evaluated as well for different settings for the neural network:

- the amount of bins used for the forecasted distribution
- the size of the dataset
- the amount of historical values used as input for forecasting: the lookback range
- kernel size of the filter
- amount of filters used
- training epochs and batch size

Evaluating the metrics for all of these at once would not give a clear overview so the changes are taken from a baseline setting and the adjusted settings are evaluated on a case by case basis.

For completeness, the naive estimator of taking the same value as the previous time step is also considered.

In this thesis two different deterministic series will be studied. The first series is a sine wave which only exhibits a seasonal component and is periodic. The sine wave has a period of 100. Since the series is clearly periodic the input of the historical values should be enough to predict the next time step. An interesting insight can be gained by looking at how many historical values are necessary to be able to predict well. In other words, how far does one need to look back to predict. A graphical representation of the sine wave can be seen in Figure 4.1.1

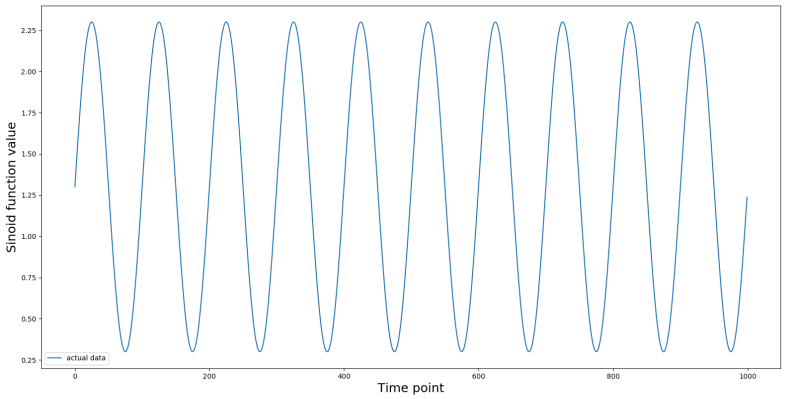

Figure 4.1.1: First 1000 time steps of the sine wave with a period of 100. On the x-axis the time is given and on the y-axis the value of the sinus.

The second series is the Lorenz system of differential equations. This is known to be a chaotic series so a small deviation from the starting value will have a large impact over time. The Lorenz system is a system of coupled differential equations in three dimensions, given by $(X(t), Y(t), Z(t))$, satisfying the following equations:

$$\begin{aligned}\frac{\partial X}{\partial t} &= \sigma\,(Y - X)\\ \frac{\partial Y}{\partial t} &= X\,(\rho - Z) - Y\\ \frac{\partial Y}{\partial t} &= XY - \rho Z\end{aligned}$$

The parameters that are usually associated with this system are used: $\sigma = 10, \rho = 28$ and $\beta = \frac{8}{3}$ with starting values $(X(0), Y(0), Z(0)) = (0, 1, 1.05)$. This series is interesting since it exhibits linear and non-linear relationships and it is important to assess whether the methodology can pick this up.

For this series it is also interesting to see if there is a difference between the short term and long term predictions. Since we are dealing with a chaotic series the expectation is that the predictions break down over time, but the setup of the network might have an impact on how long this takes and how well it models the dynamics.

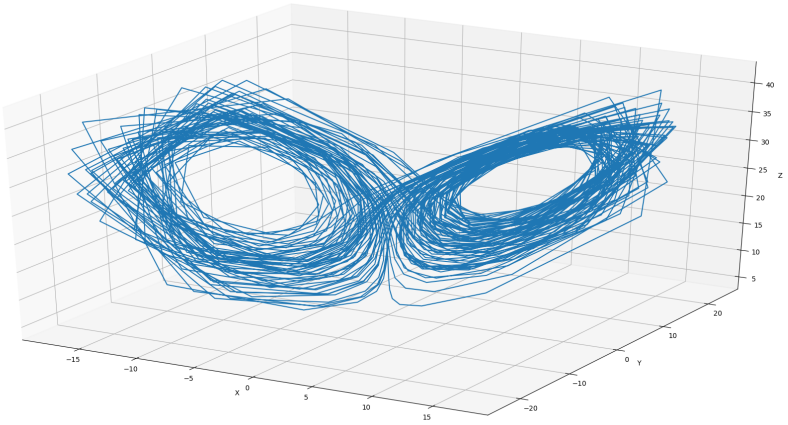

Figure 4.1.2: First 1000 time steps of the Lorenz system. This image shows the dynamics between the different coordinates.

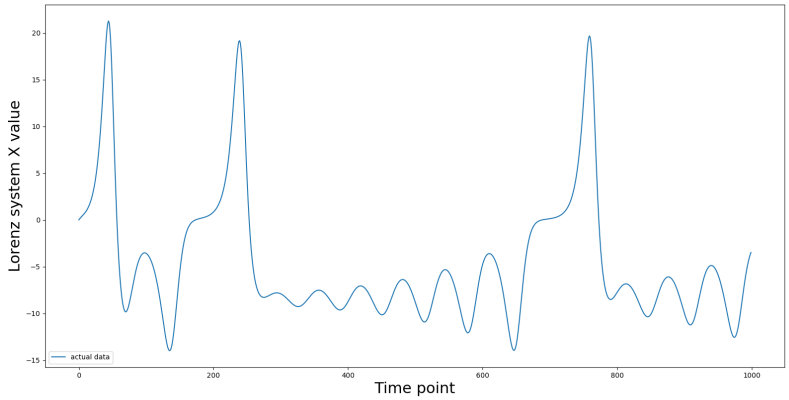

Figure 4.1.3: First 1000 time steps of the Lorenz system for the X coordinate. On the horizontal axis the time is shown and on the vertical axis the value of the X coordinate for the Lorenz system.

4.2. Stochastic time series

The main reason for using distributions in this methodology is clearly to deal with randomness. To assess how well the methodology handles randomness stochastic time series are generated. These series will be generated from several different distributions which will be explained in more detail below. For each distribution, there are several criteria that should be monitored to evaluate the soundness. A path will be generated, so iteratively drawing samples and using them as input, to see if the time series generation picks up non-existing patterns. Secondly, multiple paths are generated to determine whether or not the distribution is biased at each time step. Furthermore, the generated paths are used to perform statistical tests to determine whether or not we can confirm that the samples come from the actual underlying distribution with a specific confidence level. For i.i.d. samples such as the normal distribution, it is also interesting to check whether there is a bias in the historic part of the series.

The score used in the training, the cross-entropy loss, will also be reported as well as the accuracy and the Ranked Probability Score (RPS). The RPS does not only take the accuracy into account but also how close one is to the observation. It is a strictly proper scoring rule, so its expected score is minimized only for the actual distribution. The score is both applied to the realizations and the underlying distribution to show the difference here.

Furthermore, basic properties of the distribution are analyzed. The sample mean, variance, skewness and kurtosis are reported against the values for the true distribution. A comparison is made against the naive density estimator on the same bins. This means that if we have realizations $Y_1, \dots, Y_M$ on the training set and the bins $B_0, B_1, \dots, B_N$ the density estimator becomes:

$$\hat{f}(x) = \begin{cases} \frac{1}{M} \sum_{j=1}^{M} \mathbb{1}(Y_j \in (B_i, B_{i+1}]) & \text{if } x \in (B_i, B_{i+1}] \\ 0 & \text{elsewhere} \end{cases}.$$

Note that for a time series where the entries are independent and identically distributed we cannot expect to beat the naive estimator. Therefore it is a good benchmark to analyze how close we get.

The following four distributions will be used: Bernoulli distribution, uniform distribution, Gaussian distribution and the lognormal distribution. The Bernoulli distribution is very simple and can only take two values. The uniform and Gaussian distributions are often used in practice and should therefore be forecasted correctly. The lognormal distribution is not symmetric and has a heavy tail and is the exponential of a normally distributed variable. So, if we take the logarithm of a lognormal variable, we get a normal distributed random variable.

4.3. Mixture of deterministic and stochastic time series

Once the performance on the deterministic and stochastic series has been evaluated, it is important to consider what happens if we combine these. Standalone effects might be easier to be recognized and learned by neural networks, but a combination could be more of a challenge. Therefore, in this section the setup for a combination between deterministic and stochastic parts is explained. It should be stressed that performance on a combination of patterns over time and randomness is very important since almost all time series in the real world have a deterministic and noise component.

Varying the amount of noise (e.g. adding a normal distribution with different variances) will be used to investigate if the network is able to pick up a pattern over time even when there is randomness involved.

To do this the sine wave from before has a normally distributed random variable added to it.
The same tests as for the stochastic time series are considered, with the extension of the best estimate (mean) at every time step procedure for the deterministic series.

4.4. Conditional and multivariate time series

When considering multivariate time series, the problem gets another, extra, dimension. Next to the combination of deterministic and stochastic parts of time series, dependence between two series also plays a role. For each dimension a different neural network will be trained as explained in the Gibbs sampling section. This means that for every time step that is taken, a sample is drawn for the next time step for one dimension and this sample has an impact on the sampling of all the different dimensions on the same time step.

Measuring multivariate performance is possible but not the most straightforward approach. It is easier and more insightful to measure the results per dimension. A further benefit is that a comparison can be made by looking at the unconditional forecast of the series, as in the setup in the previous sections.

In this thesis the goal is to see how well we can learn correlation and non-linear dependence. For the correlation we use correlated normally distributed random variables. By varying the correlation it can be tested whether this has a significant impact in the ability to predict.

However, correlation is not the only aspect, it is also necessary to investigate effects over time and non-linear effects. These are both found in the Lorenz system as described before. Instead of forecasting one of the dimensions, we couple all three of them and can thus compare it to the one-dimensional method. An improvement is to be expected since more information is available.
The assessment metrics are the same as in the previous sections.

5

Numerical Results

Based on the setup explained in the previous chapter the results of these experiments are presented here and discussed. Furthermore the settings for the neural network that were used are given for reproducibility.
Note that the focus in the experiments is laid on correctly predicting the distribution for the next time step. This is due to several reasons. If we were to evaluate a generated scenario it is much harder to assess how likely such a scenario is. For distributions metrics are available to test statistical significance. Furthermore, when generating a scenario, if the distribution that is predicted is predicted perfectly the scenario generation is automatically applied correctly.

5.1. Settings to obtain the results

For the WaveNet we use 5 filters per layer, with 5 layers and a kernel size of 2. This means that we have stacked 5 layers where each time the dilation increases by a factor of 2. Every time $\frac{2}{3}$ of the data is the training set and $\frac{1}{3}$ is the testing set. Note that we do not use a validation set yet, since the hyperparameters are not being optimized. The amount of datapoints used are either $20,000$ with 75 total training epochs. The batch size is increased at $\frac{2}{3}$ as preliminary testing and research by [37] show that this improves the performance. The batch size is set to 150 and later on increased to 500. As he filter size is set to 2 and 5 dilated convolutional layers are stacked, the receptive field is $2^5 = 32$. Furthermore, the goal is to predict the density in a similar way to a histogram. This is done in 100 bins.
For all problems the Adam optimizer is used with learning rate 0.002, momentum for the average gradient 0.9 and momentum for the squared gradients 0.999.
When the neural network has been fit, based on the last points in the training set, scenarios are generated for the future. For every dataset 100 paths are generated each with 1000 steps ahead of those points in the training set.
For clarity in the results the error scores and meanings are given here:

$$\mathbf{p} = (p_1, p_2, \dots, p_N)$$

$$\text{CE} - \text{loss} : f(\mathbf{p}, x_j) = -\sum_{i=1}^{N} \delta_{ij} \log p_j$$

$$\text{RPS} : = \frac{1}{N-1} \sum_{i=1}^{N-1} \left(\sum_{k=1}^{i} p_k - \delta_{kj} \right)^2$$

MSE pathwise mean: 100 Paths $Y_{t+1}, Y_{t+2}, \dots, Y_{t+1000}$ are generated with random sampling. The mean of these 100 paths is taken every time step as an indication of where the true mean will lie. These

values are now used to compute the MSE with the realizations.
MSE distr. mean: One path $Y_{t+1}, Y_{t+2}, \dots, Y_{t+1000}$ is generated by taking the mean of the predicted distribution and plugging this into the input to predict the distribution of the next time step. These values are now used to compute the MSE with the realizations.
MSE distr. median: One path $Y_{t+1}, Y_{t+2}, \dots, Y_{t+1000}$ is generated by taking the median of the predicted distribution, so the z where $F_{Y_{t+h}}(z) = 0.5$ and plugging this into the input to predict the distribution of the next time step. These values are now used to compute the MSE with the realizations.

The Naive approach differs between the deterministic and stochastic case. In the deterministic case the previous value is used with perfect information, thus the outcomes are merely shifted a time step. In the stochastic case, the empirical density estimator is used by simply looking at how often a realization falls into a bin on the training set.

For each dataset the metrics are computed for different settings. The baseline approach has been described previously, the following parameter changes are made to test for robustness and influence of these parameters. Unless specified, the rest of the settings are the same as for the baseline approach.

- 7 layers LB*4 This means that two extra layers with dilated convolutions were added, yielding a receptive field, or look back (LB) in this case that is four times as large.
- 3 layers LB/4 This means that two fewer layers with dilated convolutions, yielding a receptive field, or look back (LB) in this case that is four times as small.
- 50 bins This means that the probability is being predicted in only 50 bins, half the amount of the baseline.
- 150 bins This means that the probability is being predicted in 150 bins, 50 more than the amount of the baseline.
- 40k datapoints This means that the dataset has 40,000 datapoints, double the baseline case.
- 10k datapoints This means that the dataset has 10,000 datapoints, half of the baseline case.
- Kernel 3, LB=27 This means that the dilated convolutions have a kernel/filter size of 3 and we use three layers of these to obtain a look back (LB) of 27.
- Kernel 4, LB=16 This means that the dilated convolutions have a kernel/filter size of 4 and we use two layers of these to obtain a look back (LB) of 16.
- Kernel 4, LB=16 This means that the dilated convolutions have a kernel/filter size of 4 and we use three layers of these to obtain a look back (LB) of 64.

Due to the nature of the probabilistic forecast it is hard to visualize useful information in a clear and comprehensible manner. Therefore most results are presented in tables.

5.2. Deterministic series

The first step is to analyze whether we can pick up non-random patterns. If the method works well it should give peaked values at the correct bin and 0 probability elsewhere. If the probability estimated is not 0 it could indicate that it is unable to correctly observe the pattern.
It is to be expected that predicting a distribution for this is less accurate than predicting an actual value since it is redundant to compute a distribution here. Simply said, there is more complexity in computing a distribution which is unnecessary for this task.
First we see if we can "forecast" a deterministic function. This is to see if the network is able to approximate an arbitrary non-linear function. We start off with a simple sinoid function:

$$y = 1.3 + \sin\left(\frac{\pi N x}{50(N+1)}\right).$$

This is chosen such that the period is 100 steps. In Figure 5.2.1 an example is given of the predicted probabilities for each bin. As can be seen the there is a single peak predicted which indicates the bin

with the next value for the sinoid. This image gives an idea of what the prediction probability vector looks like. It is clear that the network is able to predict the location, but not fully yet since there is not a peak with probability one.
For the deterministic series the naive method is repeating the value at the previous time step. Obviously this is not a great strategy for a sinoid series.

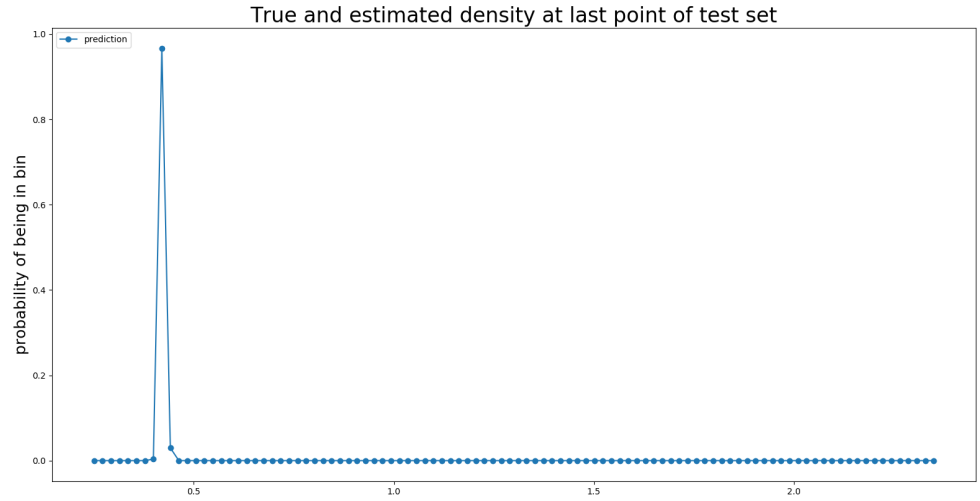

Figure 5.2.1: One step ahead probability prediction for each bin. On the x-axis we have the function value at the next time step and on the y-axis the corresponding probability forecast.

If we look several steps ahead it can be seen that the inference done leads to the result in Figure 5.2.2.

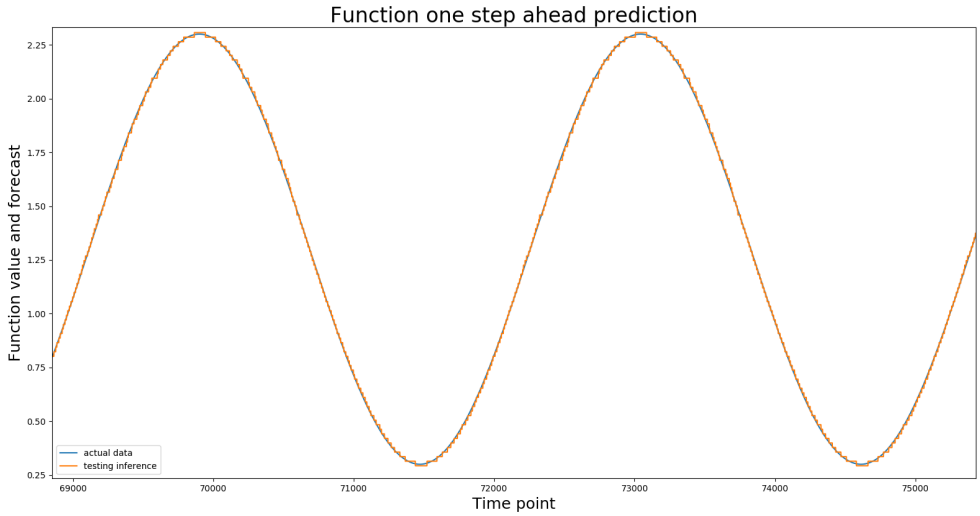

Figure 5.2.2: One step ahead sinus wave estimation.

In Table 5.2.1 we can see the different error measures on the test set. It can be seen that the performance is relatively stable to parameter changes. It can be seen that having a longer look back has a positive effect on the performance. This makes sense since more information about the series is available to predict upon. This also holds for having a bigger dataset, more examples are available thus the performance increases. The network outperforms the naive strategy for this approach.

	CE-loss	Naive CE loss	RPS ·E-5	Naive RPS · E-5
Baseline	0.6214	14.8320	4.867	28.0
7 layers LB*4	**0.3300**	14.8328	2.653	28.4
3 layers LB/4	1.2505	14.8269	9.2145	27.9
50 bins	0.400	**12.2573**	3.006	23.2
150 bins	0.9745	15.4750	7.477	29.2
40k datapoints	0.5693	14.8303	**2.248**	**14.0**
10k datapoints	1.0549	14.8354	16.04	56.3
Kernel 3, LB=27	1.1700	14.8305	8.801	28.0
Kernel 4, LB=16	1.5575	14.8253	10.84	27.9
Kernel 4, LB=64	0.6076	14.8257	4.664	28.1

Table 5.2.1: Numerical results for the losses on the test set for the sine wave

Variations in the Naive scores are due to the different look back distances which make sure that the set sizes are always slightly different, resulting in small differences.

Since the performance not only depends on the one-step ahead forecast with perfect information, we look into longer term performance. To do this paths are generated in several ways. First, we take a single path with at every time step the mean of the predicted density. This is compared with generating 100 paths and taking the mean of these paths at every time step. Finally, the most likely event is taken at each time step and then used to predict the value at the next time step. It should be noted that this approach is not the idea behind the use of this methodology but it is important to compare the performance against other methods.

	MSE naive	MSE Network	MSE pathwise mean	MSE distr. mean	MSE distr. median
Baseline	1.97 E-3	1.33 E-4	0.27860	0.5591	0.9486
7 layers LB*4	1.97 E-3	1.38 E-4	**0.00019**	**0.0106**	**0.0187**
3 layers LB/4	1.97 E-3	**1.28 E-4**	0.48150	1.0736	1.2902
50 bins	1.97 E-3	1.70 E-4	0.31350	0.4916	0.5834
150 bins	1.97 E-3	1.88 E-4	0.34070	0.7313	0.7007
40k datapoints	1.97 E-3	1.37 E-4	0.35780	0.5602	0.6089
10k datapoints	1.98 E-3	2.40 E-4	0.33110	0.3142	0.6006
Kernel 3, LB=27	1.97 E-3	2.08 E-4	0.26290	0.5735	0.8476
Kernel 4, LB=16	1.97 E-3	1.64 E-4	0.43170	0.8733	0.9920
Kernel 4, LB=64	1.97 E-3	1.47 E-4	0.02250	0.3658	0.2061

Table 5.2.2: Numerical results for the Mean Squared Error (MSE) for the naive approach and other approaches. The first two columns represent the score on the test set and are based on the perfect information from the previous timesteps. The other approaches are on a step by step basis and are thus expected to perform worse. The pathwise mean has a different size set than the other approaches and is based on the first 1000 points of the test set.

As can be seen from Table 5.2.2 the MSE of all approaches for the network to generate greatly improves with a longer lookback. This makes sense since the sinoid is periodic with a period of 100, thus this should give the best results. This can also be seen in the approaches for different kernel sizes. The lookback is a more important indicator than the kernel size for the quality of the prediction.

For time series models usually the goal is to minimize the Mean Squared Error and give a point forecast for the next timestep. This would often be the most likely value or the expected value. Both approaches are given in Table 5.2.2 and it is interesting to see that if we generate paths and take the average of these paths at every time step a lower MSE is obtained in almost all cases. This shows the value of the proposed methodology.

It can be seen that the performance increases if the amount of datapoints increases as well as when we have a larger receptive field. The MSE is more susceptible to parameter changes than the cross entropy loss and the RPS as reported in Table 5.2.1. This is to be expected since multiple steps ahead are forecasted instead of only one every time. This is a more difficult problem and thus the performance can be expected to vary more.

5.2.1. Lorenz curve

For the Lorenz curve the results are expected to be worse due to the chaotic nature of the series. Furthermore there is not enough information in only a single variable (the X variable is chosen) to accurately predict the series. Note that the cross-entropy loss and RPS for two different datasets cannot be compared effectively, therefore a comparison with the values to a problem such as the sinoid previously makes no sense for these scores. This will be done afterwards in terms of the MSE.
It can be seen in Table 5.2.3 that the performance on the cross-entropy loss and the RPS is quite robust as in the sinoid setting.

	CE-loss	Naive CE loss	RPS	Naive RPS
Baseline	0.902	8.549	7.15 E-5	1.62 E-4
7 layers LB*4	1.133	8.529	8.70 E-5	1.64 E-4
3 layers LB/4	1.089	8.547	8.29 E-5	1.61 E-4
50 bins	**0.336**	**5.051**	**2.39 E-5**	9.54 E-5
150 bins	1.198	10.488	8.91 E-5	1.98 E-4
40k datapoints	0.652	9.123	2.58 E-5	**8.60 E-5**
10k datapoints	1.517	9.833	2.05 E-4	3.73 E-4
Kernel 3, LB=27	1.083	8.550	8.49 E-5	1.61 E-4
Kernel 4, LB=16	1.357	8.552	1.00 E-4	1.61 E-4
Kernel 4, LB=64	0.993	8.546	7.90 E-5	1.62 E-4

Table 5.2.3: Numerical results for the losses on the test set for the Lorenz map

The difference in the MSE is much larger for different approaches. Note that the range of these variables is different from the sinoid by approximately a factor 20, so the MSE can be expected to be much larger than in the case of the sinoid and thus should not be compared. It is notable that the MSE hugely increases for the approach with seven layers. Since the system is chaotic a small influence can propagate to a huge difference over time and the longer look back distance is of no value. It can be seen that the approaches with a shorter receptive field perform better for the test set in the column "MSE Network". Surprisingly the approach with fewer datapoints performs better than the one with more datapoints. This might be due to more chaotic behaviour after a certain time.

	MSE naive	MSE Network	MSE pathwise mean	MSE distr. mean	MSE distr. median
Baseline	0.1719	**0.0331**	92.23	60.14	123.29
7 layers LB*4	0.1727	8.8407	80.97	69.73	107.01
3 layers LB/4	0.1715	0.0277	119.35	70.32	95.91
50 bins	0.1719	0.2646	106.71	67.33	147.76
150 bins	0.1719	0.0493	135.01	62.25	107.96
40k datapoints	0.1808	0.0582	146.15	63.50	120.54
10k datapoints	0.1879	0.8795	**85.71**	95.68	92.28
Kernel 3, LB=27	0.1718	0.0394	101.30	**61.32**	139.72
Kernel 4, LB=16	0.1716	0.0676	129.54	64.75	122.07
Kernel 4, LB=64	0.1724	0.0344	133.46	63.74	**81.92**

Table 5.2.4: Numerical results for the MSE on the test set for the Lorenz map

5.3. Randomness

Performance on deterministic series is necessary, but predicting probabilities is important if there is randomness. In this section several random variables are shown and predicted. Note that these variables are generated i.i.d. so the neural network cannot be expected to beat the naive kernel density estimator. This is included for comparison.
It is important to realize that the naive strategy in the randomness section is different from the deterministic case since we can develop a better naive approach than using the previous realization. If the time series is generated as i.i.d. random variables the best naive approach can be seen as the histogram

with the same amount of bins as the network. It is not expected to beat the performance of this, but it gives a good indication of how close the methodology of this thesis can get.

5.3.1. Normal distribution

In Table 5.3.1 the results of the numerical experiments can be found.

	CE-loss	Naive CE loss	RPS	Naive RPS	RPS (True)	Naive RPS (True)
Baseline	3.929	3.929	1.49 E-4	1.49 E-4	8.46 E-9	8.26 E-9
7 layers LB*4	3.929	3.929	1.51 E-4	1.51 E-4	9.78 E-9	8.63 E-9
3 layers LB/4	3.929	3.930	1.48 E-4	1.48 E-4	9.68 E-9	8.25 E-9
50 bins	3.180	3.179	1.45 E-4	1.45 E-4	13.5 E-9	10.5 E-9
150 bins	4.355	4.356	1.50 E-4	1.50 E-4	11.6 E-9	11.3 E-9
40k datapoints	3.932	3.932	7.42 E-5	7.42 E-5	2.99 E-9	2.66 E-9
10k datapoints	3.948	3.950	2.99 E-4	2.99 E-4	44.7 E-9	42.5 E-9
Kernel 3, LB=27	3.929	3.929	1.49 E-4	1.49 E-4	11.8 E-9	8.20 E-9
Kernel 4, LB=16	3.930	3.930	1.48 E-4	1.48 E-4	9.79 E-9	8.22 E-9
Kernel 4, LB=64	3.931	3.930	1.49 E-4	1.49 E-4	13.5 E-9	8.56 E-9

Table 5.3.1: Numerical results for the losses on the test set for the normal distribution.

As can be seen from Table 5.3.1, the reported losses and scores are quite robust to parameter changes. It makes sense that the cross-entropy loss increases for more bins since the probabilities get smaller the logarithm term becomes more negative resulting in a larger loss. When not looking at the different amount of bins it can be seen that the best result is achieved when there is more data available. This holds for both the network score as well as the naive density estimator approach. This is to be expected due to the law of large numbers. The scores for the naive approach and the network approach are very similar. This indicates that the network does a good job at forecasting the correct distribution. Furthermore the error scores do not wildly vary with different parameter settings which indicates that the used methodology is robust.
To measure this in another way, a step is taken to statistical hypothesis testing. Many different paths are generated to assess this. By choosing a confidence level of 95% we expect to reject the null hypothesis in only 5% of these cases. Generating 100 paths and looking at how many times we reject gives the following results for the Jarqu-Bera test and the Anderson-Darling test in Table 5.3.2.

Correlation	Jarque-bera	Anderson-Darling
Baseline	0.95	0.95
7 layers LB*4	0.93	0.95
3 layers LB/4	0.94	0.95
50 bins	0.95	0.96
150 bins	0.95	0.95
40k datapoints	0.97	0.96
10k datapoints	0.92	0.97
Kernel 3, LB=27	0.93	0.94
Kernel 4, LB=16	0.93	0.94
Kernel 4, LB=64	0.94	0.94

Table 5.3.2: Numerical results for the hypothesis testing for the normal distribution

It can be seen that the values are close to the expected 0.95 for the normal distribution. This shows that there is a lack of evidence to reject the null hypothesis that the generated paths are normally distributed and we thus choose to accept the null hypothesis.

5.3.2. Lognormal distribution

Similarly to the normal distribution, the scores are computed and compared. In Table 5.3.3 the corresponding values can be found. The same patterns as for the normal distribution can be seen: More datapoints improves the prediction while less data points decrease performance. The scores are again not very sensitive to parameter changes indicating robustness. Furthermore the amount of bins used in prediction has an impact on both the naive scores and actual scores. Finally the results for the approach used and the naive approach are very comparable while it cannot be expected to beat the naive score.

	CE-loss	Naive CE loss	RPS	Naive RPS	RPS (True)	Naive RPS (True)
Baseline	3.836	3.837	1.484 E-4	1.484 E-4	1.284 E-8	1.084 E-8
7 layers LB*4	3.811	3.813	1.505 E-4	1.505 E-4	1.420 E-8	1.144 E-8
3 layers LB/4	3.877	3.878	1.480 E-4	1.480 E-4	1.368 E-8	1.194 E-8
50 bins	3.094	3.094	1.441 E-4	1.441 E-4	1.374 E-8	1.129 E-8
150 bins	4.307	4.309	1.498 E-4	1.498 E-4	1.270 E-8	1.177 E-8
40k datapoints	3.787	3.787	7.390 E-5	7.391 E-5	3.154 E-9	2.740 E-9
10k datapoints	3.841	3.841	2.982 E-4	2.982 E-4	6.454 E-8	5.750 E-8
Kernel 3, LB=27	3.810	3.810	1.482 E-4	1.482 E-4	1.619 E-8	1.233 E-8
Kernel 4, LB=16	3.849	3.85	1.481 E-4	1.481 E-4	1.448 E-8	1.250 E-8
Kernel 4, LB=64	3.879	3.879	1.493 E-4	1.493 E-4	1.271 E-8	1.008 E-8

Table 5.3.3: Numerical results for the losses on the test set for the lognormal distribution

The test statistics are computed using the logarithm of the values generated, since we are dealing with the lognormal distribution. The rate of not rejecting the null hypothesis with $\alpha = 0.05$ is given in Table 5.3.4. Once again, these values indicate that the methodology is able to generate from the underlying distribution.

	Jarque-bera	Anderson-Darling
Baseline	0.94	0.95
7 layers LB*4	0.93	0.97
3 layers LB/4	0.93	0.95
50 bins	0.93	0.95
150 bins	0.94	0.95
40k datapoints	0.97	0.96
10k datapoints	0.91	0.98
Kernel 3, LB=27	0.93	0.95
Kernel 4, LB=16	0.93	0.95
Kernel 4, LB=64	0.93	0.94

Table 5.3.4: Numerical results for the hypothesis testing using the Anderson-Darling and Jarque-Bera test on the lognormal distribution

5.3.3. Uniform distribution

For the uniform distribution the same approach is taken as before. The cross-entropy loss and the RPS are computed and presented in Table 5.3.5.

	CE-loss	Naive CE loss	RPS	Naive RPS	RPS (True)	Naive RPS (True)
Baseline	4.559	4.559	1.51 E-4	1.51 E-4	9.49 E-9	8.92 E-9
7 layers LB*4	4.559	4.559	1.53 E-4	1.53 E-4	9.40 E-9	9.24 E-9
3 layers LB/4	4.559	4.559	1.50 E-4	1.50 E-4	9.42 E-9	8.80 E-9
50 bins	3.819	3.819	1.49 E-4	1.49 E-4	1.07 E-8	1.04 E-8
150 bins	4.984	4.981	1.51 E-4	1.51 E-4	1.33 E-8	1.07 E-8
40k datapoints	4.555	4.554	7.51 E-5	7.51 E-5	2.82 E-9	2.51 E-9
10k datapoints	4.566	4.559	3.03 E-4	3.03 E-4	8.54 E-8	3.89 E-8
Kernel 3, LB=27	4.559	4.559	1.51 E-4	1.51 E-4	9.30 E-9	8.86 E-9
Kernel 4, LB=16	4.562	4.559	1.50 E-4	1.50 E-4	1.36 E-8	8.89 E-9
Kernel 4, LB=64	4.559	4.559	1.51 E-4	1.51 E-4	9.69 E-9	8.89 E-9

Table 5.3.5: Numerical results for the losses on the test set for the uniform distribution

For hypothesis testing the Anderson-Darling test statistic is used. Once again this indicates that we reject the null hypothesis at the same rate as the actual theoretical distribution would have.

	Anderson-Darling no reject rate
Baseline	0.95
7 layers LB*4	0.96
3 layers LB/4	0.95
50 bins	0.95
150 bins	0.95
40k datapoints	0.97
10k datapoints	0.97
Kernel 3, LB=27	0.95
Kernel 4, LB=16	0.95
Kernel 4, LB=64	0.93

Table 5.3.6: Numerical results for the losses on the test set for multivariate correlated normal distribution

As can be seen the null hypothesis that the data comes from the uniform distribution is rejected approximately 5% of the times, which would be the same for the actual uniform distribution. This indicates that there is no strong evidence against the null hypothesis and it is chosen to be accepted so this methodology is able to generate uniform random variables.

	CE-loss	Naive CE loss	RPS	Naive RPS	RPS (True)	Naive RPS (True)
Baseline	3.064	15.152	1.44 E-4	2.86 E-4	6.61 E-7	6.95 E-6
7 layers LB*4	2.997	15.174	1.45 E-4	2.91 E-4	3.23 E-7	7.30 E-6
3 layers LB/4	3.169	15.174	1.44 E-4	2.86 E-4	1.62 E-6	6.98 E-6
50 bins	2.334	14.137	1.35 E-4	2.67 E-4	1.17 E-6	1.45 E-5
150 bins	3.464	15.486	1.47 E-4	2.93 E-4	3.81 E-7	4.60 E-6
40k datapoints	3.000	15.140	7.15 E-5	1.43 E-4	2.40 E-7	3.58 E-6
10k datapoints	3.096	15.228	2.90 E-4	5.78 E-4	1.44 E-6	1.36 E-5
Kernel 3, LB=27	3.053	15.198	1.44 E-4	2.87 E-4	5.86 E-7	6.90 E-6
Kernel 4, LB=16	3.161	15.194	1.44 E-4	2.86 E-4	1.17 E-6	6.75 E-6
Kernel 4, LB=64	3.030	15.213	1.44 E-4	2.89 E-4	3.81 E-7	6.98 E-6

Table 5.4.1: Numerical results for the losses on the test set for the sinoid with noise

5.4. Combination of a deterministic function and randomness

Since in practice there is often noise in series combined with an actual pattern it is vital to be able to pick up on this. The noise can come from measurement errors in a process without actual noise or patterns which depend on the past but are not fully deterministic. To this end a combination of the normal distribution and a sinoid is investigated. When we know the underlying distributions it is not that difficult to test whether or not the generated series corresponds to the underlying dynamics and distribution.
The function chosen is the same as in the deterministic function setting. The noise is applied to the final result and not to the input of the function. This can be seen in Figure 5.4.1.

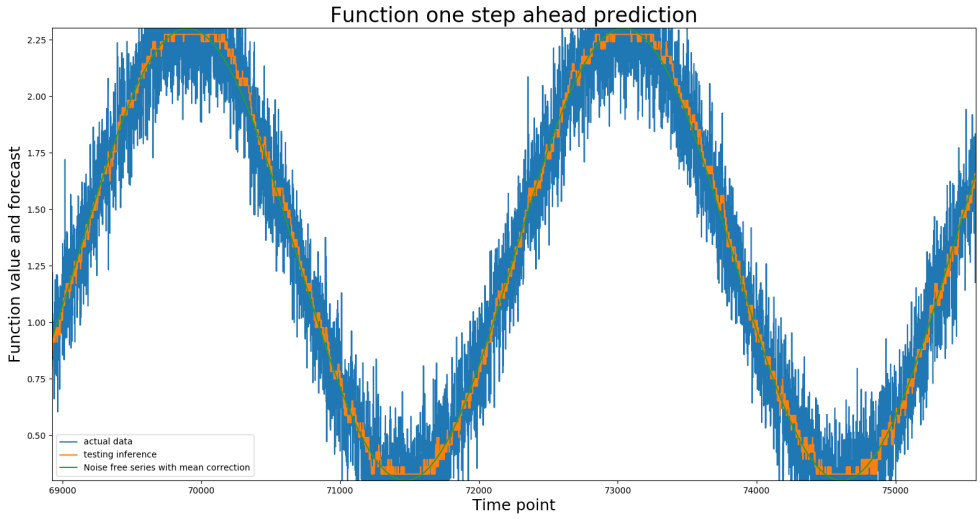

Figure 5.4.1: Noisy sinoid with inference on the expected value of the distribution and the underlying wave.

The MSE is taken against the unnoised signal. The results are comparable to the sinoid wave from before. The same patterns can be seen: more data improves the score, the same holds for a larger receptive field. This means that a combination of a deterministic and noisy signal is picked up on.

	MSE naive	MSE Network	MSE pathwise	MSE distr. mean	MSE distr. median
Baseline	0.0465	0.002475	0.7111	0.14847	1.039658136
7 layers LB*4	0.0466	**0.000676**	**0.0180**	**0.00081**	**0.010079466**
3 layers LB/4	0.0460	0.010697	1.1697	0.51314	0.830097513
50 bins	0.0468	0.002478	0.4277	0.12107	0.658091446
150 bins	0.0459	0.002032	1.0249	0.15530	1.053485299
40k datapoints	0.0466	0.001713	0.8667	0.08819	0.358403665
10k datapoints	0.0467	0.002297	0.6186	0.18647	1.104765408
Kernel 3, LB=27	0.0469	0.002424	1.1591	0.09892	0.927703215
Kernel 4, LB=16	0.0476	0.006493	0.8697	0.44825	1.061506135
Kernel 4, LB=64	0.0470	0.001220	0.3407	0.02263	0.384152328

Table 5.4.2: Numerical results for the losses on the test set for the noisy sine wave

5.5. Multivariate normal distribution

	CE-loss	CE-loss 2	RPS	RPS 2	Naive RPS	RPS (True)	RPS (True) 2	Naive RPS (True)
1D	3.929	3.929	1.49 E-4	1.49 E-4	1.49 E-4	8.46 E-9	8.46 E-9	8.26 E-9
ρ =0	3.883	3.933	1.49 E-4	1.49 E-4	1.49 E-4	6.37 E-3	6.39 E-3	6.37 E-3
$\rho = \pm 0.1$	3.898	3.933	1.49 E-4	1.49 E-4	1.49 E-4	6.45 E-3	6.47 E-3	6.45 E-3
$\rho = \pm 0.3$	3.942	3.890	1.49 E-4	1.49 E-4	1.49 E-4	6.59 E-3	6.59 E-3	6.59 E-3
$\rho = \pm 0.5$	3.965	3.799	1.49 E-4	1.48 E-4	1.49 E-4	6.19 E-3	6.21 E-3	6.19 E-3
$\rho = \pm 0.7$	3.930	3.608	1.49 E-4	1.47 E-4	1.49 E-4	6.44 E-3	6.45 E-3	6.44 E-3
$\rho = \pm 0.9$	3.955	3.126	1.49 E-4	1.44 E-4	1.49 E-4	6.63 E-3	6.63 E-3	6.63 E-3
$\rho = \pm 1$	3.931	0.800	1.49 E-4	6.21 E-5	1.49 E-4	6.31 E-3	6.39 E-3	6.31 E-3

Table 5.5.1: Numerical results for the losses on the test set for multivariate correlated normal distribution

As expected the cross entropy loss for the second series decreases if the correlation increases. There is more information available. If X, Y are distributed $N(0, \sigma^2)$ with correlation ρ and we know the realization of the first series X then the second series Y is distributed $N(\rho X, (1-\rho^2)\sigma^2)$. Since $-1 \leq \rho \leq 1$ the variance decreases the farther away from 0 the correlation gets. This is evident in the CE loss as well as in the RPS. For comparison the univariate normal distribution as described earlier is reported. It can be seen that when we have no correlation, so the random variables are independent since they are normally distributed, there is essentially no useful information in the second series. Conditioning on this second series does however not decrease the performance, which means the network is able to distinguish useful from useless information.

Correlation	Jarque-bera 1	Jarque-Bera 2	Anderson-Darling 1	Anderson-Darling 2
0	0.96	0.95	0.99	0.96
± 0.1	0.96	0.95	0.98	0.96
± 0.3	0.97	0.94	0.98	0.93
± 0.5	0.97	0.93	0.96	0.90
± 0.7	0.94	0.96	0.98	0.93
± 0.9	0.93	0.95	0.96	0.93
± 1	0.95	0.98	0.98	0.94

Table 5.5.2: Numerical results for the losses on the test set for multivariate correlated normal distribution

Apart from the scores it is important to check whether generating with this approach provides samples that are normally distributed or close enough that it is indistinguishable. In Table 5.5.2 the frequency of non-rejection of the null hypothesis is reported. These are in line with the expected frequency of the normal distribution which is 0.95. Therfore there is no clear evidence to reject the hypothesis that the generated distribution with this methodology is the normal distribution. This means we can generate and sample from multivariate distributions with and without dependence using the methodology.

6

Conclusion

In this thesis scenario generation with the help of neural networks is investigated. First a background was given on other ways to generate scenarios including ARIMA type models and Bayesian time series models. A justification for economic scenario generation was presented to establish that this has been an extension of previous modelling to possibly prevent a future financial crisis and is therefore taken into regulatory frameworks such as IFRS 9 and Solvency II.

By choosing not to go for a parametric model and apply many assumptions, the choice was laid upon neural networks as a basis for the scenario generation. The neural network framework was based upon the previous WaveNet by [7] and [21]. A novelty in this research was trying to apply a different loss function to this network for scenario generation.

For multivariate forecasting there were several approaches available: copulas, multivariate distribution and Gibbs sampling. The multivariate distribution was not chosen since it would suffer heavily from the curse of dimensionality. Copulas were dismissed since they would imply a less flexible dependence structure or a non-parametric copula. The Gibbs sampling approach was preferred over this, especially since the focus of this thesis is on scenario generation.

Afterwards research on the Ranked Probability Score was performed and coupled to the issue of generating scenarios and forecasts. It was proven that the Ranked Probability Score is a strictly proper scoring rule. This means that the expected score is minimized when the forecasted/predicted distribution is the actual underlying distribution. This also holds for the cross-entropy loss. The reason the RPS was looked into was that the cross-entropy loss lacked dependence on distance. For economic scenarios it can be valuable to be close to the realization, a property that the cross-entropy loss does not have whilst the RPS does.

Initially the idea was to also use the RPS as the error score in the neural network. However, optimizing this showed that there was no convergence to a good solution. Thorough checking with different simple datsets or giving the optimal initial solution showed that the RPS was not desirable as a loss function. This is due to the added complexity of the function taking distance into account which makes the optimization problem more complex and empirically getting stuck in local minima.

From this the methodology used in this thesis was chosen to have stacked dilated convolutions based on the WaveNet approach. This allowed for conditional forecasting. Combining these with Gibbs sampling it was possible to generate multivariate series.

A diverse series of data sets were constructed to assess the performance and robustness of the methodology on different characteristics. It could be seen that the performance was not sensitive to parameter changes for all models which was desired. The performance on long term dependence could be seen in the sinoid series and it was clear that the model was able to outperform the naive approaches here.

For the stochastic series the results showed that the WaveNet type approach is able to learn the shape of these distributions and able to generate samples from them that cannot be rejected with statistical tests. Therefore this method might be very suitable for modelling or sampling from distributions for which the exact properties are not known.

The approach also works for some tested combinations of deterministic and stochastic series, as well as multivariate generation. The only series tested upon which the results were under performing was the Lorenz system. This was specifically chosen and it could beforehand already be seen that this approach is not optimal for the Lorenz map time series, since it is chaotic thus ensuring no long term dependencies, no randomness and the necessity of very accurate measurements. Due to the binning approach the predictions or scenarios generated can never be as accurate for such a system.

A short comparison was also made to Feed Forward Neural Networks, which were overfitting strongly whereas the neural network used in this thesis did not.

The consideration for the optimizers was especially relevant due to the failure of the RPS as a loss function. After thorough evaluation, the conclusion is drawn that while the RPS might be a promising way to measure the difference between distributions, especially when taking the distance into account, it is a function that is hard to optimize. When comparing to the cross-entropy loss, the computation is more involved and that is most likely why the optimization methods did not converge to a promising solution, even for relatively simple problems.

The further research recommendations and relevant drawbacks of the methodology are discussed in chapter 7.

7

Discussion and research recommendations

As a possible extension the inference can be sped up using similar methods as [38]. At the moment training is fast but inference is slow. This method introduced by Google Deepmind researchers constructs another network which can be seen as dual or adjoint to the original network that trains slow but does fast inference. This will of course be very efficient when generating many different scenarios, especially when taking many time steps into the future.

A potentially interesting research area is anomaly detection. Since the method used in this thesis produces probabilities, it can be used to assess whether extreme events or anomalies take place when the realization falls into a bin with very low probability. This could be researched in the future for a wide range of data sets, the power of the method used here is its flexibility and robustness.

A downside to the method of forecasting the distribution is that it is hard to assess whether the predicted distribution follows the same shape as the actual underlying distribution. This is especially true since there is a single observation every time.
In the methodology in this paper the assumption is made that the behaviour of the past N observations fully determines the probability distribution for the next time step. While this is often a realistic scenario, it is also assumed that this does not change over time. In reality that may not be true, there may be trends over time that are not incorporated into this methodology. This is something that holds generally for modelling: if the assumptions are not met the results are not guaranteed to make sense.

For future research it may be interesting to investigate the performance of the neural network forecasts with a copula. A combination of this appraoch with the Dynamic Copula theory by Eban [15] might be promising. This would mean that instead of using the Gibbs sampling to get multivariate scenarios the copulas can be used. This is especially useful if there is knowledge available about the dependence structure.

A note is that the bin sizes in this thesis are set to be the same width. As shown they are allowed to be flexible. It is even possible to have a probability distribution over this bin specified. This allows one to reconstruct any possible distribution. This could incorporate aspects such as rare events by taking the final tail bins with either a different model or for instance a generalized extreme value distribution. Furthermore if the knowledge about the range of the dynamics is available it may be interesting to investigate a certain area more closely. This could easily be done by decreasing the bin size in this part giving more information there. All methodologies used in this thesis can still easily be applied this way. These more generalized applications can be interesting for further research.

Once again the data quality and how representative it is should be stressed. If the data on which training happens does not accurately describe the behaviour for the entire set, good results are less likely to happen since one of the fundamental assumptions of the modelling approach has been violated.

Another future research recommendation is to test on chaotic series such as a double pendulum. These systems are highly sensitive to small disturbances.
Another possibility that has not been applied in practice here is to have different distributions over the bins.
An important note is that while these methods can produce very good results they are often not easily comprehensible. This is one of the points critics focus on with neural networks, they can be labelled as black box methods. On the other hand, if we choose a specific model and know that the assumptions do not hold in reality the model might also be inappropriate.
The framework used in this thesis could also be applied to Generative Adversarial Networks wherein two different networks compete, one network that generates instances, the generator, and one network that assesses whether or not the instance is real or fake. The generating part of the network could be done in a way similar to the WaveNet approach in this thesis.
When looking at a model it is interesting to see if there are connections to other approaches as perhaps similar results could be proven or be useful. It is possible to describe the setup as above in terms of a Markov chain. The easiest way to see this is to look at the bins of the distribution as the states of the Markov chain.
More research can be done in applying this methodology to a wider array of time series, deterministic, stochastic, mixtures of these and multivariate series. Once the performance is known for several properties, they can be compared to other modelling and generation approaches and finally compared on data sets of which the underlying dynamics are not known.

Bibliography

[1] E. M. Varnell, Economic scenario generators and solvency ii, British Actuarial Journal 16, 121–159 (2011).

[2] S. Lerch and T. L. Thorarinsdottir, Comparison of non-homogeneous regression models for probabilistic wind speed forecasting, Tellus A: Dynamic Meteorology and Oceanography 65, 21206 (2013), https://doi.org/10.3402/tellusa.v65i0.21206 .

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, Latent dirichlet allocation, Journal of Machine Learning Research 3, 2003 (2002).

[4] M. G. Bellemare, W. Dabney, and R. Munos, A distributional perspective on reinforcement learning, CoRR abs/1707.06887 (2017), arXiv:1707.06887 .

[5] Y. LeCun and C. Cortes, MNIST handwritten digit database, (2010).

[6] H. Sangrody, N. Zhou, and X. Qiao, Probabilistic Models for Daily Peak Loads at Distribution Feeders, ArXiv e-prints (2017), arXiv:1703.06378 [stat.AP] .

[7] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, WaveNet: A Generative Model for Raw Audio, ArXiv e-prints (2016), arXiv:1609.03499 [cs.SD] .

[8] M. Binkowski, G. Marti, and P. Donnat, Autoregressive convolutional neural networks for asynchronous time series, CoRR abs/1703.04122 (2017), arXiv:1703.04122 .

[9] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, Understanding deep neural networks with rectified linear units, CoRR abs/1611.01491 (2016), arXiv:1611.01491 .

[10] R. Shumway and D. Stoffer, Time Series Analysis and Its Applications, Springer texts in statistics (Springer, 2000).

[11] P. Brockwell and R. Davis, Introduction to Time Series and Forecasting, Introduction to Time Series and Forecasting No. v. 1 (Springer, 2002).

[12] J. Hamilton, Time Series Analysis (Princeton University Press, 1994).

[13] A. Harvey, Forecasting, Structural Time Series Models and the Kalman Filter (Cambridge University Press, 1990).

[14] A. Priel and I. Kanter, Time series generation by recurrent neural networks, Annals of Mathematics and Artificial Intelligence 39, 315 (2003).

[15] E. Eban, G. Rothschild, A. Mizrahi, I. Nelken, and G. Elidan, Dynamic copula networks for modeling real-valued time series, in Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, Vol. 31, edited by C. M. Carvalho and P. Ravikumar (PMLR, Scottsdale, Arizona, USA, 2013) pp. 247–255.

[16] C. Esteban, S. L. Hyland, and G. Rätsch, Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs, ArXiv e-prints (2017), arXiv:1706.02633 [stat.ML] .

[17] J. Feldman and R. Rojas, Neural Networks: A Systematic Introduction (Springer Berlin Heidelberg, 1996).

[18] S. Hochreiter and J. Schmidhuber, Long short-term memory, (1997) pp. 1735–1780, https://doi.org/10.1162/neco.1997.9.8.1735 .

[19] N. Lintz, Image on convolutional neural networks. (2018).

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in Advances in Neural Information Processing Systems 25, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Curran Associates, Inc., 2012) pp. 1097–1105.

[21] A. Borovykh, S. Bohte, and C. W. Oosterlee, Conditional Time Series Forecasting with Convolutional Neural Networks, ArXiv e-prints (2017), arXiv:1703.04691 [stat.ML] .

[22] F. Yu and V. Koltun, Multi-scale context aggregation by dilated convolutions, CoRR abs/1511.07122 (2015), arXiv:1511.07122 .

[23] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, (2015) arXiv:1512.03385 .

[24] T. Gneiting and A. E. Raftery, Strictly proper scoring rules, prediction, and estimation, Journal of the American Statistical Association 102, 359 (2007), http://dx.doi.org/10.1198/016214506000001437 .

[25] R. Lesego Machete, Contrasting Probabilistic Scoring Rules, ArXiv e-prints (2011), arXiv:1112.4530 [math.ST] .

[26] E. Epstein, A scoring system for probability forecasts of ranked categories, Journal of Applied Meteorology 8, 985 (1969).

[27] A. Murphy, On the "ranked probability score", (1969) pp. 988–989.

[28] H. Hersbach, Decomposition of the Continuous Ranked Probability Score for Ensemble Prediction Systems, (2000) pp. 559–570.

[29] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos, The cramer distance as a solution to biased wasserstein gradients, CoRR abs/1705.10743 (2017), arXiv:1705.10743 .

[30] B. Polyak, Some methods of speeding up the convergence of iteration methods, in Ussr Computational Mathematics and Mathematical Physics, Vol. 4 (1964) pp. 1–17.

[31] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, On the importance of initialization and momentum in deep learning, in Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13 (JMLR.org, 2013) pp. III–1139–III–1147.

[32] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, (2014) arXiv:1412.6980 .

[33] T. Tieleman and G. Hinton, Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Networks for Machine Learning (2012).

[34] T. Dozat, Incorporating nesterov momentum into adam, (2015).

[35] C. M. Jarque and A. K. Bera, Efficient tests for normality, homoscedasticity and serial independence of regression residuals, (1980) pp. 255 – 259.

[36] T. W. Anderson and D. A. Darling, A test of goodness of fit, Journal of the American Statistical Association 49, 765 (1954), https://www.tandfonline.com/doi/pdf/10.1080/01621459.1954.10501232 .

[37] S. L. Smith, P.-J. Kindermans, and Q. V. Le, Don't decay the learning rate, increase the batch size, in International Conference on Learning Representations (2018).

[38] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions, ArXiv e-prints (2017), arXiv:1712.05884 [cs.CL] .