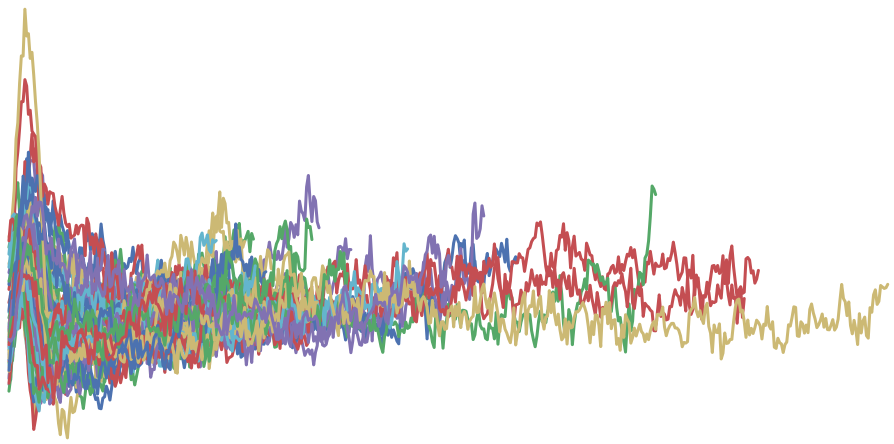
CHALMERS
UNIVERSITY OF TECHNOLOGY



Sensor Modelling with Recurrent Conditional GANs

Recurrent Conditional Generative Adversarial Networks for Generating Artificial Real-Valued Time Series

Master's thesis in Complex Adaptive Systems

HENRIK ARNELID

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

TIFX05, Master's thesis 2018

Sensor Modelling with Recurrent Conditional GANs

Recurrent Conditional Generative Adversarial Networks for Generating Artificial Real-Valued Time Series

HENRIK ARNELID

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2018

Sensor Modelling with Recurrent Conditional GANs
Recurrent Conditional Generative Adversarial Networks for Generating Artificial Real-Valued Time Series
Henrik Arnelid

Supervisors: Edvin Listo Zec and Nasser Mohammadiha, Zenuity
Examiner: Mats Granath, Department of Physics

Master's Thesis 2018
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Graphical visualization of 100 synthetic time series from the generative network trained in the RCGAN framework.

Typeset in LaTeX
Gothenburg, Sweden 2018

Abstract

Autonomous vehicles rely on many sensors in order for the vehicles to perceive their surroundings. Consequently it is important with safety verification of the sensors which typically is done by collecting data in many different scenarios which is time consuming and expensive. For this reason, autonomous driving software companies are interested in virtual verification where the scenarios are simulated. In this thesis we have developed and used the *Recurrent Conditional Generative Adversarial Network* (RCGAN) in order to model the longitudinal error of sensors. The RCGAN is a modification of the original generative adversarial network (GAN) framework which makes use of recurrent neural networks and conditioning the networks on auxiliary information. These changes allows the model to learn and be able to generate realistic real-valued multi-dimensional time series.

Keywords: Generative Model, Autonomous Vehicle, Recurrent Neural Networks, Generative Adversarial Networks, Sensor Modelling, Time Series

Acknowledgements

First of all I would like to express my gratitude to Edvin Listo Zec and the Data Analysis Team at Zenuity for giving me the opportunity to write my master thesis at this inspiring company and within a topic that I have a big interest in. Thank you for the discussions and giving me feedback continuously throughout the project and showing much interest in my work. Furthermore, I would like to thank Zenuity for accommodating me in their productive work environment and providing me with all the data and tools that was needed for my thesis. Finally, I would like to extend my thanks to Edvin, Erik, Majid and Magnus for all the ping-pong matches as well as the daily push-up challenge.

Contents

1
Introduction

The past few years has shown an ever increasing popularity in using machine learning algorithms in many different research areas. These types of algorithms has many applications within the automotive industry such as advanced driving assistance systems (ADAS), autonomous driving (AD) and manufacturing optimization [1]. A common challenge in these areas is to process data and extract important information from it, which *deep learning* methods excel at. Deep learning involves machine learning algorithms that utilize large neural networks with many hidden layers for solving tasks such as classification and predictions [2].

Zenuity is a company that work with developing software for autonomous vehicles which involves software for ADAS, highly automated driving (HAD) and AD. The vehicles are equipped with many sensors which produce data that is processed in order for the vehicles to perceive their surroundings. An important aspect of this is to evaluate the quality and performance of sensors. In order to evaluate this, data has to be gathered from the sensors in many different scenarios and conditions which is both expensive and time consuming.

1.1 Background

In order to evaluate the safety of the autonomous vehicles with a certain confidence more than 450 million of kilometers has to be driven in order to provide a good statistic on a low fatality rate [3]. This is a very time consuming and expensive task which cannot be done in reality. Thus, Zenuity and other manufacturers resort to virtual verification where models of the environment and sensors are made to match reality.

One aspect of these simulation environments which is considered in this master thesis is to model the error of the sensors as accurately to the real world as possible. These errors do usually not follow any known distribution and have an unknown correlation through time which makes it difficult to model them. However, state-of-the-art generative models has shown promising results in the generation of time series [4, 5]. One such class of models are *generative adversarial networks* (GANs) which are specialized at generating data which is similar to the real training data [6]. There are other approaches suitable for generating real-valued time series data which involves hidden markov models or state-based models [7, 8]. However, the focus will be on neural network based models in this thesis.

GANs have been very successful in various image generation tasks, such as image-to-image translation [9, 10, 11, 12], image super-resolution [13, 14] and in text-to-image synthesis [15, 16, 17], making the network architecture well suited to learn two dimensional distributions well. This suggest that they are a viable choice for learning any distribution given an appropriate network structure and framework. In this thesis we are interested in sequential data distributions. GANs have previously been used for sequential data generation, but these typically focus on discrete outputs such in language processing [18]. The amount of research for generating continuous real-valued time series is limited where only two works are known to the author [19, 20].

1.2 Problem description

In this thesis we are interested in creating a generative model that is able to recreate realistic time series that a sensor provided by Volvo Car Corporation produce. The host car is equipped with the sensor as well as a *LIDAR* that provide the ground truth reading. The setup of how the errors are recorded can be seen in figure 1.1 below. The error of the sensor is dependent on properties of the specific scenario between the host and target, such properties are relative position, angle, velocity, etc.

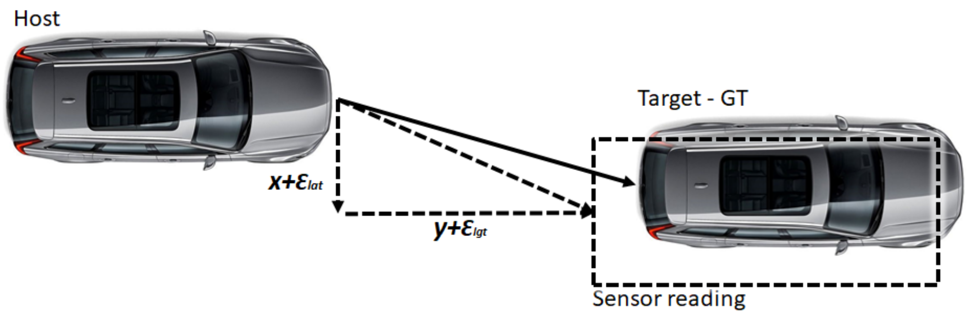


Figure 1.1: The image represents the setting of how the sensor errors are obtained. The dashed square around at target vehicle is the sensor reading which is mismatched to the ground truth (GT) of the target vehicle due to the error of the sensor. These errors ε_{lgt} and ε_{lat} is the main focus in this thesis.

The aim of this thesis is to create a model which with training is able to generate novel samples of real time series of the errors seen in the figure above. The lateral and longitudinal error of the sensor is defined as:

$$\begin{aligned} \varepsilon_{lgt} &= Y_{sensor} - Y_{lidar} \\ \varepsilon_{lat} &= X_{sensor} - X_{lidar} \end{aligned} \tag{1.1}$$

In this project different extensions of generative adversarial networks will be trained using data from sensors provided by Volvo Cars. In order to model the temporal aspect of the data we will use *recurrent neural networks*, more specific Long Short-Term Memory (LSTM) cells [21], which gives the network a ”memory” so that previous inputs influence newer inputs. Additional modifications have to be made to the original GAN in order for it to be a suitable model for this given problem. These will be covered in the next chapter.

The generative model has to be able to handle time series of arbitrary length as the length of any set of scenarios can differ substantially from one another. Furthermore, the framework should be versatile enough so that it can be tuned for time series with different amount of noise. Some signals may be cleaner than other and the network should still be able to generate novel samples to the real data.

One of the major desired characteristics of the generative model is that the output should not be deterministic, i.e. same output from two separate runs with the same input sequence. The reason for this being that the sensors have different intrinsic noise as well as noise coming from the surroundings which influence the signal. A deterministic model would not be true to the noise aspect of the signal. However, the output should still follow the trend of the real time series but output a unique time series each time the same sequence of features is given to the model in order to represent the noise fairly.

1.3 Thesis outline

This thesis is composed of five main chapters. It will start with the **Theory** chapter where Generative Adversarial Networks and extensions and modifications thereof are presented. Next chapter in order is the **Methods** chapter where the implementation, model evaluation and work process is described. In the **Results** section plots of synthetic time series is presented as well as histograms where the real and synthetic distributions can be compared. Furthermore, in the **Discussion** chapter the results and explanations thereof is given. Finally, the **Conclusions** chapter summarizes the main outcomes of this thesis together with reflections over future work.

2 Theory

The generative model that was developed in this thesis is called a *recurrent conditional generative adversarial network* (RCGAN) which follows the general framework of the original GAN presented by [6]. A number of changes as well as extensions have been implemented in order to create the generative model that is able to output sequences of real valued data subjected to a conditional input. The foundation of GANs and the extensions and modifications made in this thesis are presented in this chapter. Lastly, the RCGAN framework is presented.

2.1 Generative Adversarial Networks

Generative adversarial networks (GANs) are a neural network framework aimed to generate new realistic samples given the distribution $p(\boldsymbol{x})$ of the training data [6]. This kind of neural network architecture has shown good results in generating realistic samples in many different applications [9, 13, 15], where tasks involving images are predominant. In the framework there are two different neural networks, a generator (G) and a discriminator (D) network, which have conflicting objectives.

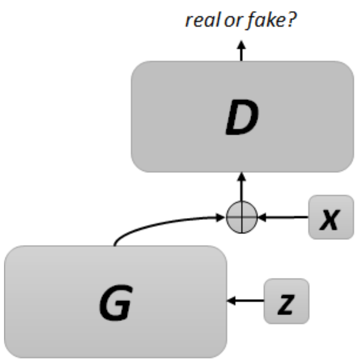


Figure 2.1: A schematical view of the GAN framework. The generator takes a randomly sampled latent vector $\boldsymbol{z}$ as input and outputs a sample that is judged by the discriminator which takes *either* a real, $\boldsymbol{x}$, or a fake sample, $G(\boldsymbol{z})$, as input at a given instance. The output from D is the probability whether the sample was real or not.

The discriminator network is being fed both real data and synthetic data produced by the generator where the aim to estimate the probability if the sample is real instead of coming from G. The generative model, G, is trained to capture the data distribution by trying to maximize the error of D. Backpropagation is applied to both networks making D better at telling data apart while G becomes better at synthesizing realistic samples. In mathematical terms the GAN is trained to solve the following optimization problem:

$$\min_G \max_D \quad \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))], \tag{2.1}$$

where $\boldsymbol{x}$ is real data from the distribution $\mathbb{P}_r$ and $\boldsymbol{z}$ are samples from a prior distribution $p(\boldsymbol{z})$, which typically is drawn from $U(0,1)$ or $\mathcal{N}(0,1)$. That is, the loss function is maximized for the discriminator and minimized for the generator. The global optima for the optimization is when $\mathbb{P}_r = \mathbb{P}_g$, in other words when the generators output distribution $\mathbb{P}_G$ matches the real data distribution. The training of the networks is split into two steps where the discriminator is trained in one step and the generator in the other step.

Starting with the discriminator, samples from the real data $\boldsymbol{x}$ and from the latent space $\boldsymbol{z}$ are drawn where $\boldsymbol{z}$ is fed through G. Both $\boldsymbol{x}$ and $G(\boldsymbol{z})$ are fed through D, yielding $D(\boldsymbol{x})$ and $D(G(\boldsymbol{z}))$. Next the loss $\mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ is calculated and the gradients are calculated and followed by updating the weights of D.

When the generator is trained samples from the latent space $\boldsymbol{z}$ are drawn and fed through G whose output is fed through D. The output from the discriminator is then used in the loss $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$ and the gradients are calculated followed by an update the weights of G. It is possible to maximize the loss $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[\log D(G(\boldsymbol{z}))]$ instead which provide stronger gradients in early learning while maintaing the same fixed point of D and G [6].

In the training scheme used in the GANs the generator never sees the true data directly, only indirectly through the gradient from the discriminator. This avoids having features in the input directly copied to the parameters of the network which in turn may gain statistical advantages compared with other models [6].

GANs using the standard implementation are famously known to have a unstable training process where a too good discriminator or generator can cause the model to collapse and not output anything meaningful [22]. There are also issues with vanishing gradients if the discriminator saturates. There is an abundance of previous research that has worked on improving the training of the original GANs [22, 23, 24, 25]. A brief introduction to some of the extensions will be given in the next section.

2.1.1 Different types of generative adversarial nets

The common objective for many variations of the GANs is to minimize the distance between two distributions, namely the real data and the synthetic data. There are a number of metrics suitable for this purpose, an example of one family of such metrics is the *f-divergence* metrics. These are incorporated in different GAN extension implementations where the optimization objective is modified. The structure of the neural net can remain the same for the different types of GANs with the exception of the activation function on the final output neuron of the discriminator. The change that is primarily made is to the loss function of the networks making it easy to implement several different extensions in order to find a suitable one for the problem at hand.

One of the more popular extensions is the *Wasserstein GANs*, abbreviated WGANs, which provides a modification to the network such that one tries to optimize with respect to the *Earth-Mover* (EM) distance instead. This approach yields cleaner gradients in all parts of the space unlike the original GANs which can experience vanishing gradients in some parts of the space. Furthermore, the discriminator has to be a K-Lipschitz function for some

K which is done by clamping the weights to a fixed box, for example $W_D = [-0.1, 0.1]$, where W_D are the weights of the discriminator [23]. The optimization objective for the WGAN is:

$$\min_G \max_D \ \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}[D(G(\boldsymbol{z}))] \tag{2.2}$$

In the WGAN the activation of the last output neuron in the discriminator is removed and is replaced by a linear activation. Enforcing the K-Lipschitz constraint by clipping the weights is not an ideal approach and may lead to issues with the training process. The authors of the WGAN have left options for enforcing the constraint for further investigation.

To improve the downsides with the gradient clipping, [24] proposed the WGAN with gradient penalty (*WGAN-GP*) instead of clipping the gradients. The modified training objective is given by:

$$\mathbb{L} = \mathbb{E}_{\boldsymbol{z}\sim p(z)}[D(G(\boldsymbol{z}))] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[(||\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})||_2 - 1)^2\right] \tag{2.3}$$

Where $\hat{\boldsymbol{x}} = \varepsilon\boldsymbol{x} + (1-\varepsilon)G(\boldsymbol{z})$, ε is drawn from U(0,1) and λ is the penalty coefficient.

The final extension that was considered is the *f-GAN* which is a framework where any f-divergence can be used for training the GANs. As with the previous extensions the structure of the networks remains the same where the big changes lie in the output activation of D and the loss functions. So, let $T(x) = g_f(v(x))$, where $v(x)$ is the linear output from D and $g_f(\cdot)$ is the output activation function. Then the training objective is defined by (2.4) below.

$$\min_G \max_D \ \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[T_\omega(\boldsymbol{x})] - \mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_g}\left[f^*(T_\omega(\hat{\boldsymbol{x}}))\right] \tag{2.4}$$

By selecting specific $g_f(\cdot)$ and $f^*(\cdot)$ functions it is possible to recover the desired f divergences. Recommended choices of g_f and f^* is presented in table 2.1 below [25]. Four different f divergences has been tested in order to evaluate if they can stabilize or improve the learning. The four different divergences are *Total variation*, *Pearson*[1] χ^2, *Neyman* χ^2 and *Squared Hellinger*.

Table 2.1: Recommended final layer activation functions and conjugate functions for variation of the f-GAN optimization objective [25].

Name	Output activation g_f	Conjugate $f^*(t)$
Total variation	$\frac{1}{2}\tanh(v)$	t
Pearson χ^2	v	$\frac{1}{4}t^2 + t$
Neyman χ^2	$1 - \exp(v)$	$2 - 2\sqrt{1-t}$
Squared Hellinger	$1 - \exp(v)$	$\frac{t}{1-t}$

2.2 Conditional GANs

When the data is dependent on a set of features it is possible for the GANs to model this by implementing a straight forward extension. The distribution $p(\boldsymbol{x}|\boldsymbol{c})$ can be learned

[1]There is another GAN modification called *Least Square GANs* [26] whose minimization objective also is the Pearson χ^2 divergence through a different set of functions. Due to this it was left out in this thesis.

by the network simply by adding the conditional feature vector $\boldsymbol{c}$ as input to both G and D [6]. The loss function for the original implementation for both the generator and discriminator would then take the following form:

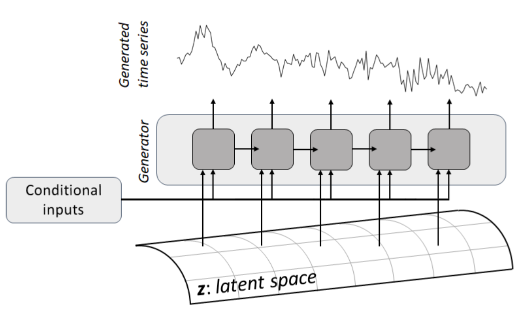
$$\begin{aligned} \mathbb{L}_D &= \log D\left[\boldsymbol{x}, \boldsymbol{c}\right] + \log\left(1 - D\left[G\left[\boldsymbol{z}, \boldsymbol{c}\right]; \boldsymbol{c}\right]\right) \\ \mathbb{L}_G &= \log D\left[G\left[\boldsymbol{z}, \boldsymbol{c}\right]; \boldsymbol{c}\right] \end{aligned} \tag{2.5}$$

Where $\boldsymbol{x}$ are real samples, $\boldsymbol{c}$ is the conditional input and $\boldsymbol{z}$ are random samples from a latent space. Here we wish to maximize both $\mathbb{L}_D$ and $\mathbb{L}_G$. As previously mentioned, in the original implementation of GANs one tries to minimize $\log(1 - D(G(\boldsymbol{z})))$ for the generator, however, by using the form above in (2.5) a much stronger gradient is obtained early in the learning process [6]. The modifications are done analogously for WGAN and f-GAN as well, where $\boldsymbol{c}$ is added to both loss functions for the generator and discriminator in the same manner as in (2.5).

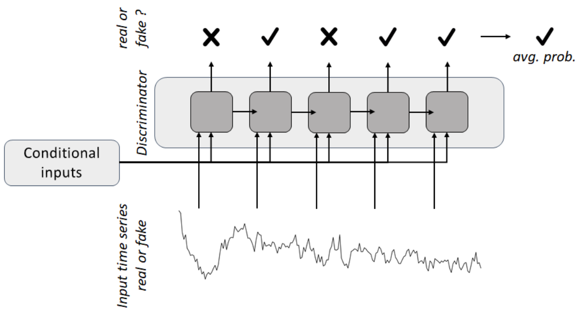
2.3 Recurrent Conditional GANs

The model that has been developed during this thesis is based upon the original GAN framework with some modifications. Firstly, both the generator and discriminator are recurrent neural networks instead of multilayer perceptrons or convolutional nets. The second modification is to include the conditional input to both networks so that they can make informed decisions/predictions given the current input. This conditional input can be in the form of one-hot encoding or other labels, but in this thesis it is a continuous signal of different properties of the car-tracking scenario mentioned in the problem description section of the report.

In figure 2.2 below we can see schematical images of the architecture of the networks. In the same manner as with the original GAN the generator takes a latent vector $\boldsymbol{z}$ as input sampled from some known distribution. Furthermore, we feed the discriminator either a real or fake time series where the network outputs a probability at each time frame whether the sample is real or fake. All predictions for each time frame are then used in the loss function. If a GAN modification such as f-GAN is used the interpretation of the output is different as the sigmoid activation is no longer used, thus the output is not a probability. Finally, both networks also take a conditional input with a set of continuous features at each time frame.



(a) Structure of the generator. It takes input from a latent space as well as a conditional input at each time frame.

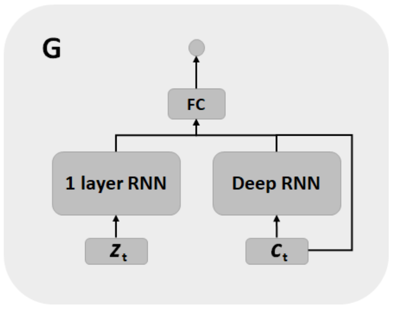


(b) Structure of the discriminator. It takes either a real or fake time series together with the conditional input as input at each time frame.

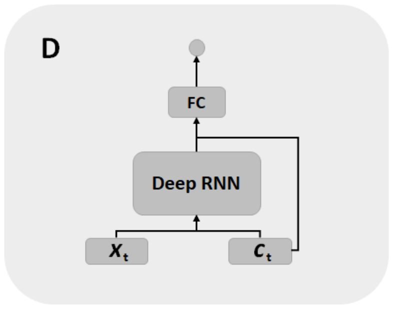
Figure 2.2: The two images show the architecture of the framework. Both RNNs in the generator and discriminator respectively are LSTM based.

The internal structure of the networks presented above and how the data flows in them is non-trivial. Starting with the generator, we have a ”noise” network and a ”context” network within it, see figure 2.3a. It was found that when the noise and the conditional input is given to the same deep RNN, similar to the structure of in figure 2.3b, the noise would become almost non-existent after a few thousand iterations. By isolating the noise to its own RNN it is possible to control the amount that is applied to the output in much greater detail through varying the noise distribution, network size and the size of the latent vector $\boldsymbol{z}$. The context network gives the generator a memory of the sequence of features that are given as the conditional input. Lastly, a skip connection was added for $\boldsymbol{c_t}$ so that information about the input is not completely lost and distorted in the deep RNN. Skip connections are when the input is given to additional layers in the network and not only to the first hidden layer, this can be seen in figure 2.3 where $\boldsymbol{c_t}$ is skip connected to a deeper layer of the network. This have for instance been used by [4] who saw improved learning when training LSTM based networks.

The discriminator has a more simple structure where the data point(s) from either a real or fake time series, $\boldsymbol{x_t}$, is being concatenated with the corresponding features at each time frame and is given to a deep RNN. Skip connection was also used in this network for the same reason. By adding the skip connection for $\boldsymbol{c_t}$ to the fully connected layer for both networks a more stable and faster learning was obtained during the training.



(a) The generator takes a sample, $\boldsymbol{z}$, from the latent space which is passed to a single layered RNN meanwhile the conditional input is fed to a multilayered RNN and each time frame t. The output from both RNNs and the skip-connected conditional input are all concatenated and fed into a fully connected layer and the final output is a linear activation of the inputs from the fully connected layer.



(b) The discriminator takes a sample $\boldsymbol{x}_t$ from a time series, real or fake, together with the conditional input at each time frame t. These two inputs are concatenated and passed to a multilayered RNN whose output goes into a fully connected layer together with the skip-connected conditional input. Finally there is one output neuron for the prediction of the specific time frame.

Figure 2.3: The two figures show the internal structure of the generator (left) and discriminator (right).

The loss function in (2.5) is modified slightly when transferred to the time series application. The new loss, (2.6), is obtained by simply taking the mean of the conditional loss across the time dimension where n is the length of the given time series. The modification to the loss is done analogously for WGANs and f-GANs as well.

$$\begin{aligned} \mathbb{L}_D &= \frac{1}{n}\sum_{i=1}^{n}\left\{\log\left(D\left[\boldsymbol{x}_i, \boldsymbol{c}_i\right]\right) - \log\left(D\left[G\left[\boldsymbol{z}_i, \boldsymbol{c}_i\right]; \boldsymbol{c}_i\right]\right)\right\} \\ \mathbb{L}_G &= \frac{1}{n}\sum_{i=1}^{n}\log\left(D\left[G\left[\boldsymbol{z}_i, \boldsymbol{c}_i\right]; \boldsymbol{c}_i\right]\right) \end{aligned} \tag{2.6}$$

3
Methods

The main work has been divided into three parts, namely develop and implement the models, construct a suitable training scheme and finally evaluate the trained models. After these three steps the focus was shifted to hyperparameter tuning in order to improve the results of the models, which is the step where a majority of the time was spent during this thesis.

3.1 Implementation of the networks

The models in this thesis has been implemented in Python 3.6 using Tensorflow 1.5. Suitable network sizes that were used are 3 layers and 64 LSTM nodes in the deep RNNs followed by the fully-connected layer with 64 neurons for both the generator and discriminator. The 1 layered RNN in the generator also had 64 LSTM nodes whose input $\boldsymbol{z} = [z_1, z_2, ..., z_{32}]^\top$ are drawn from $\mathcal{N}(0, 1)$. Although the main focus has been to implement the RCGAN during this thesis an additional network types and training schemes have been implemented in order to see how the performance of the RCGAN stacks up against these. A supervised training scheme was implemented for the generator by simply using its output in a mean absolute error loss function instead of pitching the generator against the discriminator network in the GAN setting. An additional network type that was implemented was a recurrent mixture density network (R-MDN) [27], which is a type of network that output the parameters for a Gaussian mixture model (GMM) at each time frame. See Appendix A for further explanation of the MDN and R-MDN. The data points were sampled straight from the output GMMs at each time frame, thus not including correlation between successive samples.

When training neural networks it is common to utilize GPU acceleration to decrease time spent on training the models. However, here all models were trained using the CPU as this was nearly ten times faster compared with GPU, which there are several reasons for. One of them being that recurrent neural network execute in a sequential fashion which limits the pace that the GPU can work in. The second reason is that batch training has not been a viable option due to varying length of the sequences in the data set, which further limit the benefit of training on a GPU.

3.2 Network training

The data set used for training contains 8,639 unique time series which are longer than 50 time frames. In total there are 1,828,577 time frames in the training data. The validation set contains 1,365 unique time series with a total of 202,848 time frames.

Two different strategies have been used when training the networks in this thesis. The first one involves feeding an entire time series, which can be of different lengths and calculating all the outputs. This output is then used in the loss function and the mean of it is taken in order to not have a skewed loss depending on the length of the time series. Then the gradients are applied and the weights are updated. This method works well if the data set contain mostly shorter time series as issues with vanishing gradients are not as pronounced. Furthermore, when training on longer sequences the loss function can get diluted as individual time frames does not impact the average of all the predictions from the discriminator significantly, see (2.6), for these longer sequences.

If the time series in the training set are longer or contain finer details a different strategy has been used. Here the time series are being split into smaller series which are fed to network sequentially. In between succeeding partial time series the state of the recurrent nodes has to be kept when the next part is fed so that the ”memory” of the entire time series is maintained when the next set of predictions are calculated. After each split sequence is fed, the loss function and gradients are calculated and the weights are updated. Thus, the weights are updated much more often in this training scheme.

Before any training takes place the weights are initialized, which is done differently for the LSTMs and the fully connected layers. The weights for the LSTMs are initialized randomly where samples are drawn from a truncated normal distribution with mean 0 and standard deviation 0.1. Meanwhile, the weights of the fully connected layers are initialized according to the Xavier initialization [28].

3.3 Evaluation of the models

Evaluating generative models is a difficult task as it is, for many generative models, computationally intractable to calculate the likelihood [29]. Usually one tries to generate visually appealing results that agree with the true distribution. However, it is not a viable method in the long run as it would be unfeasible for an operator to manually inspect each produced sample of the generative model. If the model is able to learn the distribution of the true data implicitly it is considered successful. By evaluating a suitable distance metric between the real and fake data distribution we can get an assessment of the trained model. However, these will not capture the temporal aspect of data, only the overall distribution. An estimation of how the model is capturing the temporal aspect can be obtained by calculating the first difference between sequential time frames for both real and fake data and then calculating the distance metric between those distributions.

3.3.1 Kullback–Leibler divergence

The Kullback-Leibler (KL) divergence is the foundation of a useful metric which has been used to score the models in this thesis. The KL divergence is defined as:

$$K(P||Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right) \tag{3.1}$$

for two discrete probability distributions P and Q. There are a number of drawbacks with this metric. Firstly it is asymmetric and more important if there are samples in P that has a zero probability in Q these will result in an infinite value of the metric, which is undesired.

3.3.2 Jensen-Shannon Divergence

The Jensen-Shannon divergence is obtained combining the KL-divergence cleverly in order to create a metric that is symmetric and without the possibility of approaching infinity The Jensen-Shannon divergence is defined as:

$$JSD(P||Q) = \frac{1}{2}K(P||M) + \frac{1}{2}K(Q||M), \tag{3.2}$$

where $M = \frac{1}{2}(P+Q)$ and $K(\cdot||\cdot)$ is the Kullback–Leibler divergence. The Jensen-Shannon divergence is limited to the interval $[0, \log 2]$ when using the natural logarithm, where 0 is obtained if P and Q are identical distributions and $\log 2$ is obtained if said distributions are completely disjoint. The square root of the JSD is called the Jensen-Shannon distance (JSd) which is the metric that has primarily used to set a score to the trained models. This metric cannot solely be used to evaluate the models as it does not consider the temporal aspect of the data, further evaluation is done by visual inspection of the synthetic time series.

4

Results

The results from the different models as well as the different training objectives is presented in this chapter. These are shown by plotting the same 100 time series as in figure 4.1, which is the real counterpart of the upcoming plots of the synthetic time series. Furthermore, histograms of the errors and first difference of the errors in the entire validation set are plotted together with the generated errors and first difference of these for a comparison between the distributions. Finally, the JSd between the real and synthetic data distributions are given for all models.

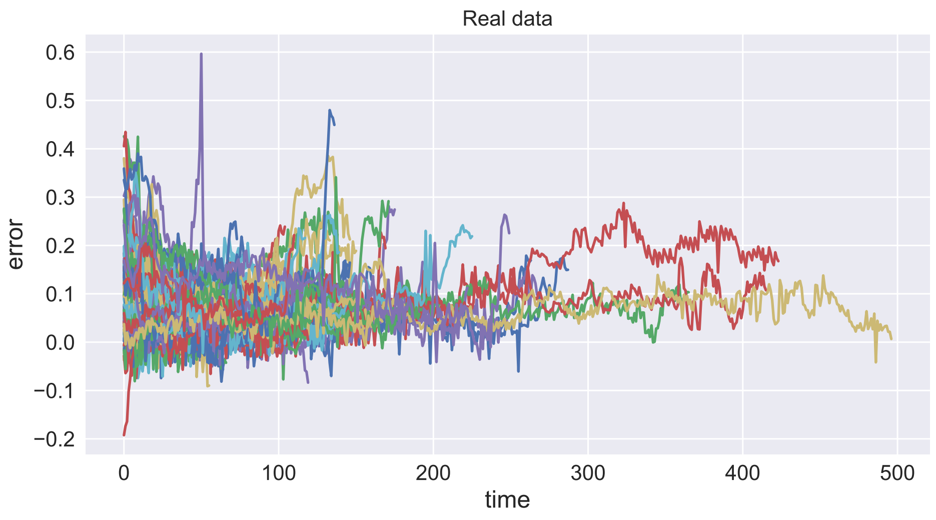


Figure 4.1: The figure depicts the longitudinal error of 100 time series from the validation data set. All errors have been rescaled due to confidentiality reasons.

All models has been trained using ADAM [30] with $\beta_1 = 0.5, \beta_2 = 0.999$ and a learning rate of 10^{-5}. Each GAN was trained for four epochs using a batch size of one time series. The parameters β_1 and β_2 control the decay rates of moving averages of gradients which are used in ADAM. Furthermore, a low dropout rate of 5% was applied to the discriminator network. All models presented have been trained on reproducing the longitudinal error, see figure 1.1. Lastly, the conditional feature vector contains a number of features such as range to the target, relative angle between the vehicles and relative speed. All features were normalized to zero mean and unit variance.

4.1 Summary of performance

The JSd scores of the three different models is summarized in table 4.1 below. The RCGAN that is proposed in this thesis is the best performing model by a significant margin. When the generator in the RCGAN is trained in a supervised fashion by minimizing a distance metric instead a much higher JSd is obtained, i.e. the synthetic data distribution does not match the real data distribution. Lastly, the R-MDN is able to capture the data distribution fairly well but with too much noise. Due to the noisier time series more values are likely to be covered near the real error for the time series in the validation set causing the JSd score of the error to become better whereas the first difference becomes worse. Plots of synthetic data with the same feature vector time series as in figure 4.1 is given for each model for a visual comparison in the upcoming sections.

Table 4.1: The performance against the validation data set for the three different models that were looked at in this thesis.

Network type	JSd	1^{st} diff JSd
RNN (MAE)	0.377 ± 0.0006	0.508 ± 0.0004
R-MDN	0.153 ± 0.0008	0.158 ± 0.0016
RCGAN	$\mathbf{0.076 \pm 0.001}$	$\mathbf{0.095 \pm 0.0009}$

4.2 Recurrent Conditional GANs

In Figure 4.9 the same set of 100 synthetic time series has been plotted. We see that there is an initial transient at the beginning of every time series before it settles. The characteristics is fairly similar to what is seen in figure 4.1 with similar noise levels and magnitude of the errors. There are however a few exceptions where the initial transient is similar for many different time series and not as diverse as with the real data. There is also a limited amount of unique features in the generated data such as sudden peaks or drops.

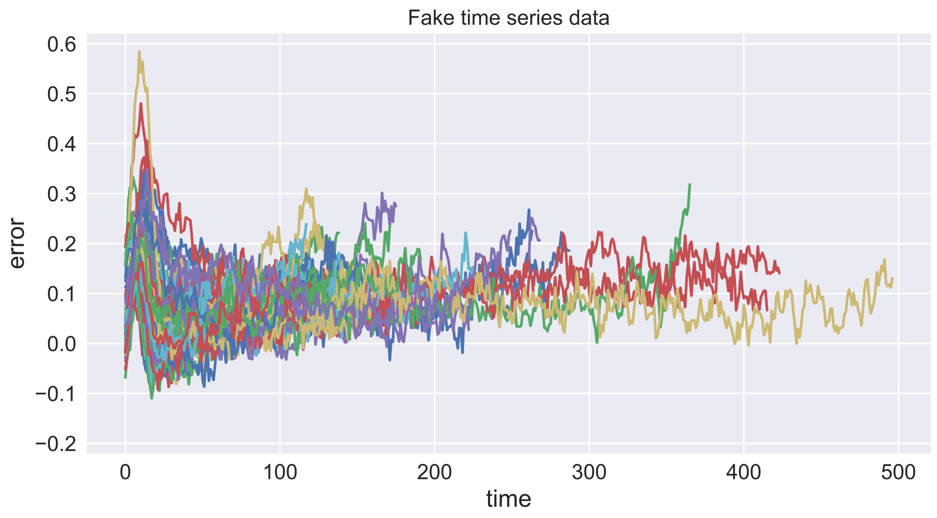
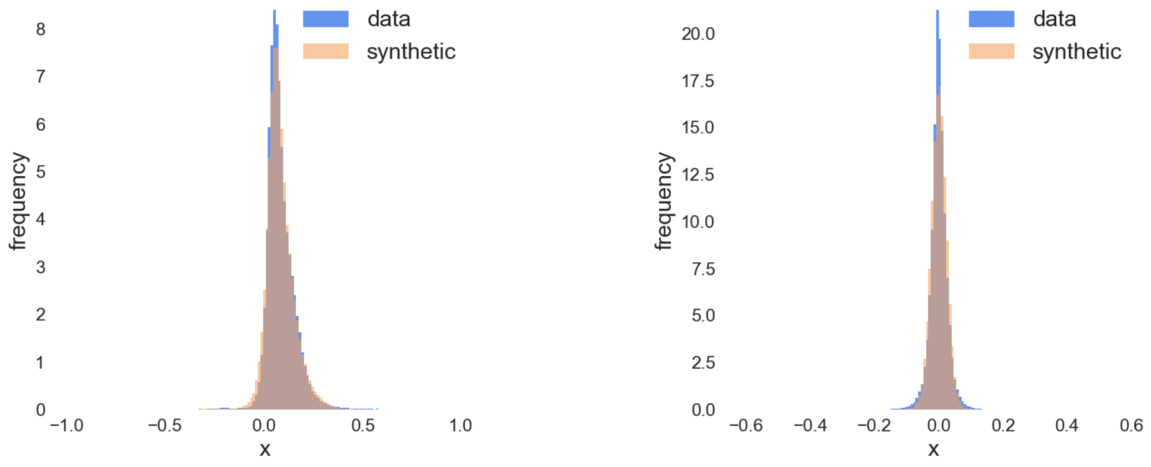


Figure 4.2: The figure depicts 100 time series of the longitudinal error generated by the RCGAN. The corresponding features to the 100 time series plotted in figure 4.1 has been given to the generator. All errors have been rescaled due to confidentiality reasons.

In the histograms in figure 4.3 below together with the JSd value we see that the generator is truly able to grasp the overall trend of the data well, with a JSd of 0.075. The lack of unique features is reflected in the histogram of first difference where the tails of the true data is not quite captured by the generator.

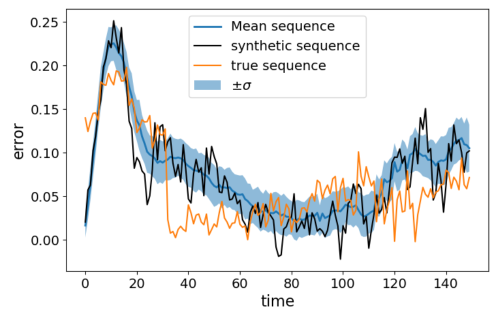


(a) Histogram of the generated errors, JSd = 0.075

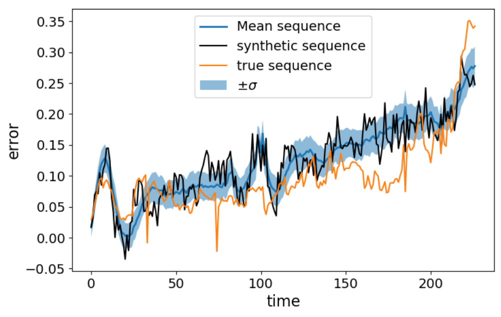
(b) Histogram of the first difference, JSd = 0.095

Figure 4.3: The histograms depict real (blue) and synthetic (sandy-brown) data. The histogram to the left show the distribution of the errors from all time series in the validation set. The right histogram shows the distribution of the first difference of the same data.

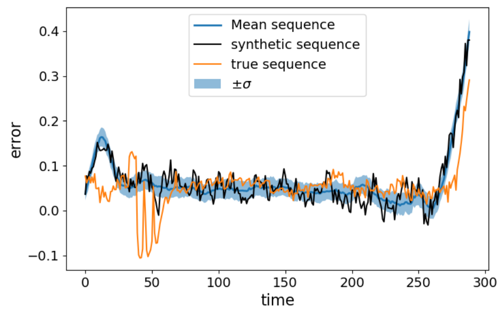
In Figure 4.4, four time series from the validation set is plotted together with the synthetic counter part. The mean of 100 runs show that the RCGAN is able to output trajectories that are close to the real data each time with a suitable noise level seen be the one standard deviation fill around the mean of the 100 synthetic sequences.



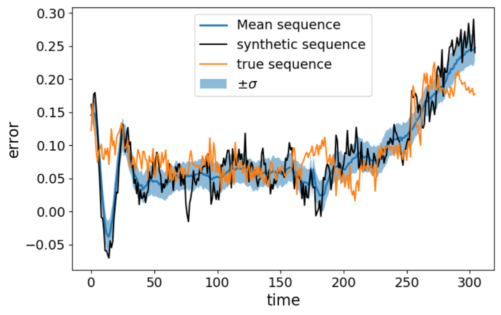
(a)



(b)



(c)



(d)

Figure 4.4: The four plots show four different time series from the validation set. In the plots we see the mean of 100 runs where the same sequence of features is fed to the generator each run, the fill around the mean is $\pm$ one standard deviation. The black trajectory is one of the 100 synthetic sequences and the orange is the actual sequence. All errors have been rescaled due to confidentiality reasons.

4.3 Recurrent MDNs

In Figure 4.5 we see a different appearance of the 100 synthetic time series. With the R-MDN model the noise on top of the error is much spikier compared with the real data, despite this the overall trend of the time series seems to be captured well. The initial parts of the generated sequences show a plausible initial spread compared with the true data.

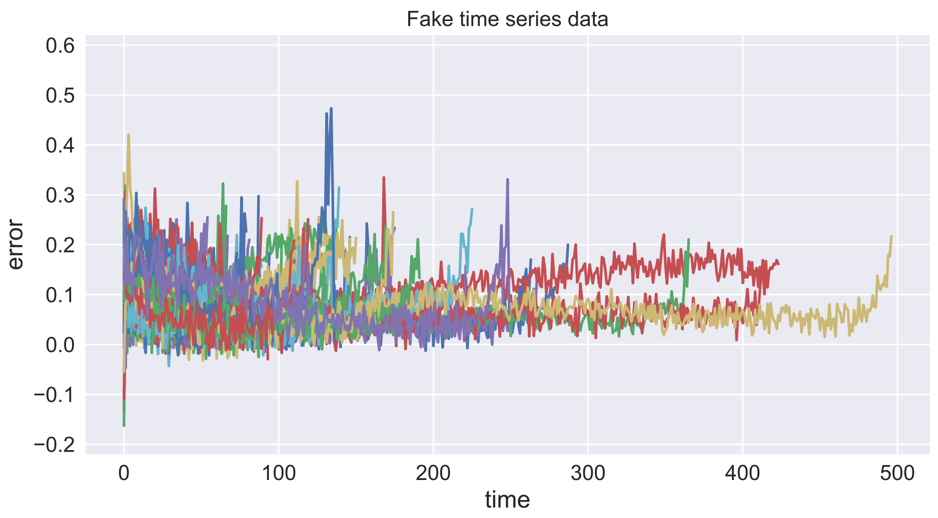
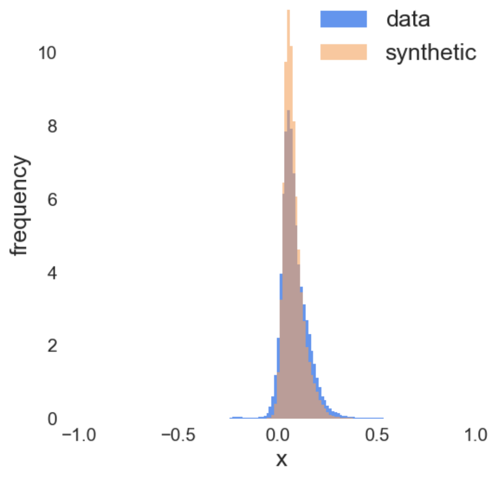
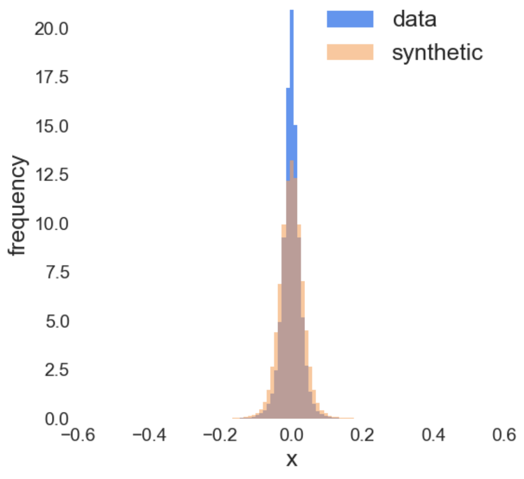


Figure 4.5: The figure depicts 100 time series of the longitudinal error that were sampled from the output GMMs in each time step of the recurrent MDN model. The corresponding features to the 100 time series plotted in figure 5 has been given to the network. All errors have been rescaled due to confidentiality reasons.

In the histogram of the errors in Figure 4.6a we see that the R-MDN is not as expressive as the real data where the tails of the distribution is not being captured. When looking at how the noise is represented in Figure 4.6b we clearly see that there is too much of it where the tails of the synthetic distribution are larger than for the true data. Due to this the tall peak at zero is not fully replicated.



(a) Histogram of the generated errors, JSd = 0.152



(b) Histogram of the first difference, JSd = 0.160

Figure 4.6: The histograms depict real (blue) and synthetic (sandy-brown) data. The histogram to the left show the distribution of the errors from all time series in the validation set. The right histogram shows the distribution of the first difference of the same data.

4.4 Network trained with mean absolute error

From the plot of the 100 time series in Figure 4.7 below we see that when the generator is trained using the mean absolute error instead of the discriminator network. In other words, the output of the generator network is directly used in the MAE loss function for a supervised training of the generator. The model trained in this setting is not able to capture the noise of the true data. Furthermore, the sinusoidal shaped initial transient is present for some of the sequences but not for all. Additionally the spread of the different time series is much tighter than what is seen for the real data in figure 4.1.

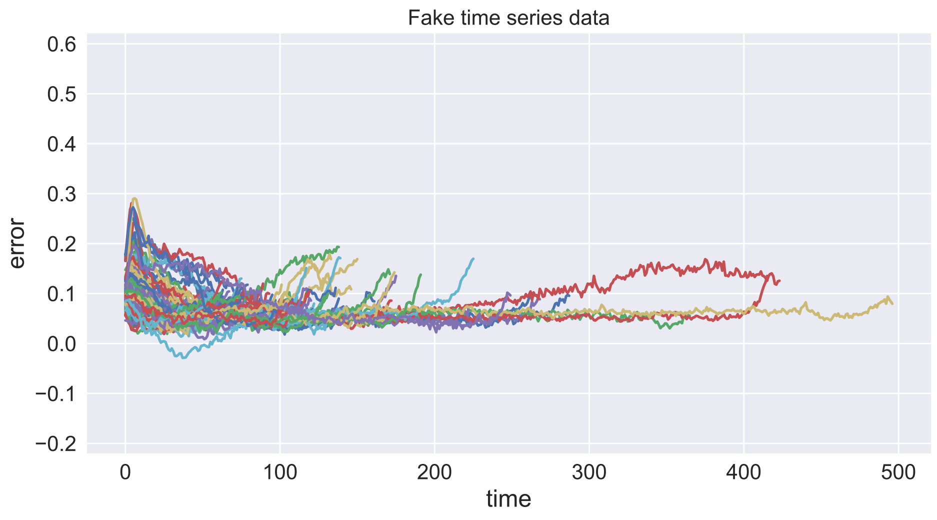
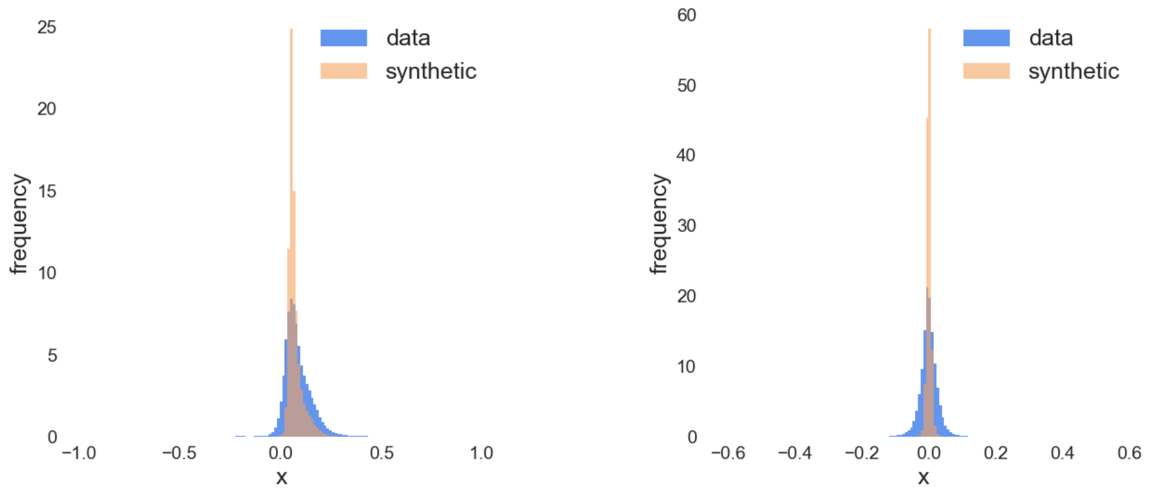


Figure 4.7: The figure depicts 100 time series of the longitudinal error generated by the generator network when trained using MAE instead of the discriminator network. The corresponding features to the 100 time series plotted in figure 4.1 has been given to the generator. All errors have been rescaled due to confidentiality reasons.

The observations are further confirmed in the histograms in figure 4.8 where the synthetic error sequences is clumped together at similar values with a short tail towards positive values. The lack of expressiveness is reflected in the missed out tails compared with the real data as well as in the very tight spread of the first difference of the synthetic errors.



(a) Histogram of the generated errors, JSd = 0.377

(b) Histogram of the first difference, JSd = 0.508

Figure 4.8: The histograms depict real (blue) and synthetic (sandy-brown) data. The histogram to the left show the distribution of the errors from all time series in the validation set. The right histogram shows the distribution of the first difference of the same data.

4.5 Comparison of GAN extensions

The networks here have a different setup of hyperparameters than the best performing network whose results were presented in 4.2 above. The amount of neurons in each layer is lower in order to lower the amount of time spent training for the evaluation of the different training objectives. Between all networks presented in this section all hyperparameters and input features has been kept the same for a fair comparison between the trained networks. In Table 4.2 below is a summary of the JSd for the different objective functions that was tested in this thesis. As previously mentioned we cannot solely rely on the JSd to judge the trained model as it does not cover the temporal aspect so the 100 time series plots are also given under each respective section.

Table 4.2: The JSd metrics of the RCGAN when trained with different loss objective functions.

Loss function	**JSd**	**1^{st} diff JSd**
Original loss	0.0853 ± 0.001	0.114 ± 0.0008
WGAN-GP	0.115 ± 0.0006	0.123 ± 0.0006
f-GAN (Total-variation)	0.132 ± 0.001	0.0916 ± 0.0007
f-GAN (Pearson χ^2)	0.110 ± 0.001	0.0765 ± 0.0008
f-GAN (Neyman χ^2)	0.0982 ± 0.0009	0.0769 ± 0.0008
f-GAN (Squared Hellinger)	0.180 ± 0.0009	0.194 ± 0.0006

4.5.1 Original loss function

One of the best performing training schemes remain as the original loss function (2.6), which allows the generator to learn the overall distribution well together with a realistic amount of noise, see figure 4.9. Furthermore, it is able to learn some finer details in the data but not as much as can be found in the real data. Examining the histograms in figure 4.10 give a similar picture where the overall error distribution agree well between the synthetic and real data distributions but the lack of finer details, e.g. spikes, result in a bit lower first difference JSd.

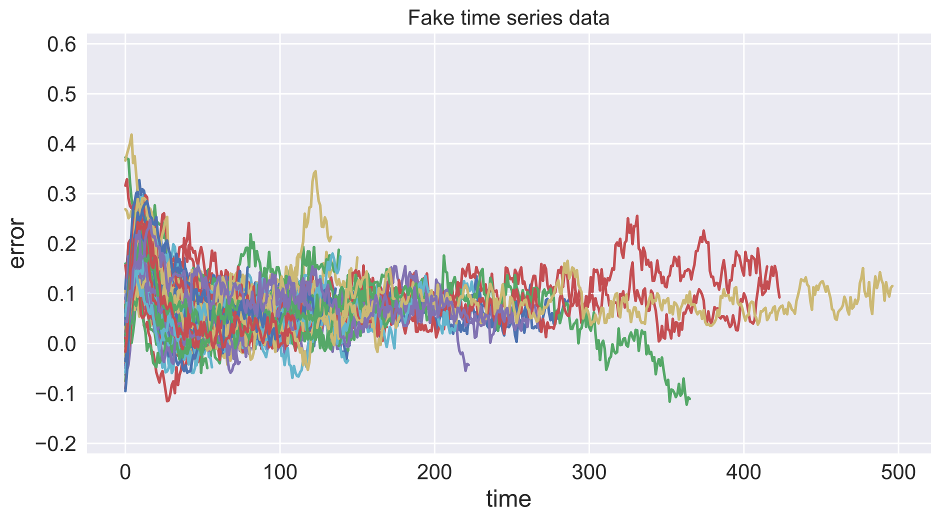
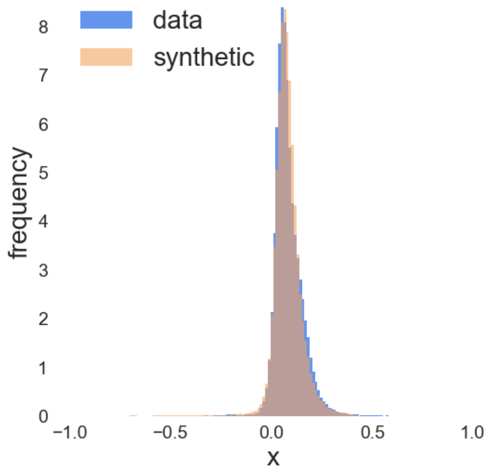
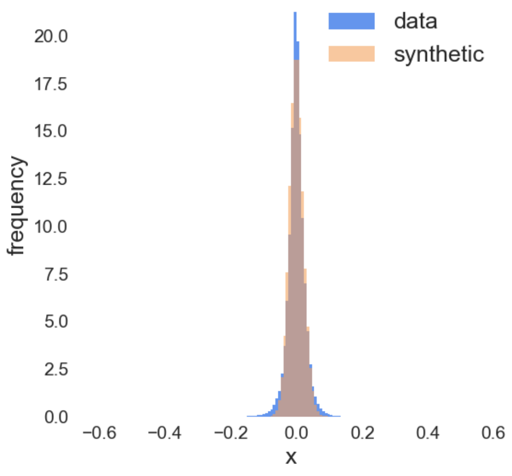


Figure 4.9: The figure depicts 100 time series of the longitudinal error generated by the RCGAN trained using the original loss function. The corresponding features to the 100 time series plotted in figure 4.1 has been given to the generator. All errors have been rescaled due to confidentiality reasons.



(a) Histogram of the generated errors, JSd = 0.0845



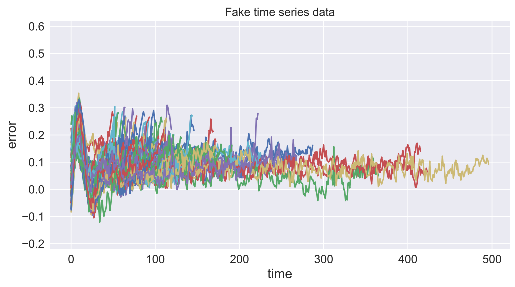
(b) Histogram of the first difference, JSd = 0.116

Figure 4.10: The histograms depict real (blue) and synthetic (sandy-brown) data. The histogram to the left show the distribution of the errors from all time series in the validation set. The right histogram shows the distribution of the first difference of the same data.

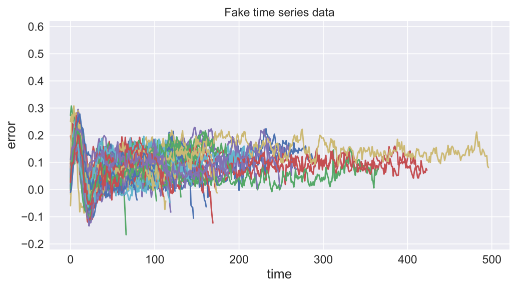
4.5.2 *f*-GAN

Training the RCGAN using the variational *f*-divergence objective functions, see table 2.1, yields different characteristics to the model depending on the selected function. In figure 4.11 we see the 100 time series samples from each of the four *f*-divergence optimization objectives that there were examined in this thesis. All four models share the same initial transient characteristic where it lasts for a different amount of time frames for each model.

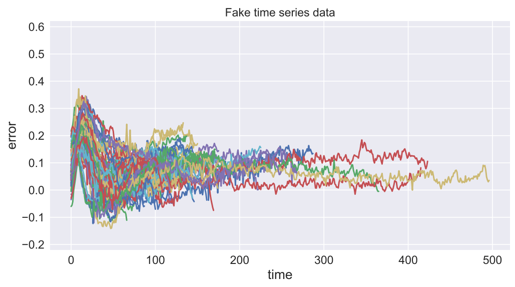
We see for both Total variation and Pearson χ^2, figure 4.11a and 4.11d respectively, that many time series end with a steep ascent. Lastly, the noise levels seems to agree well with the true data for each of the training objectives, although Squared Hellinger is slightly smoother and lack unique noise features, e.g. spikes, that the others has. These observations are reflected in the histograms in figure 4.12 and 4.13 where the histograms agree fairly well for all except Squared Hellinger. The fast ascents at the end of time series trained with the Pearson χ^2 objective are reflected in the first difference histogram, Figure 4.13d, where the tails are much wider for the synthetic distribution. These ascents can be seen in Figure 4.11d where there are spikes approaching 0.4 between about 50 and 100 time frames.



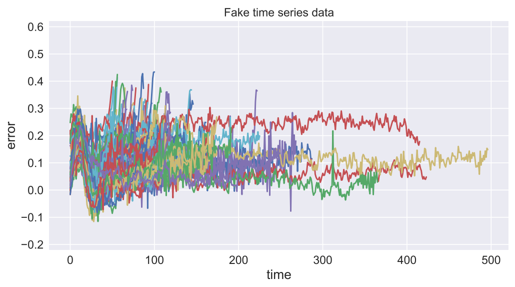
(a) Total variation



(b) Neyman χ^2

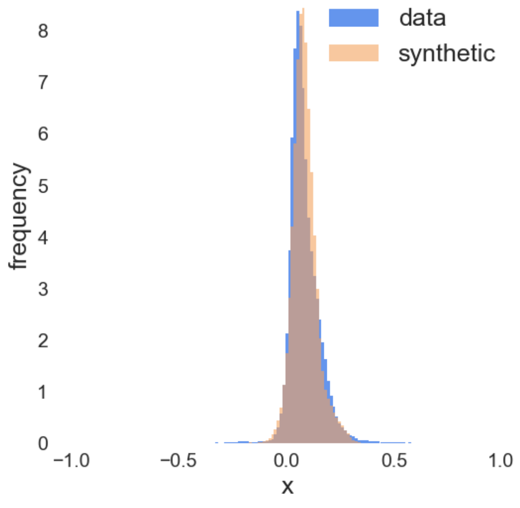


(c) Squared Hellinger

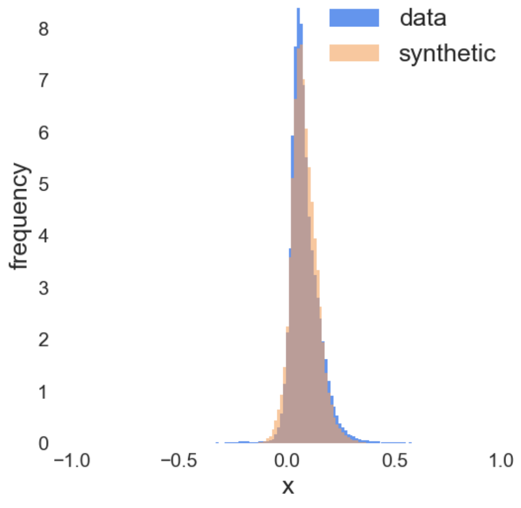


(d) Pearson χ^2

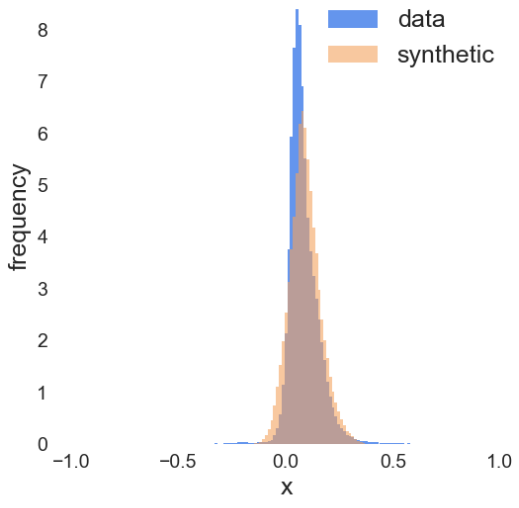
Figure 4.11: The four figures depicts 100 time series of the longitudinal error generated by the RCGAN trained using the variational f-divergence minimization objectives. The corresponding features to the 100 time series plotted in figure 4.1 has been fed to the generator. All errors have been rescaled due to confidentiality reasons.



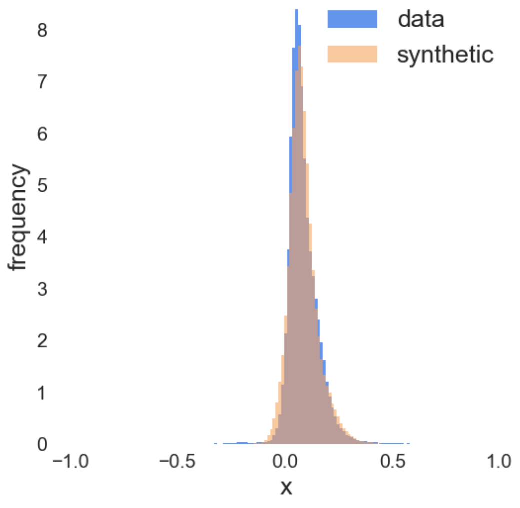
(a) Total variation, JSd = 0.132



(b) Neyman χ^2, JSd = 0.105

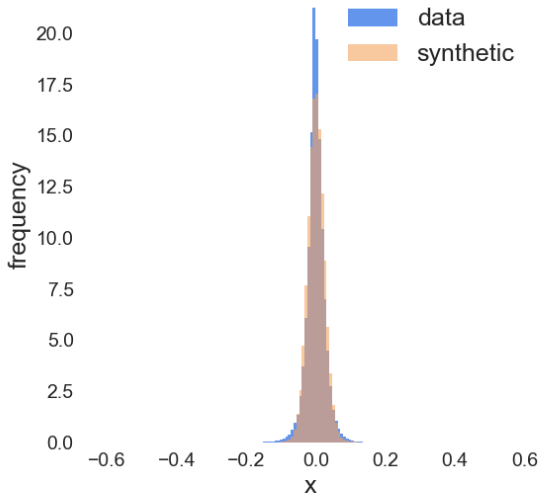


(c) Squared Hellinger, JSd = 0.179

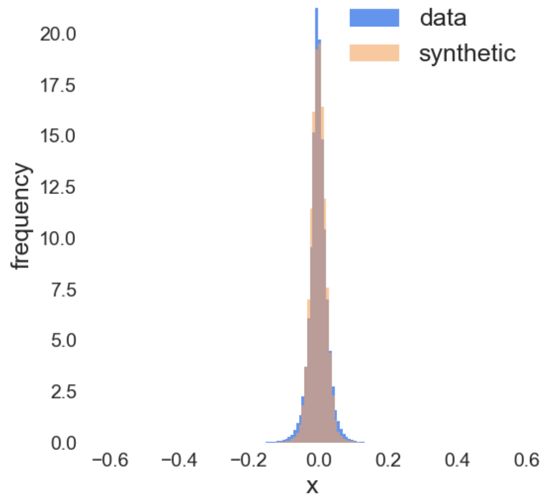


(d) Pearson χ^2, JSd = 0.109

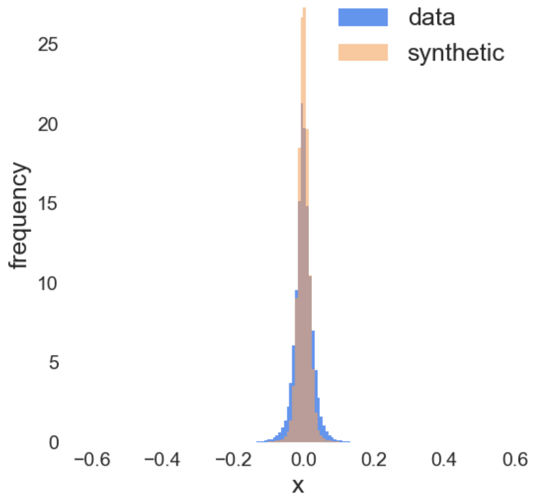
Figure 4.12: Histograms of the validation set (blue) and the synthetic errors (red) from models with different objective functions in the f-GAN setting.



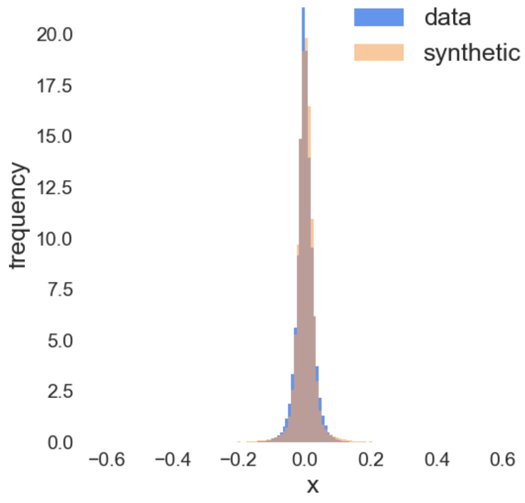
(a) Total variation, JSd = 0.091



(b) Neyman χ^2, JSd = 0.075



(c) Squared Hellinger, JSd = 0.194



(d) Pearson χ^2, JSd = 0.078

Figure 4.13: Histograms of the first difference for the validation set (blue) and the synthetic errors (red) from models with different objective functions in the f-GAN setting.

4.5.3 WGAN-GP

As with the previously presented models we have an initial transient that is very similar for all time series, seen in figure 4.14. Furthermore, the values of the generator seems to lie with a tighter spread compared with the previously presented model with next to no outlier values which is seen in the real data. Lastly, the histograms in figure 4.15 show that the trained model is not quite able to generate errors in the upper range of the positive valued errors. In addition, the first difference histogram further reflect the tight spread of values seen in figure 4.14 where the tails of the synthetic distribution are not as wide in comparison with the real distribution.

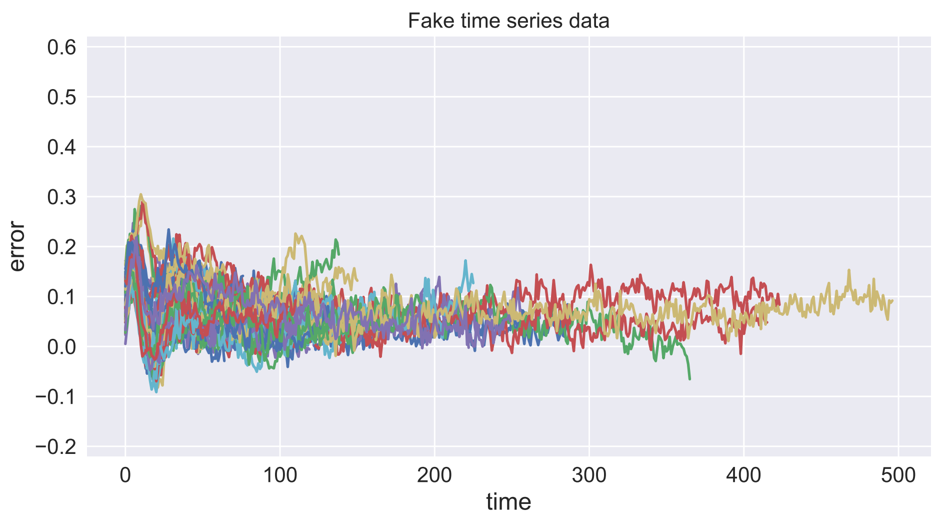
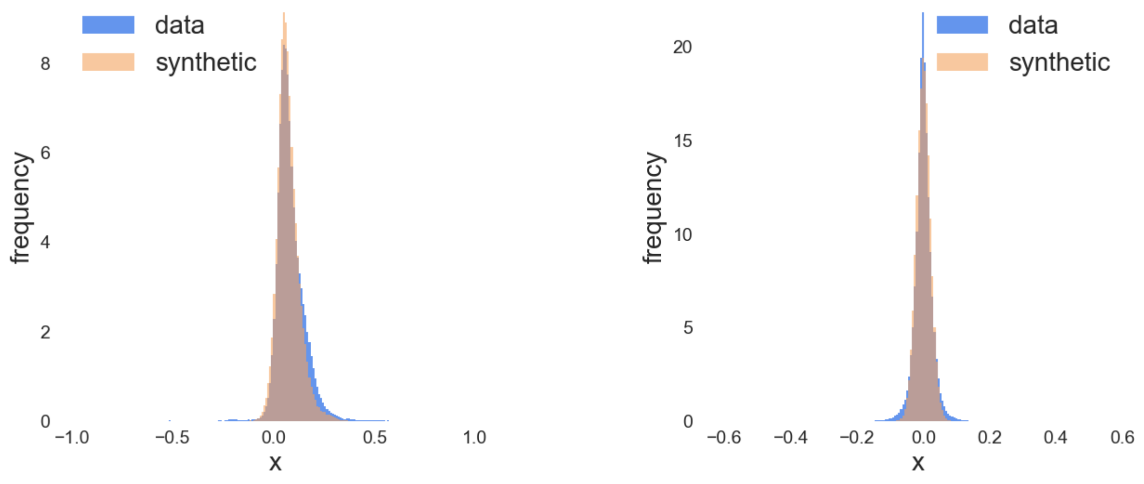


Figure 4.14: The figure depicts 100 time series of the longitudinal error generated by the RCGAN trained using the WGAN with gradient penalty loss function. The corresponding features to the 100 time series plotted in figure 4.1 has been given to the generator. All errors have been rescaled due to confidentiality reasons.



(a) Histogram of the generated errors, JSd = 0.116

(b) Histogram of the first difference, JSd = 0.124

Figure 4.15: The histograms depict real (blue) and synthetic (sandy-brown) data. The histogram to the left show the distribution of the errors from all time series in the validation set. The right histogram shows the distribution of the first difference of the same data.

5 Discussion

The resulting model from this thesis has been proven to be successful at generating synthetic time series. The network structure in figure 2.2 and 2.3 has shown great performance when it comes to learning the general trends of the data as well as outputting a suitable noise level on the signal. In this chapter we will discuss the strengths and drawbacks of the different models and training objectives of the recurrent conditional generative adversarial network that were looked at in this thesis.

5.1 Comparison of models

The recurrent conditional GAN is the best performing model on this data set. In the data set there are sequences that has two common trends, starting out at higher values and going towards lower or the other way around. The trained model is able to capture both of these trends, see Figure 4.2 and 4.4. The real data have a highly auto-regressive nature, i.e. the error is highly dependent on the error in previous time steps. In other words if the error have been large for a few time steps it is likely to be large in the next time steps as well. We see that the RCGAN is able to capture this auto-regressive nature of the real data with realistic looking noise. So it seems that the generator has learned how much of the input noise $\boldsymbol{z}$ should be let through on top of the signal.

For all GANs in this thesis we have observed an substantial initial transient for all synthetic time series, where some of the trained models exhibit more and others less. This transient makes the RCGAN have poor predictions in the beginning of the synthetic sequences compared with the real data. Furthermore the transient is similar for many different time series which is an undesirable trait that does not reflect the real data well. This issue comes mainly from the LSTM nodes in the network whose internal states has to be built up from feeding consecutive inputs. When enough time frames has been given to the networks they start to grasp the context of the sequence and the output starts to home in on the desired trajectory. Attempts to solve the issue with the initial transient have been made by trying different initialization schemes. For example initializing the LSTMs with noise from either uniform or normal distributions. This does indeed cause the transient to behave differently both for good and for worse but the main issue with this is that the learning of the model is slowed down greatly and becomes even more unstable.

The recurrent mixture density network is an interesting approach to the problem, instead of injecting noise into the network the randomness is inherent from the model where you sample from the output distributions. The nature of the data set in this thesis is heavily auto-regressive which is completely missed when sampling independently from the output distributions at each time frame. This is clearly seen in the 100 time series in Figure 4.5. One advantage to this network compared with the RCGAN is that the transient at the beginning of the sequences only lasts a couple of time frames and seems to be more diverse. This model could probably be improved significantly by for example, coupling it to an auto-regressive (AR) model where the parameters has to be optimized after the R-MDN network has finished training.

The performance of the network trained by minimizing the distance between synthetic and real samples performs much worse compared with the GAN framework. When training this regular LSTM network the noise levels decay quickly as the network learns not to trust the one-layered noise network which in a sense wish to increase the error of the output. This makes the output nearly deterministic which results in that the noise of the real data is not captured in this approach. This method of training the generator was only implemented in order to compare and justify the GAN framework in this application. If there was more time to spend on it, it would probably be possible to construct a more meaningful loss function than mean absolute error or mean squared error.

5.2 Comparison of training objective

When comparing the different training objectives exclusively on how well the overall distribution match with the real data most of them perform similarly. The exception being optimizing for the Squared Hellinger divergence in the f-GAN setting. Of all the different loss functions that were tested the original loss function was able to learn the error distribution best. When inspecting the synthetic sequences we see that it is able to capture unique features of some sequences, e.g. the yellow peak at $t \approx 120$, see Figure 4.9.

As previously mentioned, most work with GANs has been done to image generation and manipulation which does not necessarily mean that the extensions will work well with time series. From the results section we see that the original loss objective proved to be the best out of the different loss metrics that were tested. The interpretation of the original loss where the output is a probability whether the sample is real or not for every time step feels intuitively like a meaningful loss metric unlike the others whose interpretation remain open for discussion.

Each of the models trained with different f-GAN objectives seem to produce different characteristics in the synthetic time series. Both Total variation and Pearson χ^2 have problems learning the last parts of the sequences where many ”explode” towards the end. Meanwhile, Squared Hellinger seems to regularize away the noise producing smoother time series than the three other objectives and it also have a wider spread of the errors in the initial parts of the sequences. The best performing optimization objective of the four is the Neyman χ^2 f-divergence which had the best JSd scores of the f-GAN objectives. In addition, the trained model was also able to produce the most visually pleasing sequences of the four. Lastly, only a marginal improvement to the stability of the learning was observed for Squared Hellinger and Neyman χ^2 and Total variation and Pearson χ^2 were more unstable during training than the original loss function.

The WGAN-GP extension was not the top performer, but it still performed fairly well. This is surprising as the loss metric was highly unstable and could be many orders of magnitude larger for some time series during training. Despite this, we saw a more stable output distribution and learning during the training of the model compared with the original loss function.

The choice of the WGAN-GP extension may not be ideal as the extension was developed to improve most common problem that GANs are used for, namely image processing where it performs well [24]. However, when it comes to comes to the RCGAN implementation in this thesis with different lengths on the input time series problems with the gradient penalty term arise. Firstly, taking the L2-norm of the gradients of time series with different lengths skew the penalty term heavily for long time series compared with shorter ones. Secondly, further investigation has to be made on how to deal with the conditional input when calculating the gradient penalty term. Calculating the gradients with respect to both $\hat{\boldsymbol{x}}$ in (2.3) and the conditional vector $\boldsymbol{c}$ was tested but no improvement was seen. Despite this, we saw a more stable learning compared with the original GAN loss function and the model learned the overall trend of the data fairly well. As with all models where the LSTMs states were initialized with zeroes the sinusoidal transient in the beginning is present.

5.3 Training

Many articles [22, 23, 24, 25] have pointed out that the original implementation of GAN is unstable to train. According to the literature GANs often experience what is known ”mode-collapse” where the generator produce the same output no matter which input is given to the network. If this happens the training procedure has to be reset. During the work in this thesis mode-collapse has been experienced and has mainly been a result of poor weight initialization or a too large learning rate. It was much less of a problem than anticipated.

The main issue when training the models has been the instability in regard of convergence to a stable minimum, which is the main reason for examining different training objectives, e.g. f-GANs. The synthetic data distribution usually drift back and forth during training. Sometimes training for another 200 iterations will give the model a higher JSd than before and sometimes not. There are a few reasons for this, firstly there is an inherent instability to the training of GAN which has been mentioned previously. Secondly, there are issues with the data that is used for training where some time series contain irregularities. These may cause the loss to behave oddly for these which draw the generator away from a well matching output distribution. This is further emphasized by the fact that no batch training is used as the sequences in the training data are of different lengths. The reason for this being that there is no support in Tensorflow 1.5 for feeding the network sequences of different length. Attempts to address this has been made by filtering out time series with anomalies from the data set, such as when the error jumps a significant amount between two consecutive time frames. The back and forth drift can also be combated to an extent by using a relatively low learning rate which seems to dampen the drift.

Finally, it is possible to obtain a well-performing RCGAN model quite quickly in terms of wall-clock time. Training the model on an Intel Core I7-7700HQ processor for less than 30 minutes can result in a model with a JSd of 0.15 and visually acceptable sequences. Training the RCGAN in order to obtain a top performing model will take just a few hours which is great for trying out different data sets and parameter settings.

5.4 Model evaluation

In the results section we see examples of why a good JSd value not necessarily imply a well trained model. A good example of this is for the Pearson χ^2 objective in f-GANs where both JSd values for the synthetic data and first difference are low, meaning a good agreement between the distributions. Meanwhile, when the time series are inspected visually in Figure 4.11d we see that some time series ”explode” towards the end which is a behaviour that is not found in the real data. These quick ascents inflate the amount of values that falls in the tails of the histograms which exploit that the metrics is invariant under permutation. Until better metrics for time series evaluation is found, visual inspection of the generated sequences remain important.

6
Conclusion

In this thesis a recurrent conditional generative adversarial network framework has been developed. The framework fulfills the desired characteristics where the model can handle time series of arbitrary length as well as being able to tune the noise levels to the specific time series distribution that is wished to be learned. As it is possible to run the network on arbitrary long sequences it is also possible to train the framework on data sets where the length of sequences are different. Furthermore, the networks output is able to follow the trend of the real data where a believeable amount of noise is applied to the signal.

The recurrent mixture density networks also perform quite well and is able to capture the trend of the time series. However, since the samples are drawn independently from the output GMMs in each time frame we miss out on the auto-regressive nature of the real data.

Having the generator trained in a supervised fashion by removing the discriminator and replacing it by a loss function whose purpose is to minimize the distance between synthetic and real samples proved to not be a suitable approach. The unique time series criteria is not fulfilled and the noise in the output signal is not satisfactory. The network structure in the generator has proven to work well and yield good results when trained in the RCGAN framework, so the network have the capability of learning the distribution but is not captured in this setting. This further promote the use of the RCGAN framework for training the generative model.

6.1 Future work

The RCGAN architecture presented in this thesis will continue to be developed as it has shown great results for generating realistic time series. Firstly, a new initialization method or modification to the internal architecture have to be developed that allow the model to produce a more realistic initial transient. Secondly, further work has to be made in order to make it possible to utilize mini-batch training for sequences of different lengths, which is a change that may help stabilize the learning. Thirdly, the model will be implemented in the simulation environments at Zenuity in order to further evaluate and improve the model. It would also be interesting to find or develop other suitable scoring metrics in order to improve the evaluation of the trained models.

Finally, it would be interesting to implement other state-of-the-art generative models in order to compare it against the RCGAN architecture. One approach which has shown impressive results in generating synthetic wave-forms in speech-synthesis tasks is WaveNet [5], which should be able to perform well in the error sequence task as well.

Bibliography

[1] A. L. et al., “”Deep learning in the automotive industry: Applications and tools”,” *IEEE International Conference on Big Data*, pp. 3759–3768, 2016.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Kalra, Nidhi and Susan M. Paddock. ”Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?”. Santa Monica, CA: RAND Corporation, 2016. `https://www.rand.org/pubs/research_reports/RR1478.html.`

[4] A. Graves, “Generating sequences with recurrent neural networks,” *CoRR*, vol. abs/1308.0850, 2013.

[5] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, “Wavenet: A generative model for raw audio,” *CoRR*, vol. abs/1609.03499, 2016.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “”Generative Adversarial Networks”,” *ArXiv e-prints*, June 2014.

[7] E. L. Zec, N. Mohammadiha, A. Schliep, “Modelling Autonomous Driving Sensors Using Hidden Markov Models”, under review, 2018.

[8] E. Karlsson, N. Mohammadiha, “A Statistical GPS Error Model for Autonomous Driving”, in Proc. IEEE Intelligent Vehicles (IV), June 2018.

[9] P. Isola, J. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” *CoRR*, vol. abs/1611.07004, 2016.

[10] J. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” *CoRR*, vol. abs/1703.10593, 2017.

[11] Y. Taigman, A. Polyak, and L. Wolf, “Unsupervised cross-domain image generation,” *CoRR*, vol. abs/1611.02200, 2016.

[12] M. Liu and O. Tuzel, “Coupled generative adversarial networks,” *CoRR*, vol. abs/1606.07536, 2016.

[13] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. P. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," *CoRR*, vol. abs/1609.04802, 2016.

[14] C. K. Sønderby, J. Caballero, L. Theis, W. Shi, and F. Huszár, "Amortised MAP inference for image super-resolution," *CoRR*, vol. abs/1610.04490, 2016.

[15] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, "Learning what and where to draw," *CoRR*, vol. abs/1610.02454, 2016.

[16] S. E. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," *CoRR*, vol. abs/1605.05396, 2016.

[17] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. N. Metaxas, "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks," *CoRR*, vol. abs/1612.03242, 2016.

[18] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," *CoRR*, vol. abs/1609.05473, 2016.

[19] O. Mogren, "C-RNN-GAN: continuous recurrent neural networks with adversarial training," *CoRR*, vol. abs/1611.09904, 2016.

[20] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," *CoRR*, vol. abs/1706.02633, 2017.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[22] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *CoRR*, vol. abs/1606.03498, 2016.

[23] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," *ArXiv e-prints*, Jan. 2017.

[24] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *CoRR*, vol. abs/1704.00028, 2017.

[25] R. Nock, Z. Cranko, A. K. Menon, L. Qu, and R. C. Williamson, "f-gans in an information geometric nutshell," *CoRR*, vol. abs/1707.04385, 2017.

[26] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, and Z. Wang, "Multi-class generative adversarial networks with the L2 loss function," *CoRR*, vol. abs/1611.04076, 2016.

[27] C. M. Bishop, "Mixture density networks," 1994.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[29] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," *arXiv preprint arXiv:1511.01844*, 2015.

[30] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

A

Mixture Density Networks

Mixture density networks (MDNs) are a type of neural networks whose output parametrizes a mixture model [4, 27]. Subsets of the outputs are used to define different parameters of the mixture model. Where one set is used to define the mixture weights (π) and the rest is used to define the individual components of the mixture model. For example, if the output parametrizes a Gaussian Mixture Model (GMM) there is a need of three outputs per mixture component, two additional components for the location (μ) and scale (σ) parameters. In Figure A.1 a schematical view of an MDN is presented, the output is a GMM with two mixture components in this example.

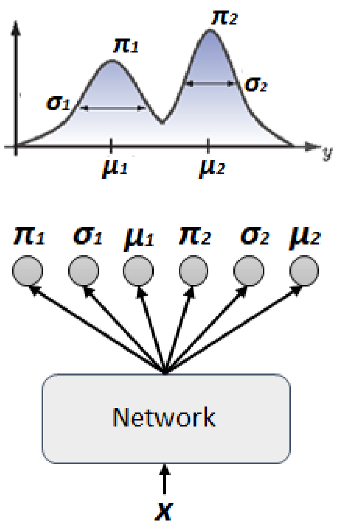


Figure A.1: A schematical view of an MDN where the output mixture is a GMM. For each input x a new set of parameters $\{\pi_j, \sigma_j, \mu_j\}_{j=1,2}$ is obtained. Here π denote the mixture weight, σ the scale parameter and μ the location parameter. From this output-GMM samples are drawn.

For each parameter a suitable activation function is applied in order to bring their values into meaningful ranges, see (A.1) below. In order to obtain a valid discrete distribution the mixture weights are normalized by applying the softmax function which makes the mixture weights to sum up to unity. Furthermore, no activation function, e.g. linear activation, is applied to μ to not limit the location of the mixture components. Finally, the scale parameters are exponentiated to limit them to positive values.

$$\begin{aligned} \pi^j &= \frac{\exp(\hat{\pi}^j)}{\sum_{j'=1}^{M} \exp(\hat{\pi}^{j'})} && \implies \pi^j \in (0,1), \quad \sum_j \pi^j = 1 \\ \mu^j &= \hat{\mu}^j && \implies \mu^j \in \mathbb{R} \\ \sigma^j &= \exp\Big(\hat{\sigma}^j\Big) && \implies \sigma^j > 0, \end{aligned} \tag{A.1}$$

where j is the number of mixture components. The MDN is trained using the negative logarithm of the likelihood, according to (A.2).

$$\mathbb{L}(x) = -\log\left(\sum_j \pi^j \mathcal{N}(x|\mu^j, \sigma^j)\right), \tag{A.2}$$

where x are real samples.

A.1 Recurrent Mixture Density Networks

The mixture density network can be combined with recurrent neural networks into a Recurrent Mixture Density Network (R-MDN) which allows the output mixture components to not only be conditioned on current input but also on previous inputs. A schematical image of a R-MDN is presented in Figure A.2, at each time step a sample S is drawn from the output distributions.

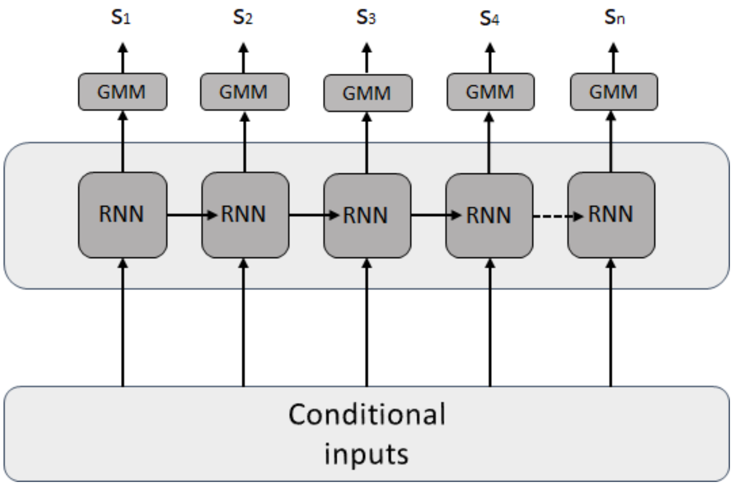


Figure A.2: The structure of the RMDN. Each time step a set of features is given to the network which outputs the parameters for the GMM from which samples S are drawn.

For the R-MDN a different mixture model is obtained in each successive time frame, this is formulated in (A.3) below.

$$\begin{aligned}
\pi_t^j &= \frac{\exp\left(\hat{\pi}_t^j\right)}{\sum_{j'=1}^{M}\exp\left(\hat{\pi}_t^{j'}\right)} && \implies \pi_t^j \in (0,1), \quad \sum_j \pi_t^j = 1 \\
\mu_t^j &= \hat{\mu}_t^j && \implies \mu_t^j \in \mathbb{R} \\
\sigma_t^j &= \exp\left(\hat{\sigma}_t^j\right) && \implies \sigma_t^j > 0,
\end{aligned} \tag{A.3}$$

where j is the mixture components and t is the number of time steps in the sequence. The training objective is also modified according to (A.4) to include the new temporal aspect of the outputs.

$$\mathbb{L}(\mathbf{x}) = \sum_{t=1}^{T} -\log\left(\sum_j \pi_t^j \mathcal{N}(x_t|\mu_t^j, \sigma_t^j)\right) \tag{A.4}$$