INCOMPLETE TIME SERIES FORECASTING USING GENERATIVE NEURAL NETWORKS

by

HARSHIT TARUN SHAH

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at The University of Texas at Arlington.

December, 2020

Arlington, Texas

This thesis is dedicated to my mom, dad and sister.

Acknowledgements

I would like to deeply thank Dr. Manfred Huber for accepting my request to serve as a thesis supervisor. His vast expertise in the multiple areas of research has helped from selecting a domain area for research to completing the research itself. I want to thank him for always answering my questions and directing me in the right direction. I am also grateful to him for letting me use the lab and its resources for my research.

I would also like to thank Dr. Farhad Kamangar and Dr. Vassilis Athitsos for taking out their valuable time to serve as my committee members. Their courses which I had enrolled into in my past semesters, helped me build relevant background knowledge for my research.

I also want to take this opportunity to thank Mr. Sri Sridharan and Mrs. Shruti Shah for motivating me through my master's program and providing me with valuable advice every time I doubted myself. They have always made me feel home in U.S.A and this thesis wouldn't have been possible without their support.

I also want to thank my friends Deep, Arya, Nishad, Haresh, Harish, Preet, Tushar, Akshay, Yugesha, Meet, Harshil, Ravi and Chintan for making my time at UTA fun and enjoyable. They have always looked out for me and motivated me throughout my research.

Lastly, I want thank my parents for believing in me and supporting me financially. Their unconditional love has given me strength to complete my master's program successfully.

Incomplete Time Series Forecasting Using Generative Neural Networks

by

Harshit Tarun Shah

Department of Computer Science and Engineering

The University of Texas at Arlington

Supervising Professor: Dr. Manfred Huber

Abstract

Dealing with missing data is a long pervading problem and it becomes more challenging when forecasting time series data because of the complex relation between data and time, which is why incomplete data can lead to unreliable results. While some general purpose methods like mean, zero, or median imputation can be employed to alleviate the problem, they might disrupt the inherent structure and the underlying data distributions. Another problem associated with conventional time series forecasting methods whose goal is to predict mean values is that they might sometimes overlook the variance or fluctuations in the input data and eventually lead to faulty predictions. To address these issues, we employ a probabilistic forecasting technique which can accommodate the variations in data and predict a full conditional probability distribution of future values given past data. We introduce a novel generative adversarial network (GAN) architecture with the goal to forecast a probability distribution on time series data and also introduce an auxiliary GAN which learns the temporal pattern of which data is missing, thereby removing the dependency on using general purpose imputation methods. We create two complex time series datasets to test our architecture and also show a comparison between our architecture's forecasting capability (with incomplete data) to a state-of-the-art architecture which is trained with complete data. We also demonstrate that our model's predicted data distribution does not collapse with incomplete data, but instead successfully learns to estimate the true underlying data distribution.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We live in a world of data and nowadays data is generated almost everywhere: sensor networks on Mars, submarines in the deepest ocean, opinion polls about any topic, etc. Many of these real world applications suffer a common problem, missing or incomplete data. Missing data can here result from a number of causes, including sensor or communication failures, resource limitations, or the simple fact that the rate at which different forms of data can be acquired can be vastly different, leading data points in the sensory time series where only subsets of the data are available. For example, in an industrial experiment some results can be missing because of mechanical or electronic failures during the data acquisition process. In medical diagnosis, some tests cannot be done because either the hospital lacks the necessary medical equipment, or some medical tests may not be appropriate for certain patients. In the same context, another example could be an examination by a doctor, who performs different kinds of tests; some test results may be available instantly, others may take several days to complete.

These challenges can be alleviated using various machine learning or filtering and estimation techniques. However, these data imputation techniques are always based on a set of assumptions and will yield good results only if these assumptions are approximately correct. After the advent of deep learning, solving even very complex tasks such as image classification, image generation, etc has become a commonplace. This has also been supported by a significant increase in computation power which acts as a backbone when training deep neural networks by developing advanced computation units such as GPUs. Hence we can also tackle these missing data challenges using the knowledge from the field of deep learning and statistical modeling, and use the newer computational units to produce the results faster.

The goal of this thesis is to tackle the problems that occur when dealing with complex, incomplete time-series sensor related data and propose a new generative neural network architecture to forecast values for situations when part of the data is missing or incomplete, by leveraging the rich structure in the available data.

Chapter 2

Background

In this chapter we will introduce the neural network architectures and other formal definitions used in the context of time series and underlying the work performed in this thesis.

2.1 Time Series Data

In this work we primarily focus on working with sensor data which falls into the domain of time series. A time series is basically a series of data points indexed with respect to time. A single data point at any point in time is called a time step. examples for sensory time series include data from a sensor recording humidity and temperature values in a house every second or every minute, a video recording which is simply a camera capturing pictures at every instance of time more formally known as frames. similarly, the Stock price of a company which is recorded every second also comes under the domain of time series.

One significant way to differentiate different types of time series data is in terms of the dimensionality of each data point:

- **Univariate time series**: Univariate time series are defined as series of records of data involving only one variable.
- **Multivariate time series**: Multivariate time series are defined as series of records of data involving two or more variables.

One of the differentiating features between these types here is that in univariate data, the only dependencies are over time while in multivariate time series data both dependencies over time and dependencies between variables have to be considered when performing missing value imputation or forecasting.

2.1.1 Components

As we know that data in time series changes over time we can identify some key components in time series data:

- **Seasonality**: A seasonal effect is simply a pattern observed in data in a systematic way at specific intervals of time. Eg: Sales increase in the month of December due to Christmas.
- **Cyclical Variation**: Sometimes change takes place over longer time periods. For instance, the US economy seems to go through periods of expansion and recession once every decade or so. This is a cyclical variation.
- **Trend**: A trend is a non-cyclic, long term pattern observed in the series. We might notice multiple upward and downward trends in data but there can be an overall upward trend.
- **Noise**: Noise exists practically everywhere and represents those irregular fluctuations that accompany a signal but are not a part of it and tend to obscure it.

2.1.2 Representation

Like any other tabular data we can also represent time series data in tabular form, the only caveat being that the order of data points matter in a time series since the order represents the data collected at that instance of time. Hence we represent a time series using a M X N matrix for the multivariate case and M X 1 for the univariate case:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots \\ \vdots & \ddots & \\ a_{M1} & & a_{MN} \end{bmatrix}$$

where M is the number of time steps in the time series and N is the number of features or variables in each data point.

2.1.3 Time Series Forecasting

Time series forecasting is the process of predicting the data of future time steps. It can be thought of as a mechanism to process or analyze the historical data and use that learned information to make predictions about the future. Eg: Predicting the price of a company's stock on the next day.

Time series forecasting can be classified into two types:

1. **One-step ahead prediction**: This method is concerned only with predicting the next time step given the time series. Eg: Predicting the value of a sensor in the 11^{th} second given 10 seconds of past data.
2. **Multi-step ahead prediction**: This, as the name suggests, is concerned with predicting a sequence of time steps given a time series. Eg: Forecasting the weather for the next 10 days given the past data.

Regression Based Time Series Forecasting

Conventional statistical methods like ARMA and ARIMA [4], some machine learning methods like Support Vector Machines, and newer, more advanced neural network based forecasting techniques are usually concerned with predicting $\mu(P(x_{t+1}|X))$ where x_{t+1} is the future prediction and X is the historical time series. Generally, these methods follow the principles of mean regression and because of that they inherit the main limitation of these principles, i.e. they do not include the fluctuations around the mean value. Hence the results can be misleading in some cases. Hence we need to employ a method to consider all those fluctuations around the mean which can be solved by a method known as probabilistic forecasting.

Probabilistic Forecasting

A probabilistic forecast represents an estimation of the respective probabilities for all possible future outcomes of a random variable. In contrast to mean forecasting, the probabilistic forecast represents a probability density function. In the following we formalize somewhat the notion of probabilistic forecasts in the context of time-series. Let X_t be the time series containing the historical data up to time t.

We can model the future time step at time t+h for each forecasting step h as follows:

$$x_{t+h} = f_h(X_t)$$

where: f_h is a forecasting model specific to time step h. If $h > 1$ it becomes a multi-step ahead forecasting problem.

Here f_h is a singe value forecasting model. This can be modified into a probabilistic forecasting model:

$$\mathrm{x}_{t+h} = F_h(X_t)$$

Here F returns a random variable x_{t+h} and not a single value. Hence the goal of F_h is to learn the conditional probability distribution of $P(\mathrm{x}_{t+h}|X_t)$ rather than a single value.

In line with this, the goal of this research is to build a model which can forecast the full conditional probability distribution of future predictions.

2.1.4 Missing Data

The presence of missing values in a time series is a big problem in real world classification and forecasting tasks. It mainly occurs due to sensor failures, reboots, or human error while recording data, but it can also be the simple result of limited data acquisition resources or different acquisition rates of different sensor modalities. Missing data introduces inconsistencies in data, and as a consequence can suggest a different distribution from the true data distribution. However, depending on what causes missing data, the gaps in the data will themselves have certain distributions. For example, if different sensor rates are the cause of missing data, the missing data time slots will have a cyclic, repetitive pattern, while in cases where sensor failures are the cause, the pattern of missing data will be long, contiguous blocks with random start times. Understanding these distributions may be helpful as they can provide additional information about the pattern of missing data. In the approach introduced in this thesis we will explicitly try to learn this pattern and we will discuss more about how we can leverage this information in order to forecast data when we have time series with missing information in Section 4.1.

Mechanisms leading to missing data can be divided into three types based on the statistical properties of the pattern with which data is missing:

1. **Missing completely at random (MCAR)**: In MCAR there is no pattern observed on the way the data is missing. Missing data points occurs entirely at random. This means there are two requirements: First, the probability for a certain observation being missing is independent of the values of other features. Second the probability for an observation being missing is also independent of the value of the observation itself. In the univariate case this means that the probability for a certain observation being missing is independent of the point of time of this observation in the series. If A represents the vector indicating whether each data item will be missing or available and $B_{observed}$ and $B_{missing}$ are the values of the observed and of the missing data items, respectively, then MCAR corresponds to the case where the probability of data items missing is independent of all data:

   $$P(A|B_{observed}, B_{missing}) = P(A)$$

   Eg: Sensor recording data for 24 hours everyday and sending it to backend systems but that due to unknown reasons and on random occasion fails to transmit data.

2. **Missing at random (MAR)**: Like in MCAR, in MAR the probability for an observation being missing is again independent of the value of the observation itself. But it is dependent on other variables. Since there are no other variables other than time for univariate time series, it can be said, that in MAR the probability for an observation being missed is dependent on the point in time of this observation. This implies that the likelihood of data missing can depend on the observed information but not on the missing information:

   $$P(A|B_{observed}, B_{missing}) = P(A|B_{observed})$$

   Eg: Machine sensor data are more likely to be missing on weekends due to sensor reboots and shutdowns.

3. **Not missing at random (NMAR)**: NMAR observations are not missing in a random manner. This data are not MAR and not MCAR. That means, the probability for an observation being missing depends on the value of the observation itself. Furthermore the probability can (but must not necessarily) dependent on other variables (or point of time for univariate time series).

$$P(A|B_{observed}, B_{missing}) = P(A|B_{observed}, B_{missing})$$

   Eg: Temperature sensor that gives no value for temperatures over 100 degrees.

In this work we will be only dealing with the Missing at random **(MAR)** case.

2.2 Recurrent Neural Networks

The key difference in time series data from conventional image or tabular data is the role of time. Data observed at each instance in time has a significance since it is dependent on the data in the past; even while reading this our brain constantly keeps history of past words and sentences in memory which helps it make sense of the current words. In the same way in a time series dataset the current observation is somehow dependent on its history and we can exploit this characteristic to forecast future data. However the traditional feed-forward neural networks can not do this since they do mot have an ability to establish temporal memory, which is why recurrent neural networks(RNNs) were introduced. These networks can address this issue since they have loops within them thereby allowing information to persist and thus the potential for temporal memory to be established. In the diagram in Figure 2.1, **A** looks at some input x_t at time step **t** as well as at some hidden state from the previous time step, **t-1** and outputs a value h_t. This can seem a bit mysterious, however it it can be looked at as just a regular neural network with multiple copies of same network, each passing a message to a successor. The unrolled representation of a RNN is shown in Figure 2.2. After looking at the unrolled version

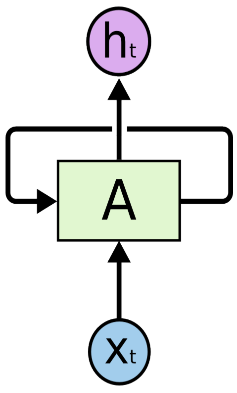


Figure 2.1: Recurrent Neural Networks

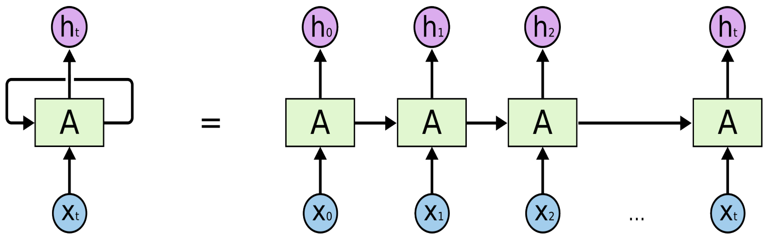


Figure 2.2: An Unrolled RNN

of the RNN the architecture becomes very intuitive. It basically takes input at some time step, process it, produces the output and sends the latent information to the next copy of the network which essentially creates a memory mechanism. In the last few years RNNs have been used for a variety of problems such as speech recognition, language modeling, image captioning, etc. Hence it automatically becomes a natural choice to use RNNs when dealing with time series data.

Modeling long term dependencies

As we have seen, RNNs can be useful in the context of time series data by retaining information from the past and connect it to the present tasks. Sometimes when we only have to look at a relatively recent piece of information to perform the present task, for example when trying to predict the heart rate of a person when walking, this would be possible based mainly on very recent information since it would be similar to the most recent few values since heart rate remains relatively stable during walking. In such cases, where the gap between the relevant information and the place that it is needed is small, RNNs can learn to use the past information relatively effectively. However, there are cases where we might need more context and in particular context that is temporally relatively far removed from the point in time at which it is relevant. Consider, for example, a case when a person runs every morning and only walks during the day. Since the gap between today morning and tomorrow morning is big, RNNs might predict the person walking even the next morning, which is not true because there exists a pattern that becomes clear after a longer period of time. Unfortunately as that time gap grows, RNNs become unable to learn to connect the information. This problem was explored in depth [5], and some pretty fundamental reasons why it might be difficult were found. To address this, special forms of recurrent networks were developed that aim to maintain information over longer time spans. The most popular of these are Long Short Term Memory networks (LSTM) that do not have this problem to the same degree.

2.2.1 Long Short Term Memory Network

To alleviate the problem of long term dependencies in RNNs, Long Short Term Memory (LSTM) [6] ,a special kind of RNNs were introduced. LSTMs are better at retaining the longer term dependencies through the use of gaiting mechanisms, which is one of the main reason why they are used frequently in the context of time series data. The problem actually faced by the RNNs is the problem of vanishing gradients [7] which is the phenomenon where the gradients becomes so small that it is practically impossible to learn anything for the network. To avoid the problem of vanishing gradients LSTMs are frequentlyused. In the same way as RNNsLSTMs also have the form of a chain of repeating modules but their cell structure is different.

Cell structure of LSTM

Since LSTMs were designed with a goal to work with long sequences of data there is an inherent difference in cell structure. By cell we mean the module which repeats to retain information. The LSTM's core concept revolves around the idea of gates which can also be thought similar to the knobs in water pipes controlling the flow of water. However in this case they control the flow of relevant information.

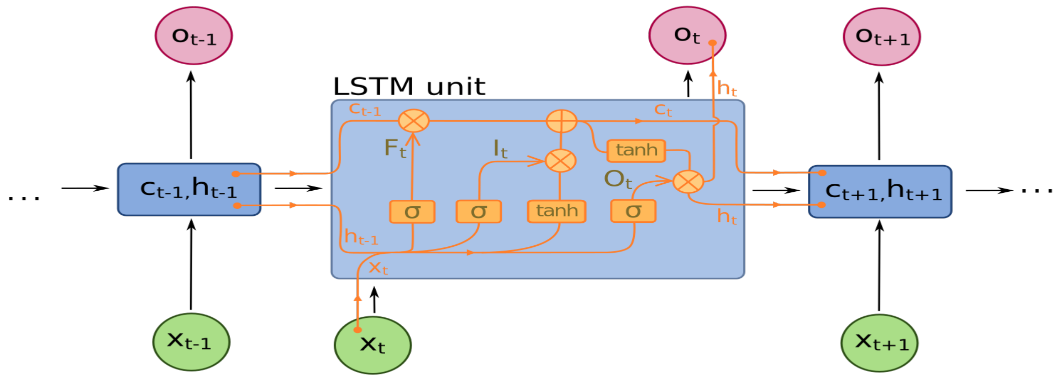


Figure 2.3: LSTM Cell Structure

The cell state here acts as a transport highway that delivers relevant information all the way down the sequence chain. It basically represents the memory of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from significantly earlier time steps can make its way to later time steps, reducing the effects of short term memory. As the cell state travels through time, information get's added or removed to the cell state via gates. The gates are different neural network components that decide which information should be part of the cell state. The gates can learn what information is relevant to keep or forget during training.

The gates contain sigmoid activations as shown in Figure 2.3. Since the sigmoid activation function can squash values between 0 and 1, it is helpful to update or forget data because any input that gets multiplied by 0 becomes 0, causing the values to disappear or be forgotten. Any input multiplied by 1 is the same value, which mean the value is kept. The network can learn which data is important and which is not and accordingly remember or forget.

Forget Gate

This gate decides what information to keep or thrown away. Labelled as F_t in

Figure 2.3 it gets information from the previous hidden state and information from the current input and passes a trainable linear function of this through the sigmoid function. The output is between 0 and 1 where anoutput close to 0 means to forget and close to 1 means to keep the information currently in the cell state. This gate is formulated as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_f)$$

where $\sigma()$ is the sigmoid function, W_f is a trainable weight vector, h_{t-1} represents the hidden state of the LSTM from the previous time step and x_t is the input at time step t.

Input Gate

This gate (labelled I_t in Figure 2.3) is responsible for updating the cell state. First a trainable linear function of the previous hidden state and current input is passed to a sigmoid unit which decides which values to update. A separate trainable linear function of the previous hidden state and current input is also passed to a hyperbolic tangent activation which squashes the values between -1 to 1 to help regulate the network followed by a product between the output of the sigmoid and hyperbolic tangent activation to update the state. This gate is formulated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_i)$$

Output from the hyperbolic tangent activation is formulated as follows:

$$\check{\mathbf{c}}_t = tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_c)$$

Cell State

Both the forget gate and the input gate are used to update the cell state where the forget gate helps remove information previously held in the cell state while the input gate regulates how to update the remaining content of the cell state using the current input. To achieve this, the cell state first gets pointwise multiplied by the forget vector (this provides the possibility of dropping values in the cell state if it gets multiplied by values near 0). Then the output of the input gate is pointwise added to obtain a new cell state that the network found relevant. This is formulated as:

$$\mathbf{c}_t = \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \check{\mathbf{c}}_t$$

where $\otimes$ is pointwise multiplication.

Output Gate

The last gate in the LSTM cell is the output gate which decides what the next hidden state should be, i.e. what information about. the cell state should be passed on to the next time step. First another trainable linear function of the previous hidden state and the current input is passed through a sigmoid activation. Then the newly modified cell state is passed through the hyperbolic tangent activation. Finally the pointwise product between hyperbolic tangent output and sigmoid output is taken to decide what information the hidden state should carry. The new cell state and the new hidden state is then carried over to the next time step. The output gate is formulated as:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, x_t] + \mathbf{b}_o)$$

And the new hidden state is formulated as follows:

$$\mathbf{h}_t = \mathbf{o}_t \otimes tanh(\mathbf{c}_t)$$

As indicated, in all the above formulations $\mathbf{W}$ represent weight matrices for the respective gates. $\mathbf{h}_{t-1}$ represent the hidden state from previous cell and x_t represent the input from current time step.

2.3 Temporal Convolutional Networks

Recently Convolutional Neural Networks(CNNs) have been in the lime light for reaching human level performance in the domain of image processing and recognition. There have been multiple architectures [8, 9] that have won the ImageNet challenge. Motivated by that researchers have started using CNNs for time series analysis [10] as well. CNNs when used in time series analysis are commonly referred to as Temporal Convolutional Networks(TCNs).

2.3.1 1-D Convolutions

A temporal convolution can be seen as applying and sliding a filter over a time series. Unlike images, the filters exhibit only one dimension instead of two. Specifically, when we are convoluting a filter of size 3 with a univariate time series where the filter values are equal to $[1/3, 1/3, 1/3]$, the result would be a moving average. The general form for a 1-D convolution operation would be as follows:

$$\mathbf{C}_t = activation(\mathbf{W} * \mathbf{X}_{t-l/2:t+l/2} + b) | \forall t \in [1, T]$$

where $\mathbf{C}_t$ denotes the result of convolution, $\mathbf{X}$ denotes a univariate time series of length T. $\mathbf{W}$ denotes a filter of length l, and b corresponds to the bias. Figure 2.4 shows a pictorial representation of a 1D convolution. The filter runs through all the time steps and generates a new representation.

Why TCNs work

CNNs have been suggested to learn specific patterns in image data [11] hence using this feature learning capability of CNNs and applying them to the domain of time series has also produced impressive results [12, 13, 14]. 1D convolutions with a single layer extract features from raw input data using learnable kernels. By computing the activations of 1D convolutions at different regions in the same input

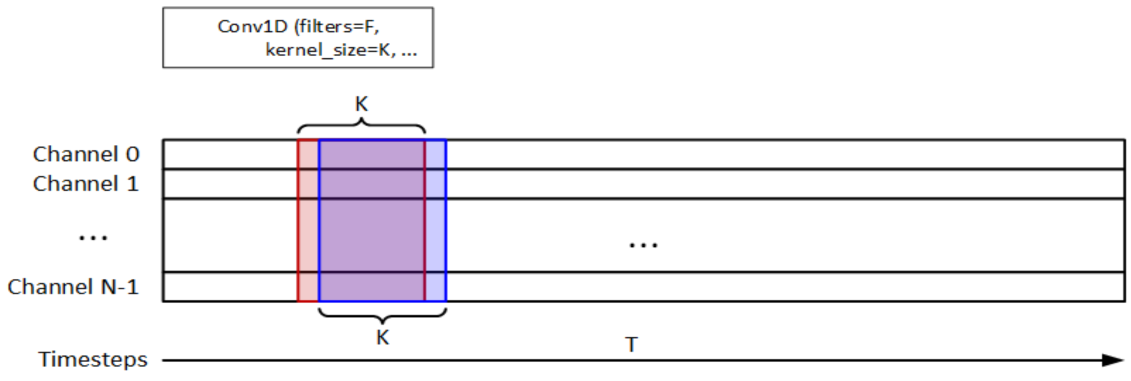


Figure 2.4: 1-D Convolution

we can detect patterns captured by the kernels, regardless of where they occur. A kernel which can detect specific patterns in the input data would act as a feature detector. Since the convolution operation uses the same filter/kernel to find the output for all time steps, it also helps them learn filters that are invariant across the time dimension. Figure 2.5 (from [15]) presents an excellent example of what TCNs learn on the GunPoint dataset, a dataset containing hand position data sequences for two movement classes.

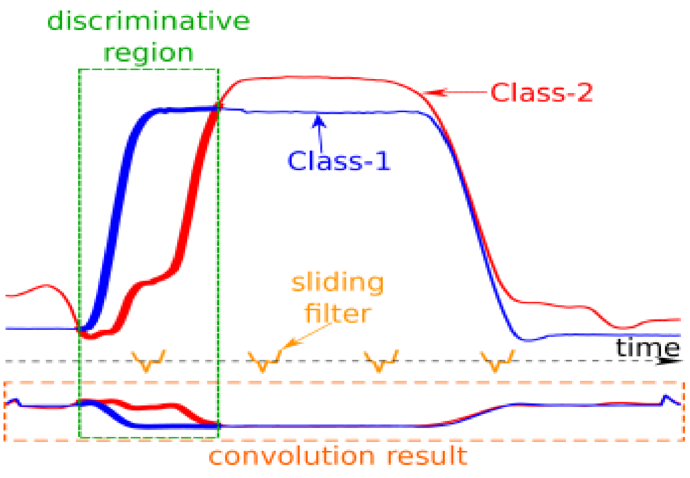


Figure 2.5: Patterns Learned by TCN

We can clearly see in the convolution result that (by applying learned sliding filters) TCN is able to discriminate between class 1 and class 2.

Temporal convolution, similar to LSTM networks learn temporal patterns in the data. However, temporal convolution has the advantage that it can see the entire section of the time series at the same time while LSTMs can only see one data point at a time. This tends to make temporal convolution more stable and easier to train than LSTMs. On the other hand, this also forces the maximum length of a pattern to be pre-determined and encoded into the design of the network, which is not necessary for LSTMs which can in principle learn temporal patterns of arbitrary length.

2.4 Generative Models

Recently there have been surveys [16, 17] claiming that we generate more than a million gigabytes of data every day and that this data could be leveraged by training machine learning models to build intelligent systems. However state of the art machine learning systems and neural networks still largely solve supervised learning problems where the data collected needs to be labelled, which is not always possible in the real world. The real world is full of unstructured data where the conventional supervised models do not work. Hence we need to approach the unsupervised problem with different types of models. The main challenge here is to develop models and algorithms that can analyze and understand this treasure trove of data.

Generative models are one of the most promising approaches towards this goal. To train a generative model we first collect a large amount of data in some domain (e.g., millions of images, sentences, or sounds, etc.) and then train a model to generate data like it, thus developing an implicit model of the data itself.

2.4.1 Generative Adversarial Networks

Generative Adversarial Networks (GAN) [18] are one of the most interesting and popular applications of Deep Learning. Recently, there has been a significant amount of interesting research investigating [19] applications of GANs in many different domains.

Architecture of a Generative Adversarial Network

The basic GAN (commonly referred to as Vanilla GAN) consists of two neural networks, one for generating the desired data called the Generator, and one for differentiating between the real data and fake data called the Discriminator. The generator model could be thought of as analogous to a watch forger whose goal is to produce the most real looking watches and sell them to the watch seller, while the discriminator model is analogous to a watch seller whose goal is to detect fake watches. The competition in this game drives both generator and discriminator to improve their methods until the fake watches are indistinguishable from real ones. Figure 2.6 depicts the architecture trying to solve an image generation task. The generator G takes as input some white noise z (sampled from ρ_{noise}) and trains to generate a realistic looking image whose distribution follows the true data distribution ρ_{data}, whereas the discriminator D optimizes to discriminate between real images and generated images by outputting a probability whether a particular image is true or fake. Basically, G and D play a two-player minimax game with value function $V(G, D)$. Formally:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim \rho_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim \rho_{noise}(z)}[\log(1 - D(G(z)))]$$

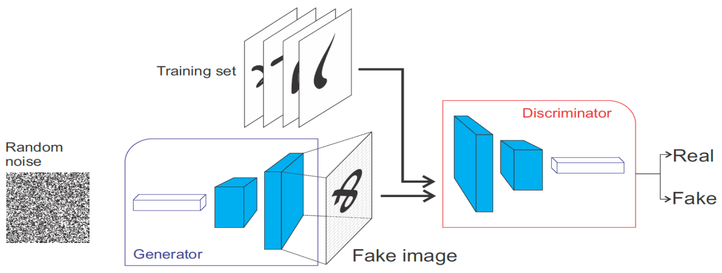


Figure 2.6: GAN Architecture

However, the Vanilla GAN is not the only GAN architecture available. Since researchers have been exploring the space of generative models there are also other types [20, 21, 2, 3, 22] of GAN available with different architectures to solve real-world generative tasks. In the following subsections we will review the other GAN architectures that were used for completing this research.

2.4.2 Conditional Generative Adversarial Networks

Conditional Generative Adversarial Networks (CGANs) [21] can be thought of as an extension of Vanilla GAN. CGANs are the conditional version of generative adversarial nets, which can be constructed by simply feeding the data, y, we wish to condition on to both the generator and discriminator. CGANs are very useful when our data distribution is multi-modal. In an unconditional generative model, there is no control on modes of the data being generated. However, by conditioning the model on additional information it is possible to direct the data generation process. Such conditioning could be based on class labels, on some part of data for in-painting like [23], or even on data from a different modality. The objective function of a conditional two-player minimax game would be as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim\rho_{data}(x)}[\log D(x|y)] + \mathbb{E}_{z\sim\rho_{noise}(z)}[\log(1 - D(G(z|y)))]$$

Here, y is the auxiliary information that we feed both to the generator and discriminator. Figure 2.7 illustrates the architecture of a CGAN.

2.4.3 Wasserstein Generative Adversarial Networks

GANs can be a powerful mechanism to train networks to generate a data distribution. However, the adversarial training where the discriminator effectively generates the error function for the generator also introduces a number of potential problems that effect the learning results. Some of the most impactful of these are:

- **Hard to achieve Nash Equilibrium**: [24] discussed the problem with GAN's gradient-descent-based training procedure. Two models are trained simultaneously to find a Nash equilibrium [25] to a two-player non-cooperative

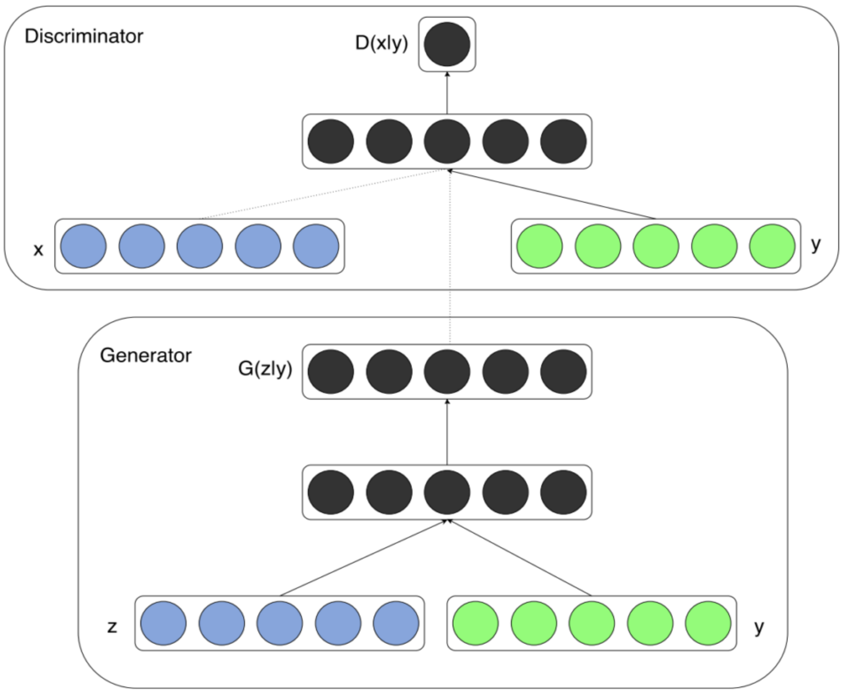


Figure 2.7: Conditional GAN Architecture

game. However, each model updates its cost independently with no respect to another player in the game. Updating the gradient of both models concurrently can not generally guarantee convergence.

- **Vanishing Gradients**: When the discriminator is perfect, the loss function falls to zero and we end up with no gradient to update the loss during learning iterations. Since this gradient of the discriminator is also used to drive the loss of the generator, this implies that no further improvement will occur in the generator, independent of the quality of the function it has learned. Figure 2.8 demonstrates an experiment illustrating that when the discriminator gets better, the gradient vanishes fast.

- **Mode Collapse**: During training, the generator may collapse to a setting where it always produces the same outputs. This is a common failure case for GANs, commonly referred to as Mode Collapse. Even though the generator might be able to trick the corresponding discriminator, it fails to learn to represent the complex real-world data distribution and gets stuck in a small

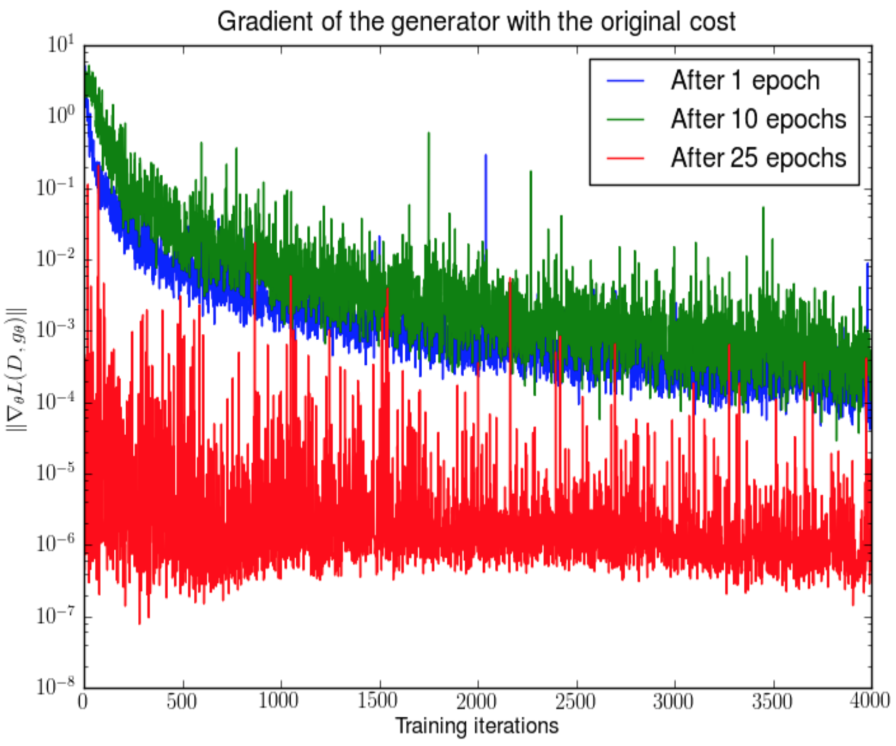


Figure 2.8: First, a DCGAN is trained for 1, 10 and 25 epochs. Then, with the generator fixed, a discriminator is trained from scratch and the gradients are measured with the original cost function. We see that the gradient norms decay quickly (in log scale) by up to 5 orders of magnitude after 4000 discriminator iterations. [1]

space with extremely low variety.

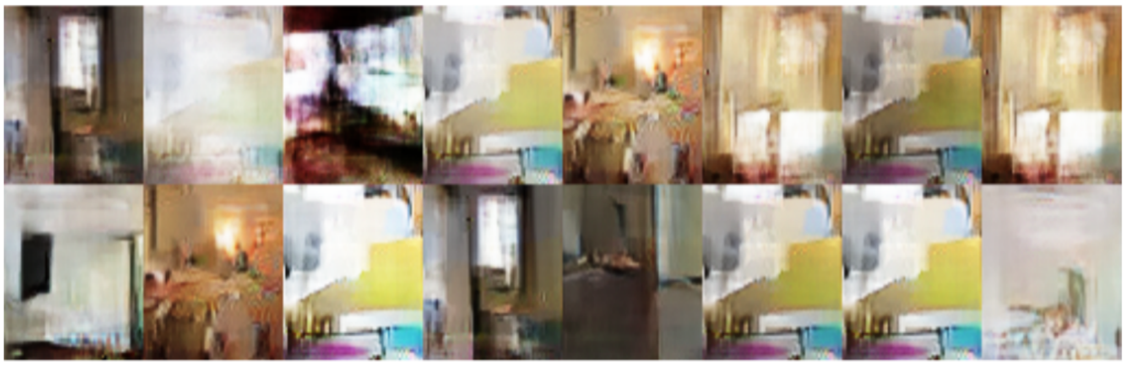


Figure 2.9: A DCGAN model is trained with an MLP network with 4 layers, 512 units and ReLU activation function, configured to lack a strong inductive bias for image generation. The results shows a significant degree of mode collapse. [2]

To alleviate the abovementioned problems, the Wasserstein Generative Adversarial Network[2] or simply WGAN was introduced.

Wasserstein Distance

The Wasserstein distance or earth mover's distance between two distributions is proportional to the minimum amount of work required to convert one distribution into the other. One unit of work is the amount of work necessary to move one unit of weight by one unit of distance. Intuitively, the weight flows from one distribution to the other until they are identical, similar to filling holes with piles of dirt. One distribution acts as a group of holes, while the points of the other distribution are dirt. A distribution can be represented by a set of clusters where each cluster is represented by its mean (or mode), and by the fraction of the distribution that belongs to that cluster. We call such a representation the signature of the distribution. The two signatures can have different sizes, for example, simple distributions have shorter signatures than complex ones.

Formally, let $P = \{(p_1, w_{p_1})....(p_m, w_{p_m})\}$ be first signature with m clusters, where p_i represents a cluster and w_{p_i} represents the weight of the cluster, and $Q = \{(q_1, w_{q_1})....(q_n, w_{q_n})\}$ be the second signature with n clusters. Also, let $\mathbf{D} = [\mathbf{d_{ij}}]$

be the ground distance matrix where d_{ij} is the ground distance between clusters p_i and q_j.

Now, we want to find a flow $\mathbf{F} = [\mathbf{f_{ij}}]$ with f_{ij} the flow between p_i and q_j, that minimizes the overall cost. The flow basically represents the amount of movement that is to be performed between specific clusters p_i and q_j in the process of building Q from P. The overall cost to be minimized represents the total work needed for the transformation and is represented as:

$$WORK(P, Q, \mathbf{F}) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}$$

subject to the following constraints:

$$f_{ij} \geq 0 \quad where \quad 1 \leq i \leq m, 1 \leq j \leq n$$

$$\sum_{j=1}^{n} f_{ij} \leq w_{p_j} \quad where \quad 1 \leq i \leq m$$

$$\sum_{i=1}^{n} f_{ij} \leq w_{q_i} \quad where \quad 1 \leq j \leq n$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} = \min(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_j})$$

Once the transportation problem is solved and we have found the optimal flow $\mathbf{F}$, the earth mover's distance is defined as the work normalized by the total flow:

$$\mathrm{EMD}(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} .$$

In a Wasserstein GAN[2] (WGAN)this distance measure is used as an estimate for the loss function rather than the traditional divergence measures. In their original paper, the authors also point out why using Wasserstein distance has nicer properties when optimized than Jensen–Shannon Divergence. WGANs can learn no matter whether the generator is performing well or not. The figure below shows a similar plot of the value of D(X) for both GAN and WGAN. For GAN (red line), it shows areas with diminishing or exploding gradients. For WGAN (green line), the gradient is smoother everywhere and learns better, even when the generator is not producing good images.

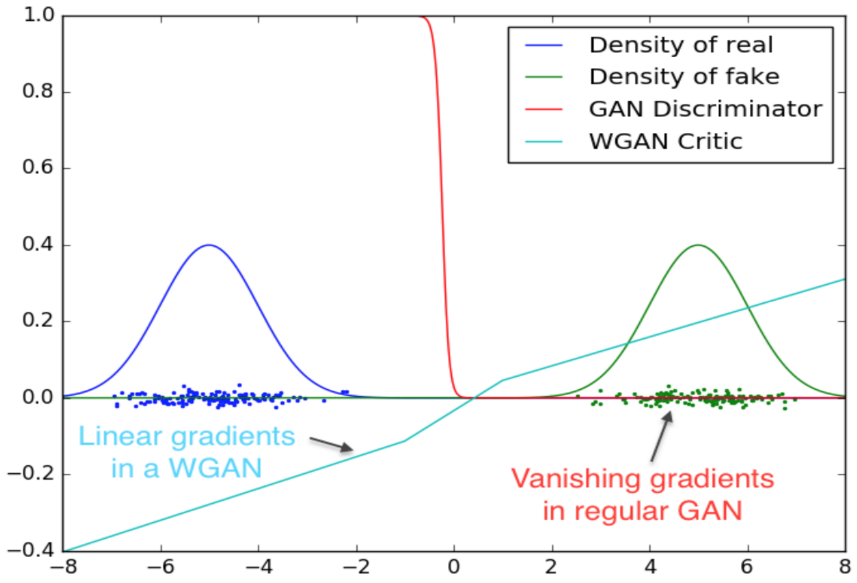


Figure 2.10: Ilustration of vanishing gradients in GAN [2]

WGAN Architecture

In the WGAN architecture, the Wasserstein distance metric is used in the following form, as detailed in [2]:

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

$\Pi(\mathbb{P}_r, \mathbb{P}_g)$denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r\, and\, \mathbb{P}_g$.

Since the exact solution of the equation for the Wasserstein distance is highly intractable, WGAN [2] uses the Kantorovich-Rubinstein duality, and simplifies the calculation to

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_{L \leq 1}} \mathbb{E}_{x \sim \mathbb{P}_x}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

where sup is the least upper bound and f is a 1-Lipschitz function following the constraint

$$\|f(x_1) - f(x_2)\| \leq |x_1 - x_2|$$

To find the 1-Lipschitz constant WGAN uses the discriminator (instead of building another neural network) and outputs a scalar score rather than a probability. Intuitively, this score can be interpreted as how real the input images are. Also, the discriminator is renamed to critic to more clearly indicate this change in function from distinguishing true and generated data points to generating an explicit score for the generator. Hence, the new cost function for WGAN is as follows:

$$\text{For Critic: } \nabla_w \frac{1}{m} \sum_{i=0}^{m} [f(x^{(i)}) - f(G(z^{(i)})]$$

$$\text{For Generator: } \nabla_\theta \frac{1}{m} \sum_{i=0}^{m} f(G(z^{(i)}))$$

To enforce the 1-Lipschitz constraint, WGAN applies a very simple clipping to restrict the maximum weight value in f, i.e. the weights of the critic must be within a certain range controlled by some hyperparameter $\mathbf{c}$.

2.4.4 Wasserstein Generative Adversarial Networks with Gradient Penalty

Challenges Faced by WGAN

The use of weight clipping in WGAN[2] as a method to enforce the Lipschitz constraint on the critic's model to calculate the Wasserstein distance introduces some problems that need to be addressed to make it an effective approach. The model may still produce poor quality data and may not converge, in particular when the hyperparameter **c** is not tuned correctly.

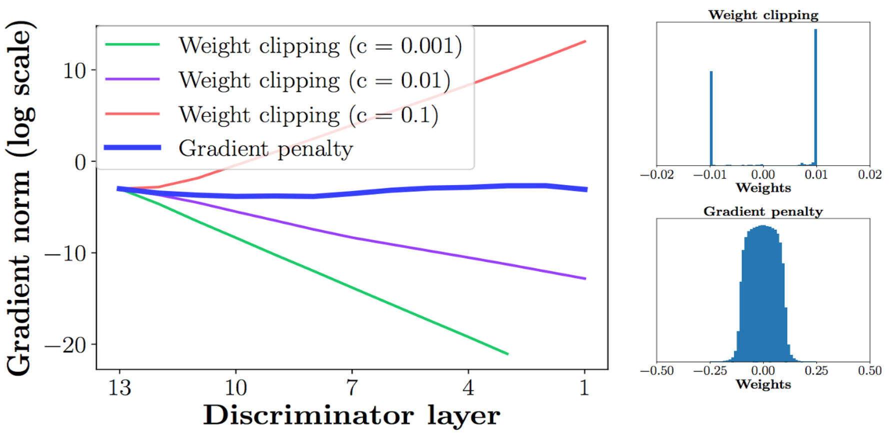


Figure 2.11: Effects of weight clipping hyperparameter **c** on gradients.[3]

The weight clipping behaves as a weight regularization. It reduces the capacity of the model f and specifically limits its capability to model complex functions.

Architecture of WGAN with Gradient Penalty

WGAN with Gradient Penalty[3] (WGAN-GP) was introduced as a variant to WGAN in order to reduce the effects of weight clipping.WGAN-GP uses a gradient penalty instead of the weight clipping to enforce the Lipschitz constraint. WGAN-GP penalizes the model if the gradient norm moves away from its target norm value 1. Hence the new cost function is:

$$L = \underset{\tilde{x} \sim \mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x \sim \mathbb{P}_r}{\mathbb{E}}[D(x)] + \lambda \underset{\hat{x} \sim \mathbb{P}_{\hat{x}}}{\mathbb{E}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]$$

where $\hat{x}$ is sampled from linear interpolations of $\tilde{x}$ and x of the form $\hat{x} = t\tilde{x} + (1-t)x$ with $0 \leq t \leq 1$, with t uniformly sampled between 0 and 1.

λ is set to 10. The point $\tilde{x}$ used to calculate the gradient norm is a point sampled between the $\mathbb{P}_g$ and $\mathbb{P}_r$.

Chapter 3

Related Work

As discussed in the background section, GANs have become increasingly popular and usable and have demonstrated their ability to generate high quality data in domains where data is not easily available. While even the state-of-the-art GAN models[18, 2, 3] have usually been applied to generating images, GANs are not limited only to image data. There have been multiple publications where GANs have also been employed for generating sequential data and have remarkable results[26, 27, 28, 29].

3.1 Missing Data Imputation in Episodic Tasks

When considering non time series data that has missing values, the first step to make the data complete includes methods like zero imputation, mean imputation or mode imputation. [30] provides a comprehensive review on missing data imputation techniques. There are also some GAN based data imputation architectures. GAIN [31]has been developed using the conditional version of GAN. Here the generator is conditional on the original data and a mask (indicating which values are observedor missing), both passed to it as a matrix, outputs a completed version of the original data. This completed data is sent as an input to the discriminator whose goal is to output probabilities not for the whole imputed matrix but for each entry in the matrix. They report the RMSE values which seems to outperform the conventional imputation methods.

On the other hand, the MisGAN [32] architecture employs a completely different approach.Their approach handles the missing data problem in two steps. They first produce a network which can learn the input data distribution no matter if the values are complete or incomplete. Secondly, they also design a separate network to impute the missing values using the data from the first generator. Overall, the MisGAN architecture consists of 4 neural networks: 2 generators and 2 discriminators. One generator is responsible for generating fake data and the second generator is responsible for generating fake masks. In contrast the discriminators classify between fake data/real data and fake mask/real mask, respectively. The whole goal of this architecture is to learn the pattern of missingness and accordingly make a synthetic missing sample which is then compared to the real sample in the discriminator. Hence in the process we anticipate for the generator to learn the true distribution of data.

3.2 Missing Data Imputation in Time Series

The methods described above however work in the non sequential case and we can not directly use those ideas in the context of time series data and therefore have to take a different approach. Since we are specifically concerned with forecasting of time series data with missing/incomplete time steps, we will shift our focus to the corresponding literature.

There have been multiple attempts to impute missing data in time series, too. A very natural choice within the area of neural network models would be to use LSTM networks since they have been designed to handle time series tasks well. RIMP-LSTM [33] introduced the residual short path structure into the LSTM and constructed a Residual Sum Unit (RSU) to fuse the residual information flows. They evaluated their model using the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) on multivariate and univariate time series datasets.

In the healthcare domain [34] proposed a LSTM network to model missing data in clinical time series. They mention multiple methods for dealing with missing data. In particular they consider imputing missing time steps with either zero or to use forward filling. When forward filling a missing value they just use the value from last time step. However, if the value is not available they impute that time step with the median estimated over all measurements in the training data. They also talk about one of the methods which states using missing data indicators along with input data. It turns out their methods performs best when using zero imputation along with missing data indicators. However they use all these preprocessing steps to ultimately perform classification whereas we want to perform forecasting.

[35] introduced recurrent neural networks in the generator as well as in the discriminator. They developed a modified version of GRU [36] which included a time decay vector to control the influence of past information. The goal of the research was to impute missing values in a multivariate time series.

RNNs were also used by [37] in the generator and discriminator. The aim of the research was to generated synthetic but realistic sensor outputs, conditioned on the environment. However they did not considered the case where the time series had missing values and only dealt with complete data.

Hyland and Esteban [38] propose RGAN and RCGAN to produce realistic real-valued multi-dimensional medical time series. Both of these GANs employ LSTM in their generator and discriminator while RCGAN uses Conditional GAN instead of Vanilla GAN to incorporate a condition in the process of data generation. They also describe novel evaluation methods for GANs, where they generate a synthetic labeled training dataset and train a model using this set. Then, they test this model using real data. They repeat the same process using a real training set and a synthetic labeled test set.

ForGAN [39] also presented impressive results by forecasting time series on chaotic datasets. They used the conditional version of GAN to forecast the next time step. The generator was conditional on some time window, then concatenated a noise vector to eventually produce the predicted value x_{t+1}. The discriminator takes x_{t+1} either from the generator or the dataset alongside the corresponding condition window and concatenates x_{t+1} at the end of the condition window. The rest of the network tries to check the validity of this time window.

We can see that a significant amount of literature is concerned with solving the imputation task. However, existing models used for imputation in the context of distributions generally work on episodic data, disregarding the temporal dependencies in the data and thus might yield incorrectly imputed distributions in time series. On the other hand, networks employ currently in time series situations generally use point wise error metrics to check the effectiveness of their model instead of valicating a complete distribution and can thus fail in the scenarios where the x_{t+1} comes from a distribution. So, to the best of our knowledge, this presented work is the first time that a GAN is employed for the forecasting task with missing data in the context of time series data.Here we want to use the missingness as a feature itself when performing forecasting without inherently tampering with the distributions. Our work is analogous to [39, 38], but applied in the context of missing data and thus pursues a different goal. As a result, we need to take a different approach to train and evaluate the performance of our model.

Chapter 4

Approach

In this chapter we will formally discuss the problem formulation and approach taken to solve that problem.

4.1 Problem Formulation

The main problem we are trying to address in this thesis is the time series forecasting problem in the context of missing information. In particular we are interested in predicting the future values of the complete time series from incomplete information about past time steps. In a time series, (multivariate or univariate) if it contains missing values the forecasting task becomes difficult. However to address the problem we can take into consideration the inherent structure in the missingness as a feature and predict future data. In the following we define what missing time series data looks like.

Missing time series can be of two basic types:

1. **Missing features**: Some features are missing at some time steps, however this can only occur in multivariate time series. This can be illustrated as

follows:

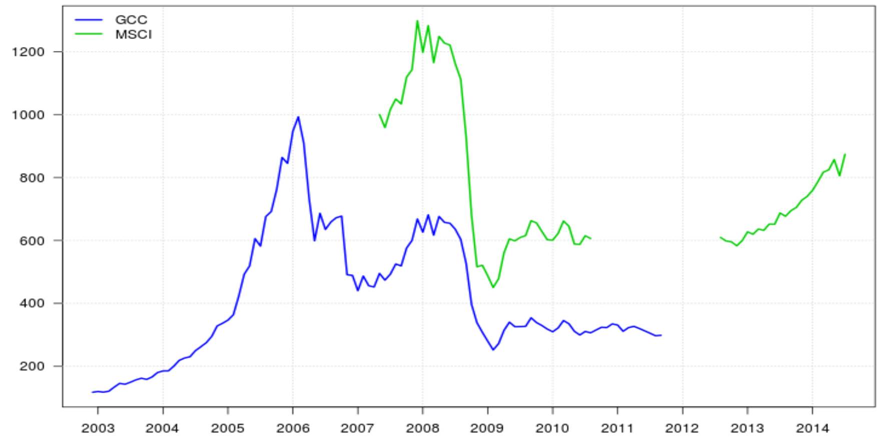


Figure 4.1: Time series data with missing features

2. **Missing time steps**: In this case the complete time steps are missing for all features. This can occur in both univariate time series and multivariate time series. This can be illustrated as follows:

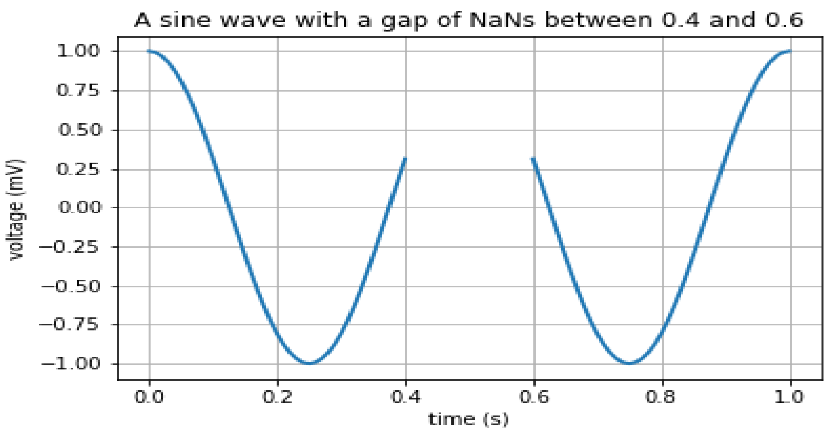


Figure 4.2: Time series data with missing time steps

We denote a time series $\mathbf{X} = \{\mathrm{x}_1, \mathrm{x}_2, ..., \mathrm{x}_T\}$ as a sequence of T observations. The t^{th} observation $\mathrm{x}_t \in \mathbb{R}^D$ consists of D features $\{x_t^1, x_t^2, ..., x_t^D\}$. $\mathbf{X}$ is a univariate time series if $D = 1$. Since the time series might have missing data we also introduce a concept of mask vector $\mathbf{m}_t$ (denoted in time series form as $\mathbf{M}$) which simply indicates whether a data point in x_t was observed or not.

$$\mathbf{m}_t^d = \begin{cases} 0 & \text{if } x_t^d \text{ is not observed} \\ 1 & \text{otherwise} \end{cases}$$

Table 4.1 and Table 4.2 illustrates the above definitions.

$\mathbf{X} =$

nan	nan	-0.32	0.64	0.11	nan
0.64	0.11	nan	-0.78	0.39	0.2
-0.78	0.39	0.2	0.10	nan	0.12

Table 4.1: The table above illustrates a multivariate time series representation with $T = 6$ time steps and $D = 3$ features. *nan* indicates missing values at x_t^d

$\mathbf{M} =$

0	0	1	1	1	0
1	1	0	1	1	1
1	1	1	1	0	1

Table 4.2: The table above illustrates a mask vector representation for **X**

To combine the mask and data we also define a masking operation f_κ that fills the missing values with some constant value κ.

$$f_\kappa(\mathbf{X}, \mathbf{M}) = \mathbf{X} \odot \mathbf{M} + \kappa\overline{\mathbf{M}}$$

In this thesis, we study a general setting for forecasting time series data with missing values. Given a time series **X** our goal would be to predict the future. We denote this future by **y**. In general **y** can be a scalar or a vector. However we want to model the full probability distribution of possible **y** values and not only asingle value. The main reason for this is that we are often not only interested in the most likely forecast value but in the entire range of potential forecasts, including their likelihoods. This, however, can only be achieved when being able to forecast all elements of the data distribution.

In real world applications the existence of missing data is usually the result of a process which includes communications failures, different data collection speeds for different features, or sensor failures due to environmental effects. As a result, the missingness of the data is often not independently random but follows a pattern distribution. To account for this, it becomes important to not only forecast data values but also missingness pattern distributions. It is therefore important to also predict the missingness patterns in order to be able to make effective forecasts. To be able to make these predictions, we can for training purposes split the time series $\mathbf{X}$ in two parts: history and prediction. We consider time steps until x_{T-1} as history and denote it as $\mathbf{H}$ and the x_T time step will serve as our prediction denoted as $\mathbf{Y}$.

History ($\mathbf{H}$) =

nan	nan	-0.32	0.64	0.11
0.64	0.11	nan	-0.78	0.39
-0.78	0.39	0.2	0.10	nan

Prediction ($\mathbf{Y}$) =

nan
0.2
0.12

Table 4.3: $\mathbf{X}$ transformed into history and prediction time series.

In the same way we also transform the mask time series $\mathbf{M}$ into a masked history $\mathbf{H}_M$ and a masked prediction $\mathbf{Y}_M$

Masked history ($\mathbf{H}_M$) =

0	0	1	1	1
1	1	0	1	1
1	1	1	1	0

Masked prediction ($\mathbf{Y}_M$) =

0
1
1

Table 4.4: $\mathbf{M}$ transformed into masked history and masked prediction time series.

4.2 Proposed Method

In this section we will discuss the key steps taken to solve the problem described previously. Since we want to model the full probability distribution of $\mathbf{y}$ values we will build a generative system to achieve it.

4.2.1 Architecture

The overall system for forecasting is made up of a simple conditional GAN where both the generator and the discriminator are conditioned on the history. The goal of the generator is to generate prediction $\mathbf{y}$ realistic enough to fool the discriminator and the discriminator tries to distinguish between true and fake predictions.

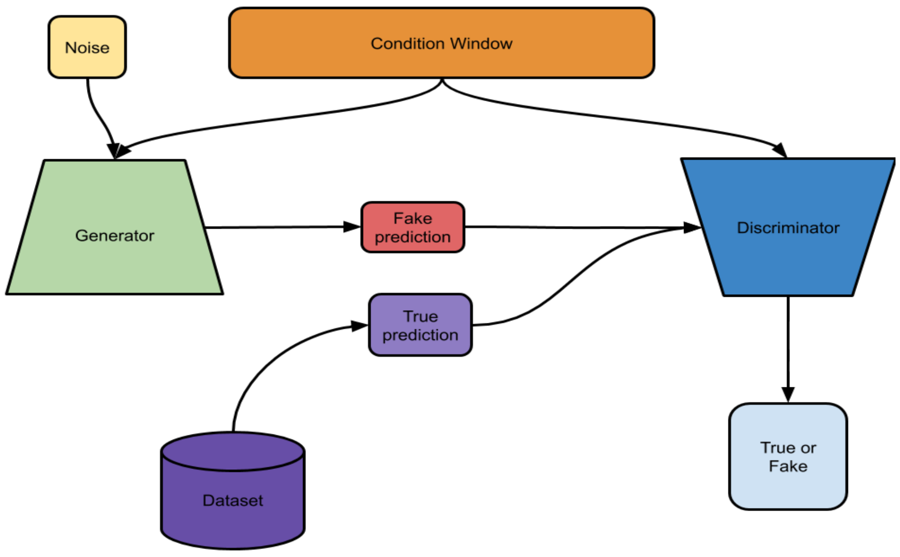


Figure 4.3: Overall forecasting architecture of the system.

Before we discuss the detailed architecture of the neural network we want to introduce an *Auxiliary GAN* whose main goal is to learn the distribution of missingness. Since the masks in the incomplete data are fully observable, we can estimate their distribution using a standard GAN.

4.2.2 Auxiliary GAN

Since we are dealing with modeling the Missing at random(MAR) case we can say that, there does exist an inherent pattern in the missing data and if so we can use a GAN architecture to learn this distribution of patterns. Since we can observe the mask from our time series we can train a network to learn the distribution of missing values [32] occurring in the series and in turn we will use this information to mask the forecasted data which goes as an input to the discriminator. As we know that data generated by the previously described data forecasting generator will always be complete while real data follows the missingness distribution, the discriminator could easily figure out that it is fake data because the true data never contained complete values. Hence, in order to encourage the discriminator to learn to recognize meaningful data distributions we mask the data generated. Therefore there is no need to develop a customized approach to impute missing values, rather we can directly use the standard GAN architectures to forecast the data.

4.2.3 Training Scheme

Now, that we have defined the structure of our overall system we can now look at the steps necessary for training the system.

First, we train the Auxiliary GAN to learn the mask distribution. The architecture for the Auxiliary GAN is the same as shown in Figure 4.3 where the condition window is the corresponding mask vectors $\mathbf{M}$ of the time series $\mathbf{X}$ and the goal would be to predict M_{T+1}. The loss function for this Auxiliary GAN is defined as follows:

$$L_{aux}(D_{aux}, G_{aux}) = \mathbb{E}_{\text{M}_{T+1}}[D_{aux}(\text{M}_{T+1}|\mathbf{M})] - \mathbb{E}_z[D_{aux}(G_{aux}(z|\mathbf{M}))]$$

The loss function defined above follows the Wasserstein GAN formulation [2].

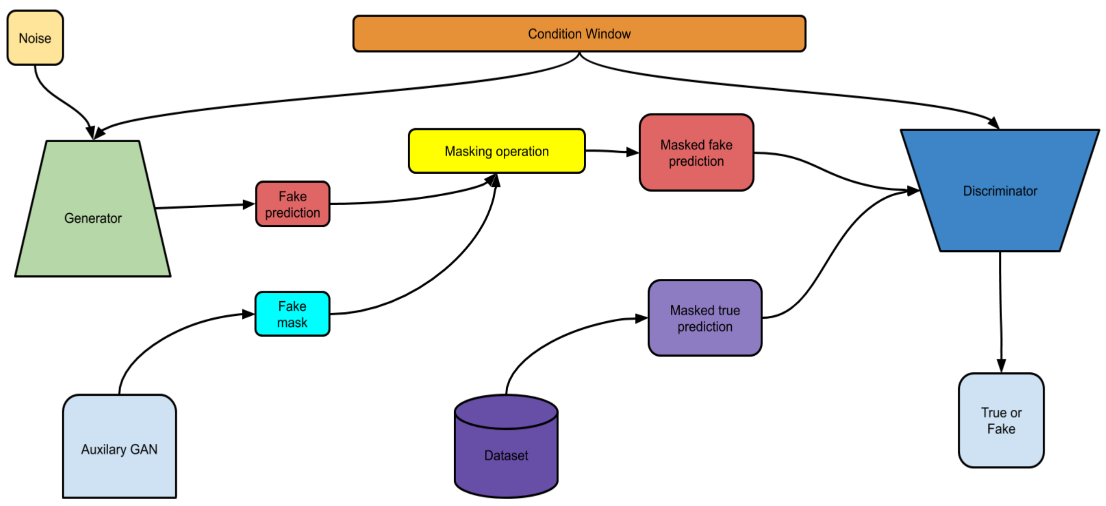


Figure 4.4: Overall operation of the forecasting system.

D_{aux} and G_{aux} represent the Auxiliary GAN's discriminator and generator, respectively. We optimize the generator and the discriminator with the following objective:

$$\min_{G_{aux}} \max_{D_{aux}} L_{aux}(D_{aux}, G_{aux})$$

After the Auxiliary GAN has completed the training we can use it to generate synthetic masks for the forecasting GAN. Once the data forecasting GAN has generated the x_{T+1} prediction we use the masking operation (refer to Figure 4.4) to mask the generated time step and use it as input to the discriminator. The loss function for this forecasting GAN is defined as follows:

$$L_{data}(D_{data}, G_{data}) = \mathbb{E}_{\mathrm{x}_{T+1}}[D_{data}(\mathrm{x}_{T+1}|\mathbf{X})] - \mathbb{E}_{z}[D_{data}(G_{data}(z|\mathbf{X}))]$$

D_{data} and G_{data} represent the data forecasting GAN's discriminator and generator, respectively. We optimize the generator and the discriminator with the following objective:

$$\min_{G_{data}} \max_{D_{data}} L_{data}(D_{data}, G_{data})$$

4.2.4 Network Specifications

Here, we discuss the skeletal structure for both generator and discriminator. Note that we can use the same specifications for both the Auxiliary GAN and the data forecasting GAN.

Generator

The architecture is pretty intuitive. We start with a temporal convolution layer, followed by a LSTM layer to generate a hidden representation for the conditional window. We can consider this as a pre-processing step followed by concatenation of noise input and hidden representation followed by a dense layer and finally the prediction. The dense layer applied to the hidden representation with noise forms the main generator for the forecasting distribution. (See Figure 4.5 for illustration.)

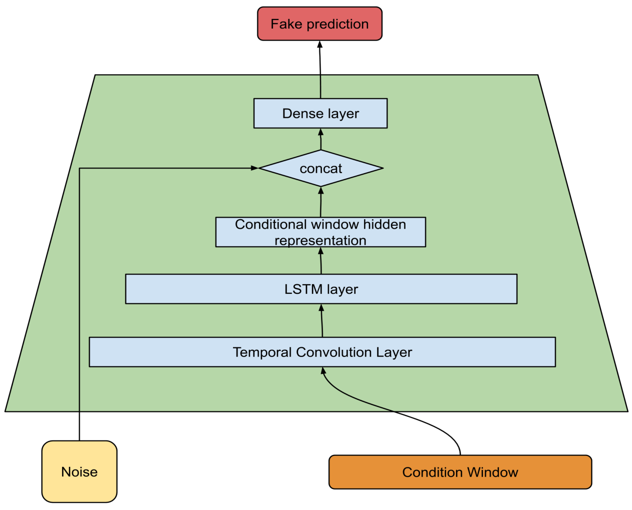

Fake prediction
Dense layer
concat
Conditional window hidden representation
LSTM layer
Temporal Convolution Layer
Noise
Condition Window

Figure 4.5: Generator Architecture.

Discriminator

The architecture for discriminator is analogous expect that we have two dense layers after the concatenation step (See Figure 4.6 for illustration). The intuition for this is to make the discriminator a little bit deeper than the generator as suggested in [40]. We can also do this with respect to the temporal convolution filters and LSTM units and we will discuss more about these hyper-parameters in the experiments section.

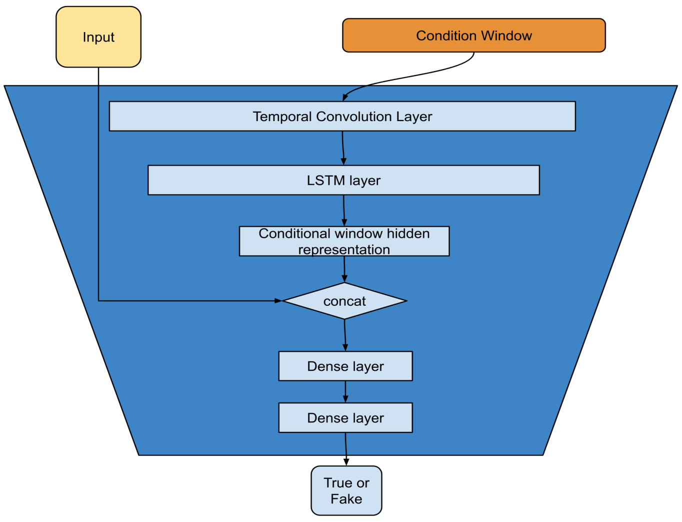


Figure 4.6: Discriminator Architecture.

Chapter 5

Experiments and Discussions

This chapter deals with the practical aspect of training and optimizing the Auxiliary GAN and data forecasting GAN.

5.1 Datasets

To test the forecasting capability of the architecture we test it with two famous chaotic datasets that have also been used in other time series forecasting work.

5.1.1 Lorenz Dataset

In 1963 E. N. Lorenz developed a three parameter family of three-dimensional ordinary differential equations as a model for atmospheric convection with applications to weather forecasting. This model appeared, when integrated numerically on a computer, to have extremely complicated solutions. The equations are as follows:

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = \rho x - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

the variable x measures the rate of convective overturning, the variable y measures the horizontal temperature variation, and the variable z measures the vertical temperature variation. The three parameters σ, ρ and b are respectively proportional to the Prandtl number, the Rayleigh number, and some physical proportions of the region under consideration. The most interesting characteristic of this equations is the emergence of chaos [41, 42] for certain values of σ, ρ and b. To construct the dataset we chose $x_0 = y_0 = z_0 = 1.0$ and created 10000 samples each with a gaussian noise of mean 0 and standard deviation 3 to create a distribution. The constant parameters σ, ρ and b have values 7, 70, and 2.667 respectively. We generate 101 time steps where the last time step serves as the x_{T+1} prediction.

5.1.2 Mackey-Glass Dataset

The time delay differential equations suggested by Mackey and Glass [43] has been used widely for forecasting tasks. These equations have originally been explored as models for physiological control processes and illustrate chaotic behavior and in particular dynamic bifurcations. The equation is defined as follows:

$$\frac{dx}{dt} = \beta\frac{x_\tau}{1 + x_\tau^n} - \gamma x; \quad \gamma, \beta, n > 0$$

where β, γ, τ and n are real numbers. x_τ represents the value of variable x at time $(t - \tau)$ and depending on these parameters the equation displays a range of periodic and chaotic dynamics. We set β to 0.2, γ to 0.1, n to 10 and τ to 17 and generate 10000 samples. We generate 201 time steps where the last time step serves as the x_{T+1} prediction.

5.1.3 Mask dataset

For learning the mask distribution we create synthetic mask datasets using the following steps:

1. Randomly sample a positive integer s from an uniform distribution. s should be less the number of time steps.
2. Create an array a with values 0.5, 1, 2. This will serve as missing patterns meaning if we have data available for s time steps then data is missing for $a[i] * s$ time steps where i can be either 0, 1 or 2. $a[i] * s$ should also be less than number of time steps in order to avoid out of bounds errors.
3. Then we create an empty mask array and move over it until we reach the total number of time steps. We set mask values to 1 for s time steps and to 0 for $a[i] * s$.
4. We repeat this process for all the samples in the dataset thereby creating missing value masks for each and every sample.

We create a mask dataset with the same number of time steps as the number of time steps for each dataset and the last time step serves as the prediction $\textsc{m}_{T+1}$.

5.2 Evaluation Metric

Since we are aiming to estimate the distributions of future time steps, the natural selection for an evaluation metric would be to use Kullback-Leibler (KL) divergence to assess the generated values. However some questions arise regarding the symmetric properties of Kullback-Leibler divergence and hence we choose Jensen-Shannon (JS) divergence to assess the quality of predictions since it provides a normalized and symmetrical version of the KL divergence. To define the JS divergence we need to define KL divergence first.

5.2.1 Kullback-Leibler Divergence

The KL divergence score intuitively quantifies how much one probability distribution differs from another probability distribution. The KL divergence between two distributions Q and P is often stated using the following notation:

$$KL(P||Q)$$

where the || operator indicates "divergence" or P's divergence from Q. It can be calculated as follows:

$$D_{KL}(P||Q) = -\sum_{x \in X} P(x) \log \left(\frac{P(x)}{Q(x)} \right)$$

The intuition for the KL divergence score is that when the probability for an event from P is large, but the probability for the same event in Q is small, there is a large divergence. When the probability from P is small and the probability from Q is large, there is also a large divergence, but not as large as the first case. Now, that we have defined the KL Divergence we can look at JS divergence.

5.2.2 Jensen-Shannon Divergence

JS divergence uses the KL divergence to calculate a normalized score that is symmetric. This means that the divergence of P from Q is the same as of Q from P, or stated formally:

$$JS(P||Q) = JS(Q||P)$$

It can be calculated as follows:

$$JS(P||Q) = 1/2 * KL(P||M) + 1/2 * KL(Q||M)$$

where M is calculated as follows:

$$M = 1/2 * (P + Q)$$

5.3 Results

Now, since we have already defined our evaluation metric we can visualize and calculate the divergence values. Since we want our architecture to forecast all possible values given the history we randomly select one sample from the test set and generate a number of x_{T+1}values and calculate the JS divergence between the generated values and the true prediction distribution. To evaluate the architecture we also train the neural network with complete data which in-turn gives us an upper bound in terms of performance. Then, we again train the network with incomplete data to show the network's efficiency.

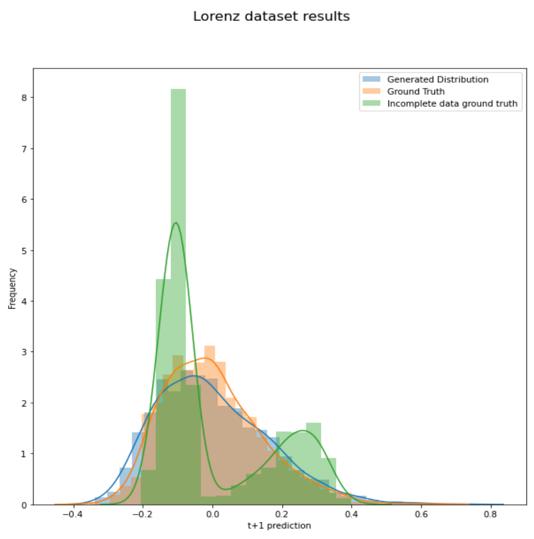
5.3.1 Lorenz Dataset Forecasting

On the Lorenz dataset we test the network with multiple settings such as percentage of missing values and no of features missing and report the corresponding JS divergence values.

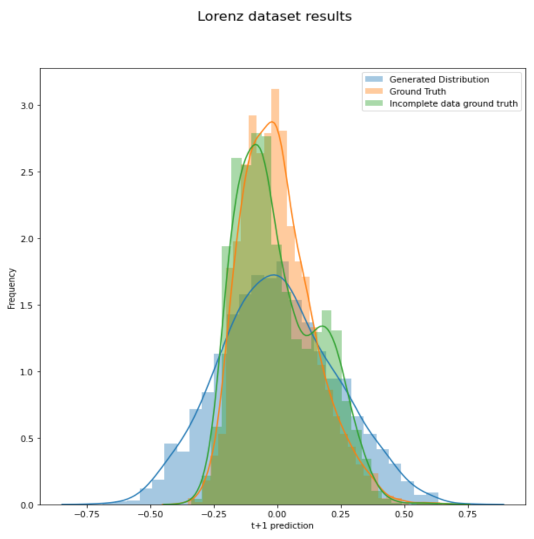
Metric: JS Divergence	No of features missing		
	0	1	2
0% missing data	0.4663	-	-
20%-30% missing data	-	0.6223	0.625
40%-50% missing data	-	0.6338	0.6462

Table 5.1: JS divergence values with different settings on Lorenz dataset

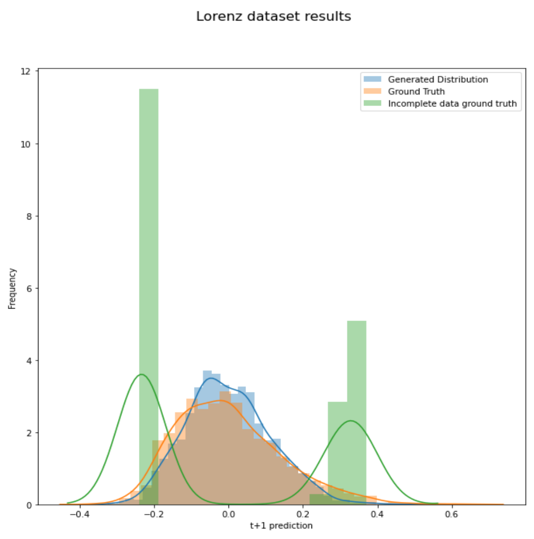
For visualizing the results we used principal component analysis to reduce the x_{T+1} dimension to 1D for effective plotting. in particular, in the following we visualize the ground truth data distribution for the prediction step as well as the generated data distribution for this step. In addition we show the distribution of the missing data values for the corresponding time step to illustrate the systematic relation between the missing data and the history resulting from the missingness pattern distribution. The results for different settings can be observed in the following Figure 5.1. From Figure 5.1 we can see how the network is able to forecast the true data distribution given missing data. We also notice that the discriminator does not collapse to a single mode in the missing data distribution and does approximate the true data distribution so that the generator can in turn learn efficiently.



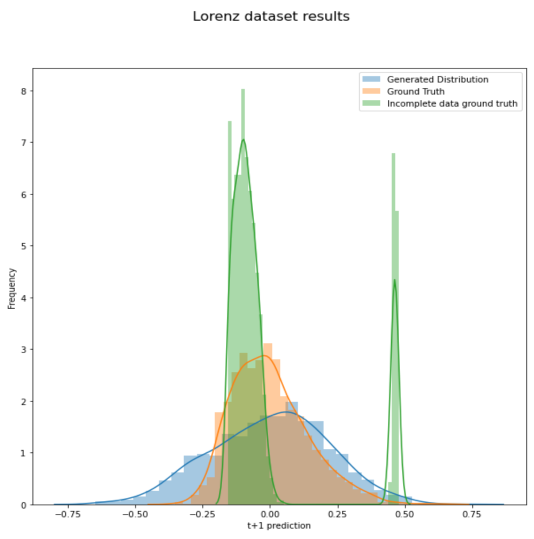
(a) 20% incomplete time steps in 1 feature



(b) 50% incomplete time steps in 1 feature



(c) 20% incomplete time steps in 2 features



(d) 50% incomplete time steps in 2 features

Figure 5.1: Visualization of Lorenz data set results

5.3.2 Mackey-Glass Dataset Forecasting

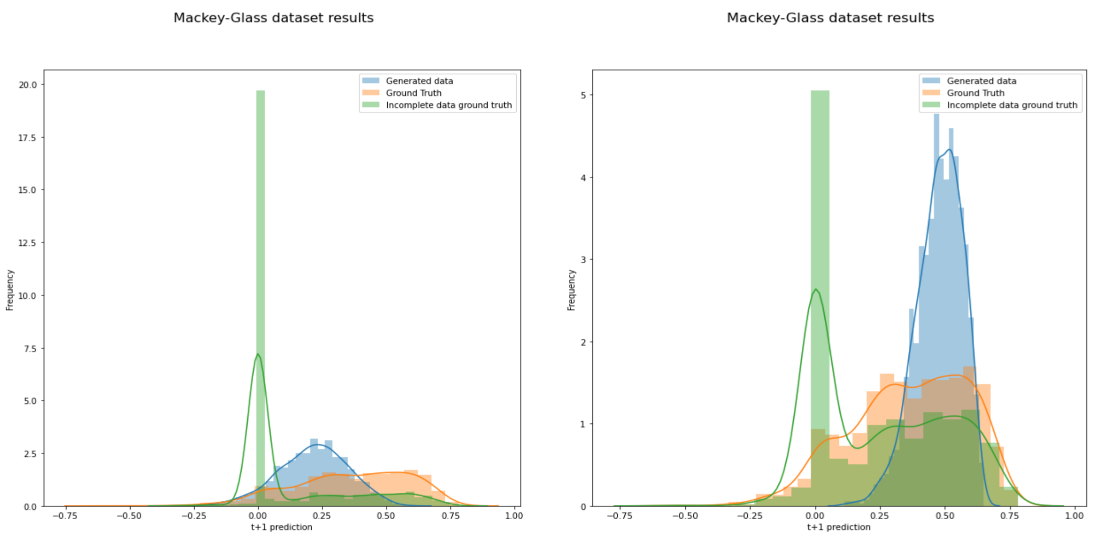
For the univariate case we used the Mackey-Glass dataset. In this case, since the data has only one feature we cannot make more than 1 feature missing, hence we check the robustness of our model with the percentage of missingness. We also set upper bounds by training the model with complete data and then compare how

Metric	0% missing data	20 - 30% missing data	40 - 50% missing data
JS Divergence	0.0309	0.0862	0.1393

Table 5.2: JS divergence values with different settings on Mackey-Glass dataset

the model performs with incomplete data.

Figure 5.2 illustrates data generated for different settings in the Mackey-Glass data set. As in the case of the Lorenz dataset, we again show the ground truth distribution, the forecasting distribution, and the distribution of the missing values. Again, the latter shows that missing data items are not independently distributed but depend on the previous time series.



(a) 20% incomplete time steps (b) 50% incomplete time steps

Figure 5.2: Visualization of Mackey-Glass data set results

In the above figure we can see that when 40-50% data is missing the model is not able to generate the complete distribution of the ground truth. However, we can clearly see it does not collapse in the incomplete data distribution.

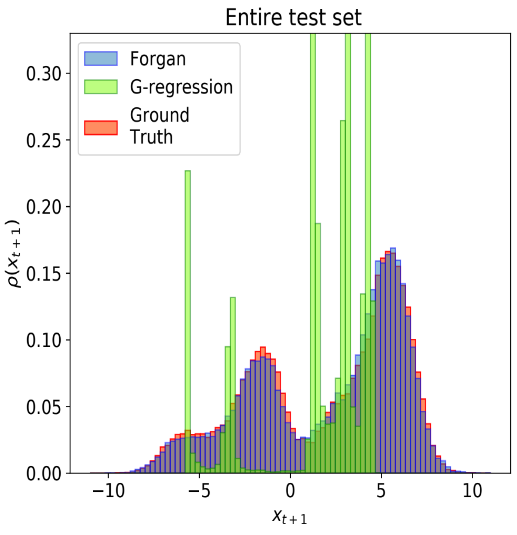
5.3.3 Comparison with Other Methods

We also compared our model with the state-of-the-art ForGAN model [39]. Since the state-of-the-art method does not solve the missing data problem we use their results as an upper bound to assess our model and also visualize the results. [39] reported KL divergence values in their work hence we also adapt to the same performance metric and report our KL divergence scores on missing data. [39] has also made their synthetically generated dataset publicly available which makes it easier for us to compare.

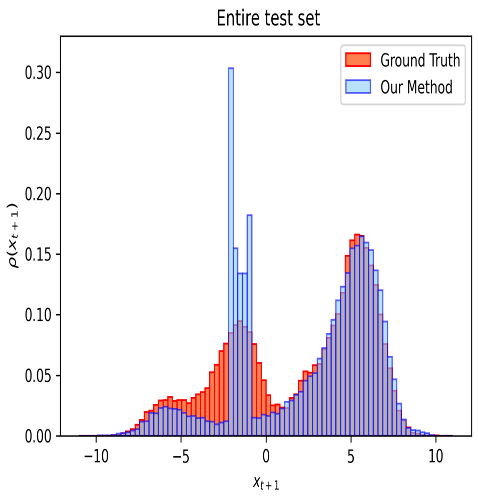
Metric	KL divergence	
State-of-the-art	Complete data	0.0167
Our Method	0 % missing data (untuned hyperparameters)	0.0523
	20 - 30% missing data	0.0958
	40 - 50% missing data	0.1028

Table 5.3: KL divergence values with different settings on the state-of-the-art dataset

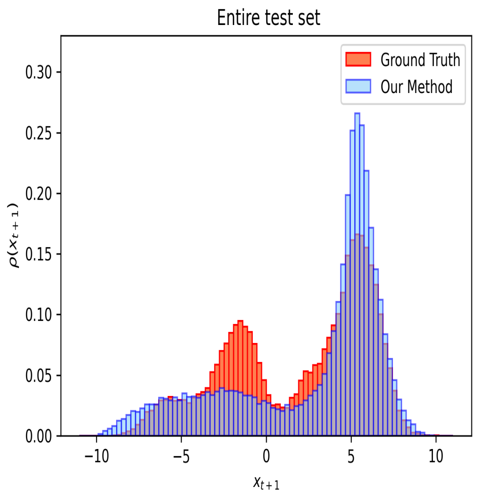
The KL Divergence values shown in Table 5.3 show that our method reaches results that are close to the KLD values with complete data using the state-of-the-art ForGAN model. Also, Figure 5.3 illustrates the results on the dataset provided by [39]. Again, here we can observe that our method does not collapse in the incomplete data distribution and learns the true data distribution as illustrated in Figure 5.3.



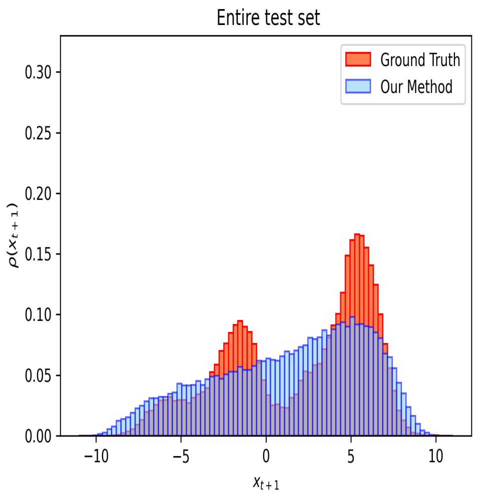
(a) State-of-the-art



(b) 0% incomplete time steps with untuned hyperparameters



(c) 20 - 30% incomplete time steps



(d) 40 - 50% incomplete time steps

Figure 5.3: Comparison with state-of-the-art method

Chapter 6

Conclusion and Future Work

This thesis presents a novel generative model for time series forecasting with missing data. This model directly predicts the forecasting distribution without first performing imputation and, to our knowledge, is the first model to make complete distribution forecasts in the context of time series data with missing information. In this work we dealt with two problems. Firstly it attempts to deal with missing data and second it aims at effectively producing probabilistic forecasts for time series data with missing values. Our experiments show promising results. The use of an Auxiliary GAN to learn the mask distributions helps significantly in the data generation. The Auxiliary GAN effectively prevents the data generation GAN to fall into the incomplete data distribution and allows the network to generate truely probabilistic forecasts. It also allows us to by-pass the preprocessing step of imputing the missing values that is present in general purpose imputation techniques. Moreover, directly predicting the distribution in the context of missing data rather than first imputing values and then making predictions based on single imputed values also avoids the problem of overcommitment in the imputation phase which could lead to incorrect predictions especially in the context of systems that exhibit bifurcations where imputed values would miss complete "branches", i.e. modes of the forecasting distribution. After comparing our results with a state-of-art architecture that operates strictly on complete data, we see that our method is able to perform probabilistic forecasting despite having missing data with only small degradation in performance.

However, there is always room for improvement. In the experiments we have performed manual search for tuning of hyper-parameters and for the future work we want to employ some automated methods such as using genetic algorithm, bayesian optimization or random search methods for tuning the hyper-parameters. We also want to explore multistep probabilistic forecasting using this method in the future.

Bibliography

[1] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan, 2017.

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.

[4] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control, 5th Edition.* Wiley, 2015.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), March 1994.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[7] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, April 1998.

[8] Alex Krizhevsky, Ilya Sutskever, and GeoffreyE.Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[10] John Cristian Borges Gamboa. Deep learning for time-series analysis. *CoRR*, abs/1701.01887, 2017.

[11] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[12] Dimitri Palaz, M. Magimai-Doss, and Ronan Collobert. Analysis of cnn-based speech recognition system using raw speech as input. In *INTERSPEECH*, 2015.

[13] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3995–4001. AAAI Press, 2015.

[14] Davide Figo, Pedro C. Diniz, Diogo R. Ferreira, and João M. P. Cardoso. Preprocessing techniques for context recognition from accelerometer data. *Personal and Ubiquitous Computing*, 14(7):645–662, 2010.

[15] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, Mar 2019.

[16] How much data is created on the internet each day? https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/.

[17] How much data is generated each day? https://www.weforum.org/agenda/2019/04/how-much-data-is-generated-each-day-cf4bddf29f/.

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[19] Hamed Alqahtani, Manolya Kavakli-Thorne, and Gulshan Kumar. Applications of generative adversarial networks (gans): An updated review. *Archives of Computational Methods in Engineering*, 2019.

[20] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric, 2015.

[21] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[22] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[23] Ian Goodfellow, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Multi-prediction deep boltzmann machines. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 548–556. Curran Associates, Inc., 2013.

[24] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[25] John Forbes Nash Jr. *Non-cooperative games*. PhD thesis, Princeton University, May 1950.

[26] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient, 2016.

[27] Yizhe Zhang, Zhe Gan, and L. Carin. Generating text via adversarial training. NIPS Workshop, 2016.

[28] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial learning for neural dialogue generation, 2017.

[29] Ofir Press, Amir Bar, Ben Bogin, Jonathan Berant, and Lior Wolf. Language generation with recurrent generative adversarial networks without pre-training, 2017.

[30] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, 2010.

[31] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. Gain: Missing data imputation using generative adversarial nets, 2018.

[32] Steven Cheng-Xian Li, Bo Jiang, and Benjamin Marlin. Misgan: Learning from incomplete data with generative adversarial networks, 2019.

[33] Lifeng Shen, Qianli Ma, and Sen Li. End-to-end time series imputation via residual short paths. In Jun Zhu and Ichiro Takeuchi, editors, *Proceedings of Machine Learning Research*, volume 95, pages 248–263. PMLR, 14–16 Nov 2018.

[34] Zachary C. Lipton, David C. Kale, and Randall Wetzel. Modeling missing data in clinical time series with rnns, 2016.

[35] Yonghong Luo, Xiangrui Cai, Ying ZHANG, Jun Xu, and Yuan xiaojie. Multivariate time series imputation with generative adversarial networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1596–1607. Curran Associates, Inc., 2018.

[36] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[37] H. Arnelid, E. L. Zec, and N. Mohammadiha. Recurrent conditional generative adversarial networks for autonomous driving sensor modelling. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 1613–1618, 2019.

[38] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans, 2017.

[39] A. Koochali, P. Schichtel, A. Dengel, and S. Ahmed. Probabilistic forecasting of sensory data with generative adversarial networks – forgan. *IEEE Access*, 7:63868–63880, 2019.

[40] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks, 12 2016.

[41] Colin Sparrow. *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*. Springer, New York, NY, 2012.

[42] Stephen H. Kellert. *In the Wake of Chaos: UNPREDICTABLE ORDER IN DYNAMICAL SYSTEMS*. University of Chicago Press, 1993.

[43] MC Mackey and L Glass. Oscillation and chaos in physiological control systems. *Science*, 197(4300):287–289, 1977.