ACIT5900

MASTER THESIS

in

Applied Computer and Information Technology (ACIT)

May 2022

Applied Artificial Intelligence

Can GANs replicate eye-gaze trajectories?

Marit Øye Gjersdal

Department of Computer Science

Faculty of Technology, Art and Design

OSLOMET

Acknowledgements

I am sincerely grateful to my supervisor Pedro Lencastre for all the support, help, and motivation throughout this semester. This thesis would not be possible without his constant guidance and encouragements. I would also like to thank my co-supervisors Prof. Pedro Lind and Prof. Anis Yazidi. Their guidance and feedback have been greatly appreciated. Lastly, a special thanks my fellow students for keeping me sane during this endeavour, and to my family for encouragement and support.

Marit Øye Gjersdal,
15.05.2022

Abstract

Generative Adversarial Networks (GANs) have gained popularity in the field of computer vision. Recently, GANs have shown promising results in generating sequential data, such as time series data and text. This thesis explores the ability of a variety of time-series GAN architectures to generate realistic eye-gaze trajectory data, with the aim to create synthetic datasets for research within Machine Learning (ML) and statistics. The experiments were conducted in four stages, with increasingly more complex data for the GANs to generate, to study the limitations of each GAN model. The first experiments were done on synthetically generated data of Vector Autoregressive (VAR) processes and intermittent processes, and the final experiment was conducted on real eye-gaze trajectories. We show that even though several time-series GAN models are capable of generating seemingly realistic VAR processes and intermittent processes, there is still some way to go to be able to generate realistic eye-gaze trajectories. We discuss the limitations of a range of GAN models, and propose future experiments and models which could be further studied.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence.
ANN	Artificial Neural Network.
AR-FNN	Autoregressive Feed Forward Neural Network.
BERT	Bidirectional Encoder Representations from Transformers.
CNN	Convolutional Neural Network.
CWGAN	Conditional Wasserstein GAN.
GAN	Generative Adversarial Network.
GRU	Gated Recurrent Unit.
IoR	Inhibition of Return.
KS	Kolmogorov Smirnoff.
LSTM	Long Short-Term Memory.
ML	Machine Learning.
NLP	Natural Language Processing.
RCGAN	Recurrent Conditional GAN.
RCWGAN	Recurrent Conditional Wasserstein GAN.
RNN	Recurrent Neural Network.
SigCWGAN	Conditional Sig-Wasserstein GAN.
TimeGAN	Time-series GAN.
VAE	Variational Auto-Encoder.
VAR	Vector Autoregressive.

Chapter 1

Introduction and motivation

Serving as one of our primary senses, eyesight transcends its primary vital function of collecting visual information. The very way we forage for information using our eyes has been observed to be unique to every person, and a person's gaze has shown to also provide information about several aspects such as age, gender, ethnicity (Kröger et al., 2019; Mardanbegi et al., 2018), personality traits (Berkovsky et al., 2019), drug consumption (Grace et al., 2010; Steffens et al., 2016), empathy levels (Z. Yan et al., 2017), IQ and skills in a particular field (Kasneci et al., 2021; Thompson, 2021), as well as getting useful information that could help diagnosing ADHD (Chauhan et al., 2020; Lev et al., 2020), Alzheimer's (Antoniades et al., 2010; Burrell et al., 2012; Perneczky et al., 2011) or autism (Wadhera & Kakkar, 2019), or be used as a biometric (Bargary et al., 2017; Rigas & Komogortsev, 2017). Thus, studying eye movement trajectories has been, and still is, of great interest in the fields of medicine and psychology.

When it comes to foraging for visual information, due to millions of years of biological evolution the eyes present a complex behaviour that is highly efficient at finding objects of interest. The process of eye-gaze during foraging for information in new images and scenery shares properties with other search processes found in nature (Brockmann & Geisel, 2000).

Gathering precise eye-trajectory data is still a tedious task, which can only be done accurately using professional eye-trackers. This data is often used for research in medicine and psychology. However, since data from psychology and psychiatry is extremely sensitive, gathering these data opens up challenges regarding the confidentiality of the data. This concern typically makes these datasets inaccessible for public usage and research (Hazra & Byun, 2020). Generating new unique data which shares the properties of the real data but is untraceable to the real subjects might open doors for future studies both within Machine Learning (ML) and statistics. Such data is valuable for applications in medicine and psychology, which rely on abundant eye-tracker data.

Machine Learning (ML) is a sub-field of Artificial Intelligence (AI) that is concerned with learning from data without being explicitly told the rules of the data domain. ML has shown to be very useful for capturing complex structures in data and can efficiently be used for generation or classification. Training artificial neural networks (ANNs) in the Generative Adversarial Network (GAN) (Goodfellow et al., 2014) fashion can work very well for generating data replicating complex structures. Even though GANs were originally designed for computer vision tasks, they have more recently also gained popularity for generating sequential data (Brophy et al., 2021).

For this thesis we will study a range of different state-of-the-art GAN models for sequential data, to find out whether these models will be ideal for replicating human eye-gaze trajectories. The end goal is to study whether existing GAN models have the ability to capture all the properties of eye-gaze trajectories to a degree where it's indistinguishable from real data. The research question this thesis addresses is thus as following: Can GANs replicate eye-gaze trajectories of humans searching for information?

Machine learning models are often described as black boxes, where you can feed them an input and they produce an output, but what happens inside the model

and why it makes the decisions it makes is hard to explain. They are however extremely useful for capturing complex structures. Statistical based methods on the other hand are explainable, but they are not as suitable for representing very complex structures. Mixing machine learning with statistics can be used in a complementary manner to improve them both. By using Machine Learning models we can check if the statistical methods have captured all the properties of the eye-gaze data or are missing some aspects, and likewise the other way around we can use statistics to measure the performance of the machine learning models.

Such is the case in this thesis, where statistics and AI can play a complimentary role. When the data is of such complex structures that its quality cannot be measured statistically, there doesn't exist any way to accurately evaluate whether the generated data indeed captured the properties and is indistinguishable from real data or not. This is the case for eye-gaze trajectory data. We will therefore first be using synthetic data, where the properties are known, for the testing of a range of time-series GANs. This way we can to study the limitation of each of the GAN models. We will first use sequences of simpler processes called Vector Autoregressive (VAR). Then we will generate intermittent processes, which are processes that both the eyes' foraging process and other searching processes in nature are hypothesised to follow (Land, 2019). If the GAN models are able to generate data that well captures the properties of the synthetic data, we can finally train them on real human eye-gaze trajectory data. The aim is that once trained, the generator can be used to generate eye-gaze trajectory data with the same properties as the training data.

The hypothesis of this thesis is that it is possible to train recent state-of-the-art time-series GANs which can replicate human eye-gaze trajectories, capturing the advanced properties of the human gaze. Further, we hypothesise that if a GAN can successfully be trained to generate data that captures the properties of synthetic data made of intermittent processes, this success can be transferred to real eye-gaze trajectory data. So, if the GANs are trained on real eye-gaze trajectory data

instead, they will be able to generate data with the same properties as the real eye-gaze trajectories.

We assume that by using the synthetic data and evaluating the success of the models using statistical properties, these results will be transferable to how the GAN performs on the real data. There is a risk that the unknown properties of the real data are so different from the synthetic data that the model is not able to capture them even if it was able to capture the properties of the synthetic intermittent process data, so this is something we will stay aware of.

This thesis is structured as follows. Chapter 2 first covers theoretical background on the properties of eye-gaze trajectories and their foraging process, and then introduces neural networks for sequential data domains, before reviewing recent state-of-the-art models of generative neural networks for sequential data. Finally, we also present other research on replicating eye-gaze trajectories. Chapter 3 presents all the experiments we conduct, the models we use, and how we will be evaluating and comparing the result. Chapter 4 gives the results of the experiments conducted, using both synthetic data and real eye-gaze trajectories. In the final chapter, Chapter 5, we discuss the results, our approach, and suggests future ideas for addressing our research question.

Chapter 2

Background and State of the Art

The eyesight is one of our five primary senses and serves as one of the primary information collectors for our brain. The way we use our eyes to forage for visual information has already been used in developing algorithms for self-driving cars (Rao & Frtunikj, 2018), where quickly searching for objects is key. Eye-tracking is also being used for task like optimizing advertisements (Scott et al., 2016), website optimization (Wedel & Pieters, 2017) to determine where in the ad or website the subjects will be looking to help in design optimizations, and rendering games (Kaplanyan et al., 2019) by using a model which guesses where on the screen the player will be looking next. This way rendering times can be decreased by first fully rendering the part of the screen the player is estimated to look at.

Using eye trajectories also have a much larger potential, especially within medicine and psychiatry. The way each of us forage for visual information is unique (Bargary et al., 2017; Rigas & Komogortsev, 2017), and the way we search for information in new images has shown potential potential as a tool in helping to diagnose ADHD (Chauhan et al., 2020; Lev et al., 2020), Alzheimer's (Antoniades et al., 2010; Burrell et al., 2012; Perneczky et al., 2011) and autism (Wadhera & Kakkar, 2019).

Even though it has been shown that we can use eye-gaze trajectories for separating between for example, ill and healthy individuals, there still might remain other properties of eye-gaze trajectories that have not been identified. In particular properties on the memory effect of the eyes, as they are difficult to model mathematically (Lambiotte et al., 2015). Being able to retrieve more properties could possibly help in developing useful tools and application for medicine and psychology. The solution to retrieve more properties might lay in Artificial Intelligence (AI), and with the recent rise of advanced machine learning methods the possibility to utilising eye-tracking along with AI as a medical tool in diagnosing patients has become a potential reality.

This chapter introduces relevant background on the eyes, the way we forage for visual information, and some central properties of eye-trajectories. Further follows an introduction to the relevant AI tools that can be used for eye-trajectories, as well as an introduction to Generative Adversarial Networks (GANs), which usages will be explored in this thesis for generating new unique trajectories imitating the trajectories of the eyes. Finally, in this chapter we provide an overview of state-of-the-art methods for handling and generating sequential data, and an overview of the current research in generating eye-gaze trajectories using neural networks.

2.1 Eye-gaze trajectories

2.1.1 Foraging for information

There are some striking similarities found between various searching techniques in nature. The way sharks forage for food in the ocean (Sims et al., 2012), the movement of bacteria (Ariel et al., 2015; X.-F. Yan & Ye, 2015), and even children exploring the Walt Disney Resort (Rhee et al., 2011) all share similar searching properties in the context of foraging. The foraging hypothesis is a term in biology, which says that through thousands of years of evolution searching techniques in nature have been optimized for survival, and that this is the reason why all these

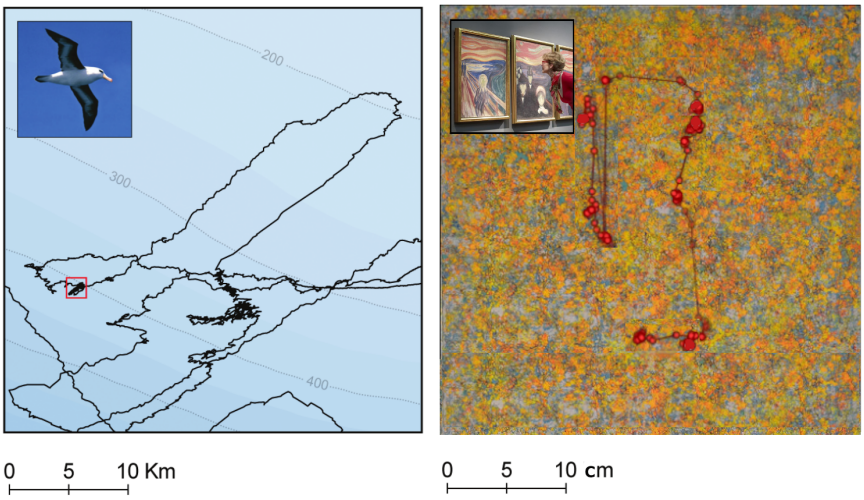
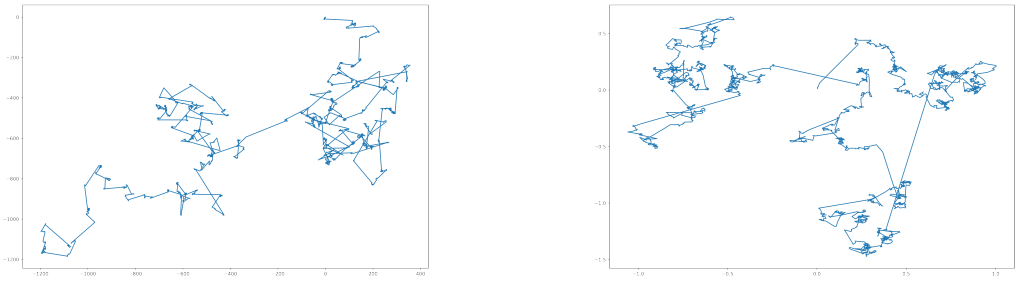


Figure 2.1: The foraging hypothesis claims that through evolution searching techniques have been optimized for survival, and therefore different searching processes in nature follow seemingly the same searching techniques. The left image is a trajectory of an albatross hunting for fish, and the right image is the trajectory of a persons eyes when examining "Artwork 71" by German artist Rainer Gross (Lencastre, 2021). Courtesy of Prof. Sergyi Denisov.

different searches in nature follow seemingly the same underlying process.

The eyes are also hypothesised to search for information in a similar manner (Brockmann & Geisel, 2000). Figure 2.1 shows the foraging strategy of an albatross searching for fish (Viswanathan et al., 1996), and the eye-gaze trajectory of a person studying a painting. Both these trajectories share the pattern of switching between longer fast movements followed by frequent small movements in one area.

Though the foraging pattern is often assumed to be universal, there is still debate in the literature on which underlying searching process the foraging hypothesis in general follow, or if there is no universal process describing the foraging dynamics (Brockmann & Geisel, 2000). This uncertainty happens partly because of the similarity between the underlying processes, and due to the lack of tools to distinguish them. The two candidates discussed in the literature are Lévy flights (Brockmann & Geisel, 1999) and intermittent processes (Bénichou et al., 2011). Figure 2.2a shows an intermittent process, while Figure 2.2b shows Lévy flights. As the figures illustrate there is a striking similarity between the two



(a) Intermittent process. **(b)** Lévy flights.

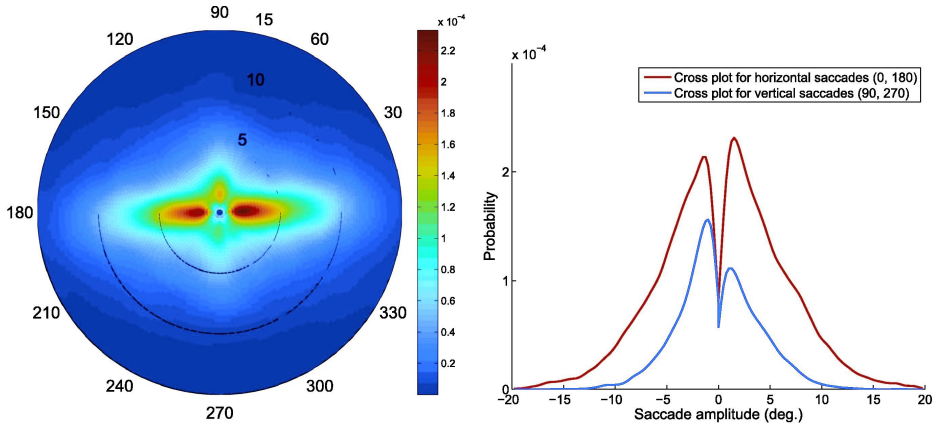
Figure 2.2: There are disagreements in the literature of whether eye-gaze trajectories follow Lévy flights or intermittent processes. These two processes can produce seemingly quite similar trajectories, however the theories behind them are fundamentally different. (a) Intermittent process. The trajectory consists of an alteration between two sub-processes: saccades and fixations, where saccades are longer jump-like movements and fixations are short random movements in a smaller area. (b) Lévy flights. The trajectory is made up of a random walk with a heavy-tailed distribution of jump lengths, determined by a parameter α, which results in movements similar to the saccades and fixations in the intermittent process.

trajectories, even though the processes behind them are fundamentally different.

The intermittent processes theorise that the eyes' trajectories consist of two types of processes, alternating between them. These two are saccades and fixations. Saccades are the fast jump-like movements over a larger distance where the eyes are moving too fast to retrieve information, while fixations are tiny movements over a small area, which allows the eyes to gather information.

Lévy flights on the other hand characterizes the trajectories as consisting of only one process, a random walk, and that the difference seen between short and long movements comes down to a heavy-tailed probability distribution of having large steps. A Lévy flight is defined by a single parameter, usually labelled α, which ranges from 1-3, that controls how frequent these extreme events are happening. An α of one means more extreme, while three is the least extreme.

Either way, whether the eyes actually follow an intermittent process or Lévy



(a) $(1.93, 1.94)$ (b)

Figure 2.3: Distribution of saccades from four openly available eye-trajectory datasets. (a) Polar plot of the probability distribution of saccadic jumps' directions in degrees, and lengths in saccadic amplitude for the next fixation given the current fixation in the centre. The eyes favour relatively short horizontal saccades, while vertical saccades are less likely, and very rarely diagonal. (b) Probability distributions for horizontal saccades (0 and 180 degrees), and vertical saccades (90 and 270 degrees). The distance is shown in saccadic amplitude. Horizontal saccades are more frequent, and short movements are most common. Figures borrowed from Le Meur and Liu (2015), Fig 3 (a) and (b), page 156. The colours of the original figure have been slightly adjusted for better and more inclusive visibility.

flight as a strategy to forage for visual information, neither of these processes seem to be able to fully capture all the advanced properties of our eye-gaze, and there are still hidden features in our eye-gaze foraging process that statistical methods are not fully able to capture.

2.1.2 Features of eye-gaze trajectories

When the eyes are searching for information, they are shown to follow a set of universal features. These features regard the relation of saccades and fixations, the lengths and directions of the movements, and that the eyes show a non-Markov behaviour.

The first of these fundamental properties of the eye movements is that the most probable saccades are of rather small horizontal movements (Le Meur & Liu, 2015).

It has been shown that we make more horizontal saccades than vertical ones, and even fewer diagonal saccades, and among the vertical ones there tends to be more saccades upwards than downwards. These properties of the saccades are all demonstrated in Figure 2.3 (a), which shows the probability distribution of saccade lengths and direction of the next fixation spot given the current fixation point located in the centre of the plot. The lengths are measured in saccadic amplitudes which are the eyes' change of degree from one fixation to the next, while the direction is shown in degrees in the image from the current fixation to the next. Figure 2.3 (b) demonstrates the probability distributions for horizontal saccades in comparison to vertical saccades, which also shows that horizontal saccades are more frequent, and shorter movements are most common.

Another central feature of the eyes was demonstrated in a study comparing searching strategies of the eyes in different difficulties of searches (Credidio et al., 2012). It was shown that even though the searching strategies changed drastically as the difficulty of the task increased, they shared a universal distribution of jump sizes. The distribution of jump sizes from the four difficulties of searches are shown in Figure 2.4. The small movements during fixations are very frequent, while the saccadic jumps are decreasingly frequent as the distance of the jumps increases.

A third universal property of the eye-trajectories is that the eyes will not refixate on a previous visited location for the first few seconds following the fixation, meaning that eye-trajectories show a non-Markov behaviour (Le Meur & Liu, 2015). This process is called the inhibition of return (IoR) and is what allows the eyes to explore a larger area while also gradually giving them the chance of returning to previously visited locations. The inhibition decreases with time and lasts for about 1.5 to 3 seconds until the probability of refixating shows no decreased probability anymore.

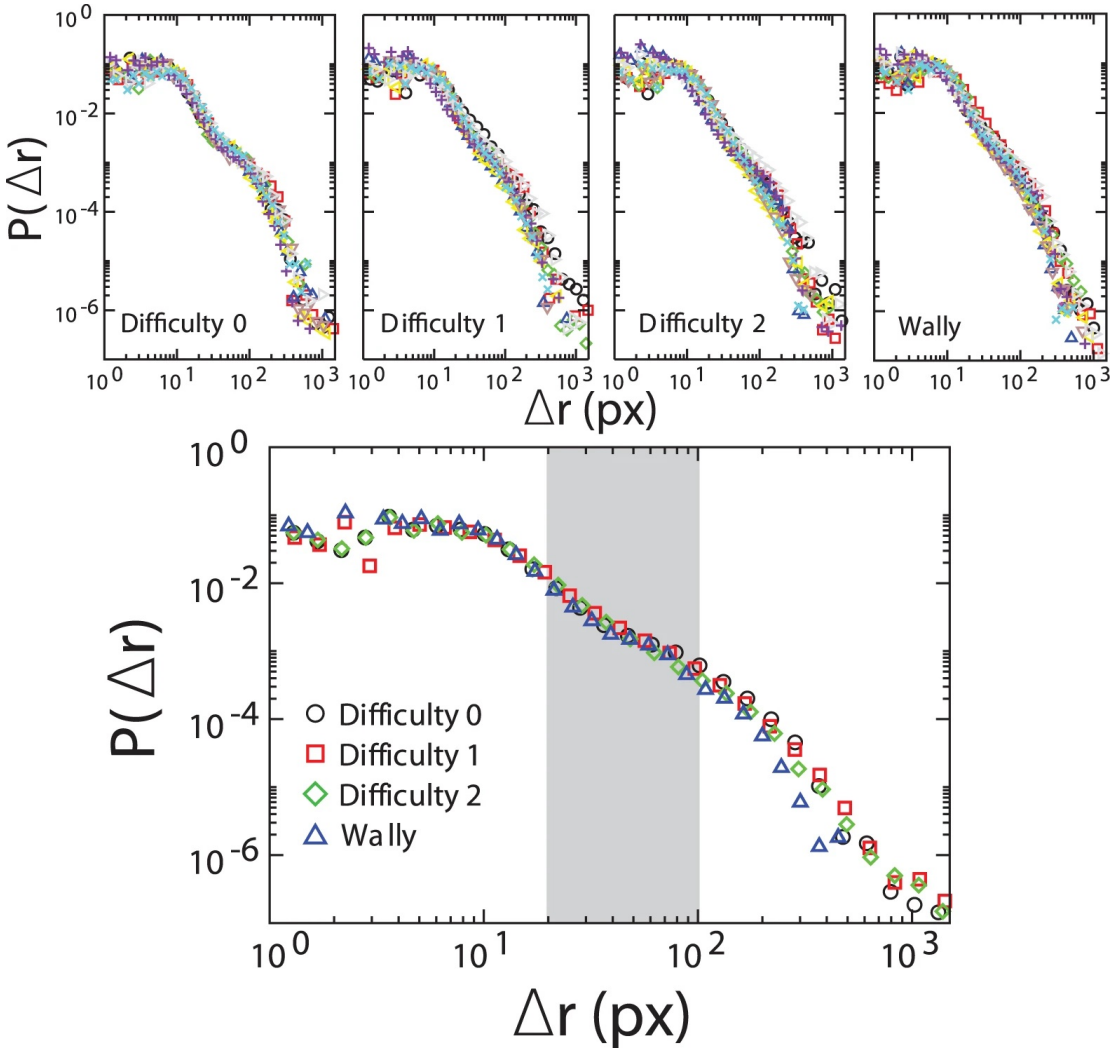


Figure 2.4: Log-log plot of the distribution of jump distances. The four upper plots shows the distribution of each person's trajectories for three different difficulties of visual searches as well as a "find Wally" search. The final figure shows all the above plots together, given the average of all subjects for each image. Even though the searching strategies the test subjects used for the four different images were shown to be drastically different, the frequency distribution of position increments is the same for all. The small movements that happen during fixations are very frequent. The saccadic movements, which are from about 100 to 10000 times larger than the fixations, are decreasingly frequent as the distances of the jumps increase. The grey shaded area in the middle marks the overlapping distances where some increments are a part of saccadic periods, and some are a part of fixation periods. Figure is borrowed from Credidio et al. (2012), Figure 3 on page 3.

2.2 GANs - history and applications

Machine Learning (ML) is the field within artificial intelligence regarding computers learning from previous experiences or previously seen data instead of being explicitly given rules for how to make decisions or behave. Machine Learning has shown to work well for solving complicated problems which have previously been very challenging to solve using only mathematical or statistical models (Jordan & Mitchell, 2015).

Within ML, the field of Artificial Neural Networks (ANNs) arose, inspired by the way the brain learns information from experience. ANNs consist of layers of perceptrons, which are simple mathematical operators inspired by the neurons in the brain, enabling them to solve even very complicated tasks. Other highly popular neural networks that followed ANNs are Convolutional Neural Networks (CNNs) (LeCun, Bengio et al., 1995) which are designed for two-dimensional data such as images, and Recurrent Neural Networks (RNNs) for sequences of data (Mikolov et al., 2014).

In 2013 Variational Auto-Encoders (VAEs) were introduced, which can be seen as one of the predecessors of GANs (Kingma & Welling, 2013). An autoencoder relies on the idea of using two networks, an encoder and a decoder, where the encoder makes a compressed representation of the data, and the decoder tries to reconstruct the data. They are trained with the goal to minimize reconstruction error. A loss function is used to compare the original to the reconstructed data, often by using mean squared error, and back propagation is used to train the network. Figure 2.5 shows a simple illustration of the autoencoder architecture. Variational autoencoders use the decoder of the autoencoder as a generator. The decoder is trained using the encoder, but when generating new samples, the decoder is instead fed random vectors. This enables the decoder to generate unique samples that share the properties of the training data. VAEs have showed useful in applications such as generating new unique levels in platform games (Sarkar & Cooper, 2021),

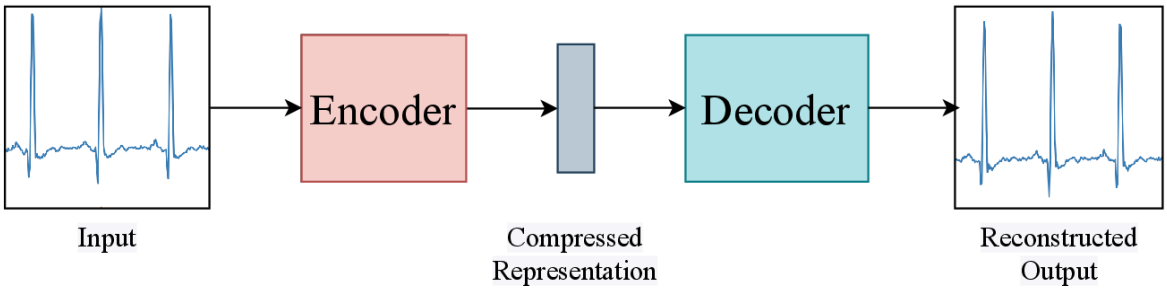


Figure 2.5: Autoencoder architecture illustration. Autoencoders consists of an encoder, who creates a compressed representation of the input, and a decoder whose task is to reconstruct the original data. A loss function is used to compare the original to the reconstructed data, and back propagation to train the network. The figure is borrowed from Brophy et al. (2021), Figure 1 on page 2.

generating traditional style poems (Li et al., 2018), and for generating faces (Hou et al., 2017). Autoencoders can also be used for generating and predictions of time-series. Gensler et al. shows how LSTMs can be used in VAEs for solar-power forecasting (Gensler et al., 2016), and Wei et al. present a framework using LSTMs and autoencoders to predict traffic flow (Wei et al., 2019).

The idea of Generative Adversarial Networks (GANs) was introduced by Goodfellow et al. (2014) and has since gained a massive popularity. GANs are a framework for training neural networks to generate data of complicated structures, with the goal of generating data with as similar data distributions to the real data as possible. From early on they were shown to be highly efficient for tasks which required to learn the data distribution of a domain of images, such as improving resolution, generating, or manipulating images (Goodfellow et al., 2014). More recently GANs have also become popular for time-series tasks (Brophy et al., 2021).

2.2.1 How GANs work

A GAN typically consists of two different networks, one generator G and one discriminator D. The generator is given a random noise vector z as input, which is essentially random noise, then generates an output which is fed to the discriminator. The discriminator will either get real data x or data generated by the generator $G(z)$

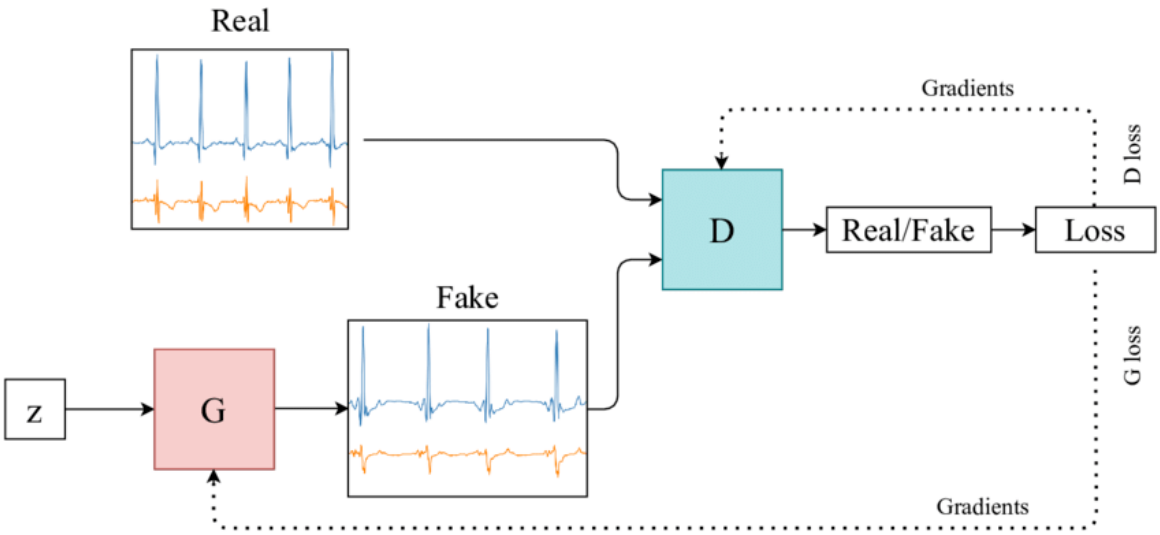


Figure 2.6: Generative Adversarial Network (GAN) architecture illustration. The generator G is given a random noise vector z as input, and outputs a fake sequence. Real and fake sequences are given to the discriminator D, who classifies them as either real or fake. The losses are calculated based on the correctness of the guesses. The G loss is used to calculate gradients for the generator and the D loss is used to calculate gradients for the discriminator. The figure is borrowed from Brophy et al. (2021), Figure 2 on page 3.

and its task is to try determining whether the data is real or fake. The two networks are trained together in a min-max game fashion. The discriminator is trained to maximize its correct labelling of the input as real of fake, while the generator tries to minimize it. Ideally, this simultaneous adversarial training will eventually lead to the generator learning to create outputs that mimic the distribution of the original data. Figure 2.6 shows an illustration of the GAN architecture.

When training a GAN it goes through iterations, called epochs, where at each epoch all the training data is separated into batches. For each of these batches the generator network generates a batch of data, which is is fed to the discriminator along with a batch of the real data from the training set. The discriminator will, for each sample, output a probability that the sample came from the real data. The minimax loss function Equation 2.1 as first described by Goodfellow et al. (2014), is then used to calculate a score of how far off the discriminator was in its predictions. After the loss of the batch is calculated for both the generator and the discriminator,

back propagation is used to adjust the different weights in the two networks.

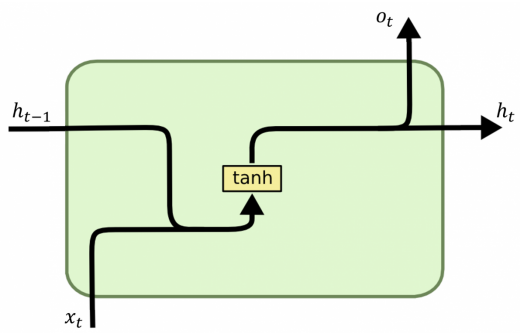
$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_{\text{generated}}(z)}[1 - \log D(G(z))] \quad (2.1)$$

In the minimax loss function (Equation 2.1), the generator G tries to minimize the output while the discriminator D tries to maximize it. Here, x represents real training data and z the random vector input that is given to G for generating. $D(x)$ is the rate of correct guesses the discriminator got on the real data, and $D(G(z))$ is the rate of correct guesses on the generated data. $\mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)]$ is the log probability of the real data x being classified as real, and $\mathbb{E}_{z\sim p_{\text{generated}}(z)}[1 - \log D(G(z))]$ is the log probability p that the generated data is classified as fake.

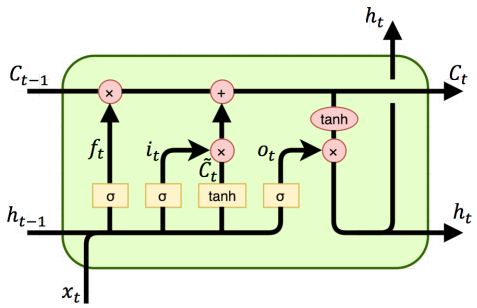
2.2.2 Recurrent Neural Networks

Since most of the early examples using GANs were for tasks such as image-generation, the type of networks used in GANs are most often CNNs (Goodfellow et al., 2014). CNNs are designed to work well for images and other data where the features have relations across several dimensions of the data, such as in images where each pixel has relations to nearby pixels in both horizontal, vertical, and diagonal directions (LeCun, Bengio et al., 1995). When generating sequential data, it is also possible to use methods like CNNs, by for example representing the sequence as a one-dimensional grid. However, these methods do not consider the data as sequences across time but rather as fixed inputs. CNNs are also normally not suited for sequences of varying lengths since the input and output sizes generally have to be of a fixed size. Instead, to operate on sequential data, a faster and more appropriate type of methods are Recurrent Neural Network (RNN)s (Mikolov et al., 2014).

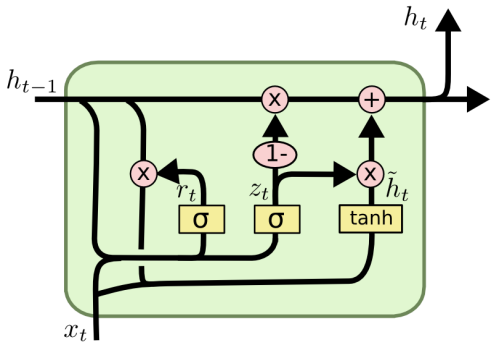
RNNs are a class of neural networks made for sequential data, that can be used to predict coming values in a series of data. They can take varying sequence



(a) RNN cell architecture



(b) LSTM cell architecture



(c) GRU cell architecture

Figure 2.7: RNN, LSTM and GRU cell architectures. RNN networks consists of operators that the data are passed through. RNN cells, LSTM cells and GRU cells are three popular operators used in RNNs. All three figures are borrowed from **Lopez2019RNNGRU** (**Lopez2019RNNGRU**)

lengths, unlike other simpler networks which only take fixes-sized inputs. An RNN originally consists of operators called RNN cells (Elman, 1990). An RNN network can consist of one or more layers of RNN cells. In addition to the recurrent layers, the networks have an input and an output layer.

Figure 2.7a shows the architecture of a single RNN cell. The output state h_t at time step t is calculated as shown in Equation 2.2. The inputs are the state output from the previous cell, h_{t-1}, as well as the current input data x_t. It calculates the weighed sum of the inputs using the adjustable weights U and V and a bias vector b_h, and uses an activation function σ which for RNNs is normally Tanh activation function.

$$h_t = \sigma(U_h x_t + V_h h_{t-1} + b_h) \tag{2.2}$$

RNN cells suffer from the vanishing gradient problem. Each cell only gets the output state from the previous time step as input, and this state was largely dependent on the input vector to said cell. At every time step the states from earlier in the sequence will be a smaller part of the equation, at an exponential rate. Since the most recent data points in the sequences will have contributed the most to the output, those will affect the gradient the most, and hence the adjustment of the weights. Due to this, RNN cells are not able to learn long-term dependencies.

Long Short-Term Memory (LSTM) is a type of RNN which uses LSTM cells, and was created to tackle the problem of lacking long-term dependencies in RNNs (Hochreiter & Schmidhuber, 1997). Nowadays most RNNs are LSTMs.

Figure 2.7b shows the architecture of an LSTM cell. LSTM cells have two hidden states: the hidden state h_t which is similar to the hidden state of the RNN cell and can be seen as the short term memory in LSTM, and the cell state C_t which can be seen as the long term memory of the network. Through the cell state C_t, information from even the earliest time steps can be carried through to the last time step. Information is added to and removed from the cell state via the input gate i and the forget gate f. The third gate, the output gate o, is used to calculate the hidden state h_t.

The cell state of the LSTM is calculated by Equation 2.3a where f_t is Equation 2.3b, i_t is Equation 2.3c, and $\hat{C}_t$ is Equation 2.3d. h_{t-1} is the hidden state from the previous cell, x_t is the current input vector, b are the biases, and W the adjustable weights. To calculate the next hidden state h_t, we multiply the cell state C_t with the output o_t, giving $h_t = \tanh(C_t)o_t$. The output o_t is given by $o_t = \sigma(h_{t-1}W_{ho} + x_t W_{xo} + b_o)$.

$$C_t = C_{t-1} f_t + i_t \hat{C}_t \tag{2.3a}$$

$$f_t = \sigma(h_{t-1} W_{hf} + x_t W_{xf} + b_f) \tag{2.3b}$$

$$i_t = \sigma(h_{t-1} W_{hi} + x_t W_{xi} + b_i) \tag{2.3c}$$

$$\hat{C}_t = \tanh(x_t + h_{t-1}) \tag{2.3d}$$

Even though LSTMs are able to tackle the long-term dependency problem of traditional RNNs, they also present a new issue, namely the increased size of the network and time it takes to train due to the increased complexity of each cell. Another more recent cell type is the Gated Recurrent Unit (GRU) presented in 2014 (Chung et al., 2014). GRU cells are essentially a simpler and more compact version of LSTM cells, and have been shown to work about as well as LSTMs.

GRU cells only carries one hidden state h_t, not two like in the LSTM. Figure 2.7c shows the architecture of a GRU cell. Instead of the three gates of the LSTM, GRUs has a reset gate r to decide how much from the previous state to remember, and an update gate z to decide how much from the previous state to forget.

The reset state r at time t is calculated by $r_t = \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r)$. The update state z is $z_t = \sigma(x_t W_{xz} + h_{t-1} W_{hz} + b_z)$. The candidate hidden state is $\hat{h} = \tanh(x_t W_{xh} + (r_t h_{t-1}) W_{hh} + b_h)$. Then finally the hidden state is calculated by $h_t = h_{t-1}(1 - z_t) + z_t \hat{h}_t$.

In addition to the GRU cells presented here, there has also been proposed several other alternatives to the LSTM cell. The LSTM cell is however still the best performing cell except from a few other cell architectures such as the GRU which gives comparable results, according to a review by Greff et al. (2016).

There is no clear answer to which one of GRU or LSTM one should choose, this all depends on the problem and the data. Since the GRU cells have fewer parameters, they work well for making faster and more compact models. For data

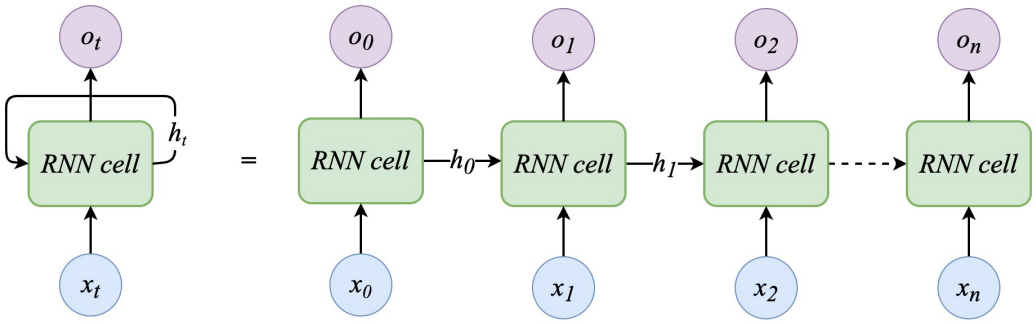


Figure 2.8: Unfolded Recurrent Neural Network. An illustration of how we can think of the RNN cell as a series of cells, one for each time step in the input sequence. The figure is an example of a single layered, non conditional RNN utilizing an RNN cell. For every data sample x in time step t in the current training data sample in the batch, x_t is fed to an RNN cell along with the output from the previous cell h_{t-1}, and o_t is the output at each time step.

with simpler structure and shorter sequences GRU cells can work very well and be trained faster than with LSTM cells. For longer sequences LSTMs should in theory be superior, as they can also capture long-term dependencies in the sequences (Y. Yu et al., 2019).

Whether using RNN, LSTM or GRU cells, the recurrent neural networks can consist of one or more recurrent layers, also called hidden layers, which are the layers consisting of these operator cells (Salehinejad et al., 2017). For multilayered RNNs, the output from each time step in the first layer will be fed to each corresponding cell in the next layer as input along with the output of the previous cell.

Figure 2.8 shows a visualisation of an unfolded one layered RNN, where we can see that for each x_t in the current training data sample of length n, the samples are fed into an RNN cell along with the output h from the cell that x_{t-1} was fed into. In case of LSTM cells instead or RNN cells, the unfolded version will still be the same, except that both the output state h_t and the cell state C_t are passed on to the next cell.

Figure 2.9 is an illustration of a multilayered RNN. For every cell in the first layer,

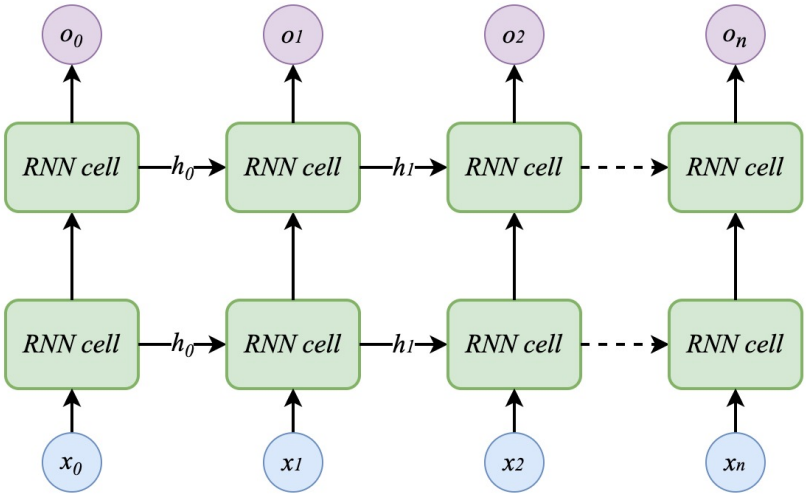


Figure 2.9: Multilayered Recurrent Neural Network. The figure shows an illustration of an unfolded multilayered RNN with two layers. The output from every cell in the first layer is fed as input to the corresponding cell in the second layer, along with the output h from the previous cell in the same layer.

its output o_t is fed as input to the corresponding cell at t in the second layer, along with the output from the previous cell in the second layer h_{t-1}. The advantages of using multiple layers in RNNs is that this can allow the network to learn more complex structures, since each layer will have have its own adjustable weights.

An RNN can also be bidirectional, meaning it passes the information in both directions of the network (Schuster & Paliwal, 1997). Using a bidirectional RNN means that the output at each time step is both determined by previous and future data. In many types of data current elements in the sequence is highly dependent on future steps, such as in NLP where a word can have different meanings based on which words come later in the sentence. Bidirectional RNNs open a way for RNNs to better work for these types of data. Figure 2.10 shows an illustration of how bidirectional RNNs works. The network consists of one forward passing layer and one backward passing. The outputs from the cells in the two layers are concatenated for the final output.

An RNN can also have multiple bidirectional layers. In this case the forward- and backward-passing layers seen in Figure 2.10 will be considered as one

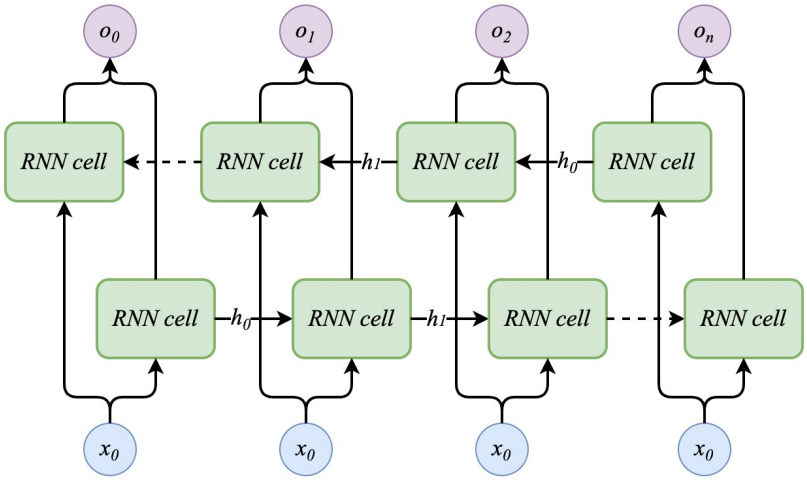


Figure 2.10: Bidirectional Recurrent Neural Network. The network has one forward passing layer and one backward passing layer, where the sequence is taken in the opposite order. The outputs of the two layers are concatenated, so the output at each time step t in the forward passing layer is concatenated with the output at step $n - t$ in the backwards passing layer

bidirectional layer, and multiple of these can be stacked to create a multilayered bidirectional RNN.

2.2.3 GANs using RNNs

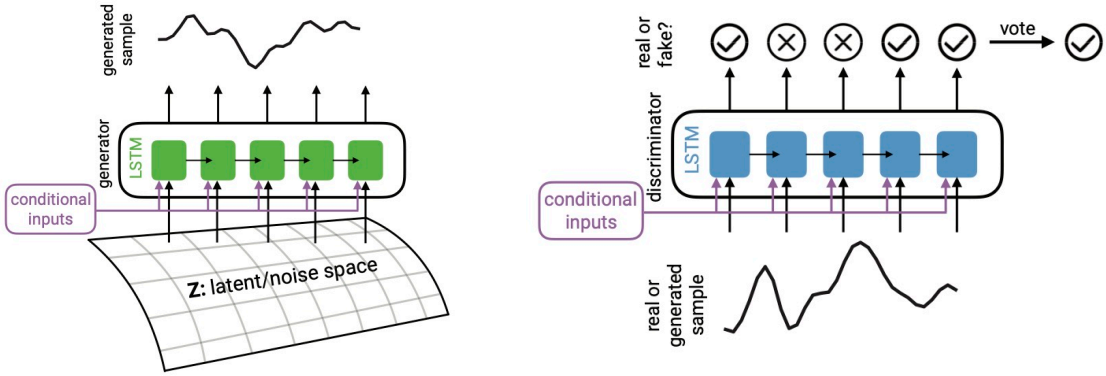
Though initially used in the context of image recognition and replication, GANs have more recently been used in the context of time-series (Brophy et al., 2021). GANs for time-series usually utilizes an RNN network for the generator, mostly using LSTM or GRU cells. For the discriminator they can use either an RNN or a CNN. According to Brophy et al. (2021) there has not yet been established in the literature a standard way of training or evaluating GANs for time-series, unlike with the traditional CNN-based GANs where there is more agreement and established methods.

As with other GANs, Recurrent GANs can also be either conditional or unconditional. This means that the model either is given a condition as input in addition to the noise vector, both during training and generating, or is only given the noise vector as input (Esteban et al., 2017a). Unconditional GANs are used for generating data that has the same distribution as the training data, and will only be

able to produce general samples of that specific domain. Meanwhile, the conditional GANs can be for a range of different problems: for problems where the generated samples should be of certain categories within a dataset, where it is trained on a labelled dataset using the label as a condition; for filling in missing data in a sample, where its trained on sets of complete data and its incomplete counterpart; or for predicting the future steps in a sequence, which is the primary aim of many time-series GANs.

Figure 2.11 is an illustration of a simple RNN GAN architecture. Both the generator and discriminator are LSTM networks. Figure 2.11a shows the generator, which is given both random noise and a condition as input, and outputs a sequence. Figure 2.11b is the discriminator, which takes either a generated sequence or a real sequence from the training set as input, as well as the condition, and tries to determine if the sequence was real or not.

RNN GANs learn using the same concepts as other GANs, as described in subsection 2.2.1. At each batch of data, the loss for each of the networks are calculated, often by using some sort of minimax function, such as the original minimax function in Equation 2.1. The weights of both the networks are then updated using back propagation through time (Brophy et al., 2021). How much the weights are updated at each epoch is determined by a learning rate. A higher learning rate will allow the network to learn faster, but if it is too high it can lead to unstable training and to not reaching optimum weights (Salehinejad et al., 2017). This process is repeated for all the batches in the training data, which constitutes one epoch in the training. The GAN will most likely need several epochs to learn to generate realistic data. The number of epochs must be determined based on factors such as the amount of training data, the size of the network, and the computational resources available.



(a) Generator in RNN GAN.

(b) Discriminator in RNN GAN.

Figure 2.11: Architecture of a conditional RNN GAN. In this example both the generator (a) and the discriminator (b) are LSTM networks. The generator (a) is given a condition and random noise as input, and outputs a sequence. The discriminator (b) is given either a generated sequence or a real sequence, and its condition, and it tries to determine whether the sequence is real or not. Figures are borrowed from Esteban et al. (2017a), Figure 1 on page 4.

2.2.4 Transformers

In addition to RNNs, there have also recently come other more sophisticated methods for handling sequential data. Transformers (Vaswani et al., 2017) are a newer type of model for discretely represented time-series data which is created to handle large domains of data. They were initially created for Natural Language Processing (NLP), and have shown a high performance in tasks such as language understanding and machine translation. Transformers are even used in Google Translate, and is one of the reasons that the translator has reached its good performance in the recent years (Caswell & Liang, 2020). Transformers have later also been used in other domains of discrete sequential data, such as music generation (Muhamed et al., 2021a; N. Zhang, 2020).

A variant of transformers are Bidirectional Encoder Representations from Transformers (BERT). BERT is a variant of transformers that are made to have ”attention” in both directions of the sequence (Devlin et al., 2018), unlike the regular Transformers which are said to have ”attention” in one direction of the sequences.

They both achieve attention by using 'attention blocks' which can connect any data in the sequence to one another, hence eliminating the long-term dependency issue found in RNNs. BERT are seen by many as having revolutionized the NLP field (Chernyavskiy et al., 2021).

The drawback of Transformers and BERT though is that they are only made to deal with discretely represented data, such as discrete tokens for the different words in a language (Vaswani et al., 2017). This may limit which problems areas outside of NLP they will be ideal for.

2.3 State-of-the-art GANs for time-series

GANs made for time-series data using either RNNs or Transformers have shown to be able to replicate data in a range of contexts such as generating biomedical signals (Hazra & Byun, 2020), predicting financial time series (Wiese et al., 2020), Natural Language Processing (L. Yu et al., 2017), predicting pedestrian trajectory (Lv et al., 2021), and for generating music (Dong et al., 2018; Mogren, 2016)

In this section we will give an overview of the recent status of time-series GANs research. Within the time-series GAN literature, the problems and the type of data the models aim to generate are of a wide variety. We start off by presenting a range of GANs utilizing RNNs and CNNs for different representations of data and for a variety of problems. Then we go into a variety of Transformer GAN architectures and their applications. Finally, we look at research on generating eye-gaze trajectories and foraging trajectories using Machine Learning, in particular ones using GANs, to assess the status of existing literature with similar research questions as this project.

2.3.1 Time-series GANs

GANs made for time-series can be categorized based on the representation of the data they aim to generate. Sequential data can be of either a continuous or

a discrete representation. When data is continuous, all consecutive values are assumed to have a numerical relation, and that there is also a continuous numerical relation for any value between two consecutive values. However, when the data is discrete, each value represents some fixed meaning in the data domain, and there is not necessarily a relation between the numerical values (Brophy et al., 2021). An example is from NLP, where this numerical representation is normally called a token (Cartuyvels et al., 2021). In a tokenized sentence, the numerical token representations can be close numerically even if the words are unrelated. This is because the tokens are primarily just a way to represent the words numerically, so that they can be understood by ML models, and to separate between identical words with different meaning, such as the difference between the animal “bear” and the verb “to bear”.

The time-series GANs in the literature are either created for continuous representation or discrete representation of data (Brophy et al., 2021). It is therefore essential to choose a model wisely based on the representation one wishes to use. For some data domains it has become established in the literature which representation to use, like in NLP, while in other domains like music there is still not an agreement on which representation is best suited. Recent literature has also shown promising results by combining continuous and discrete representation in the same models (Cartuyvels et al., 2021).

Cartuyvels et al. (2021) emphasizes that most deep learning models are made to handle continuous representation, and thus that continuous time-series GANs are often seen as the most convenient to use. Transformer models, from NLP, which were presented in subsection 2.2.4 is however an example of a category of models made specifically for discretely represented data. We present some recent state-of-the-art Transformer GAN models in subsection 2.3.2

In 2017, L. Yu et al. (2017) presented SeqGAN, which is a discrete time-series GAN which contributed a lot to the further development of sequential GANs.

SeqGAN was trained to generate sequences of tokens, and was primarily made for NLP, but also tested on music. The generator is an LSTM, and the discriminator is a CNN. L. Yu et al. points out that giving GANs' generators random noise vectors as input does not make sense for discrete time-series. The noise vectors are there to ensure nondeterministic outputs and works because they lead to slight changes in output values in the first layer of the model. Instead, their suggestion is to feed the generator with its already generated sequence as a current state input rather than giving noise vectors. The SeqGAN generator is essentially trained to only generate the next step in the sequence, but by repeating this for the output it already created enables it to generate long sequences. SeqGAN uses an LSTM as the generator, but L. Yu et al. point out that the model works with GRU cells as well.

QuantGAN was created by Wiese et al. (2020) for capturing long-range dependencies in financial time-series with a discrete representation of the data. QuantGAN uses Temporal Convolutional Network (TCN) which is a convolutional network with skip connections, for both the generator and the discriminator. Their results show that TCN can work better than recurrent networks using GRUs or LSTMs for certain time-series generation tasks.

Among the first examples of GANs for generating continuous sequential data was C-RNN-GAN, made in 2016 by Mogren (2016). They attempted to generate music, by representing each time step using four continuous values: tone length, frequency, intensity, and time spent since the previous tone. This let them represent several tones at a time, which enabled C-RNN-GAN to generate polyphonous chords. The model consists of two RNNs using LSTM cells, where the discriminator is bidirectional, and the generator is unidirectional, and is trained using the minimax loss function Equation 2.1 as described in subsection 2.2.1. Mogren explained that during training the discriminator tended to get too strong, so freezing was applied when the discriminator loss became much smaller than the loss of the generator, to prevent the training from stagnating.

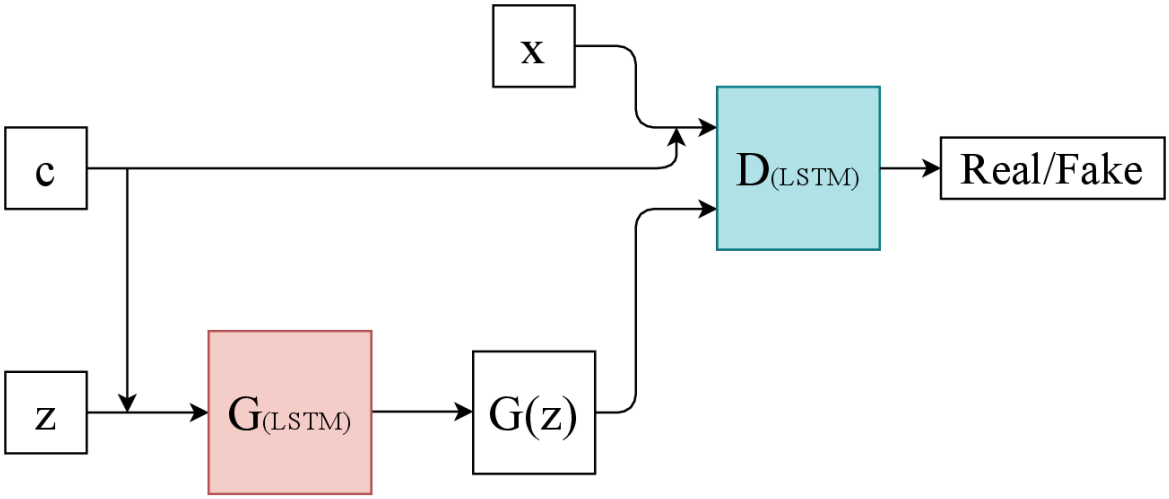


Figure 2.12: RCGAN architecture. RCGAN is a conditional RNN GAN consisting of two LSTMs. The Generator G is given noise z and a condition c as input. The discriminator d is given either real data x and its condition c, of the genrators output $G(z)$, as input, and classifies the data as either real or fake. The figure is borrowed from (Brophy et al., 2021), Figure 8 on page 10.

To evaluate the output of C-RNN-GAN, four measurements within music theory were used: polyphony, scale consistency, repetitions, and tone span. For these measurements the GAN seemed to perform well and generated increasingly complex music as training proceeded, however Mogren stated that by human judgement the generated music is not comparable to the training data music.

Recurrent Conditional GAN (RCGAN) (Esteban et al., 2017a) is a model for replicating multi-valued time-series of medical data by Esteban et al. given underlying states of the patients as conditions. As with C-RNN-GAN, RCGAN is designed for continuous sequences, but RCGAN is conditional and hence takes a sequence as input and predicts a continuance of the sequence. Esteban et al. aimed to generate synthetic datasets of realistic time-series of otherwise sensitive data, such as in medicine, to overcome the privacy concerns of using real datasets in research. RCGAN uses LSTMs for both the generator and the discriminator. Figure 2.12 is an illustration of the RCGAN architecture, which follows the general RNN GAN architecture as describe in subsection 2.2.3. The authors, Esteban et al., also propose a method for evaluating time-series GANs called “train on synthetic,

test on real", as a way of validating the usefulness of the generated data. They train a classification model on data generated by the GAN, of two or more classes, and then test if the classification works on classifying the real data.

Another continuous GAN is Sequentially Coupled GAN, SC-GAN, which was made for estimating medication dosages for individuals (Wang et al., 2019). SC-GAN uses two generators, where the first one is given the current state of an individual as condition, and then the output is fed to the other generator for generating recommended medication dosage. SC-GAN was able to perform close to real data, and for its intended usage it outperformed SeqGAN, C-RNN-GAN and RCGAN.

TimeGAN (Yoon et al., 2019a) was demonstrated to perform higher than several state-of-the-art GANs, among others C-RNN-GAN and RCGAN. Time-series GAN (TimeGAN) Yoon et al., 2019a combines an autoencoder architecture, which is demonstrated in Figure 2.5, with a GAN architecture. In total TimeGAN consists of four networks, which are an encoder and a decoder, constituting the autoencoder, and an RNN as generator and a bidirectional LSTM as discriminator of the GAN architecture. Instead of giving the discriminator the original real data as input to classify it as real or fake, it is given the encoded vector of the real data. The architecture of TimeGAN is illustrated in Figure 2.13.

TimeGAN uses three losses to control the training: Reconstruction loss; supervised loss; and unsupervised loss. The reconstruction loss is calculated using the output of the decoder, and is used to calculate the gradients for the encoder and the decoder. The unsupervised loss is based on the discriminators output and, similarly to the minimax loss function of the original GAN (subsection 2.2.1), is used for calculating the generator and discriminators gradients. Finally, the supervised loss is calculated based on both the generators output and the encoder output of the real data, and is used for calculating the gradients of the generator and the encoder.

Conditional Sig-Wasserstein GAN, SigCWGAN (Ni et al., 2020a) addresses the

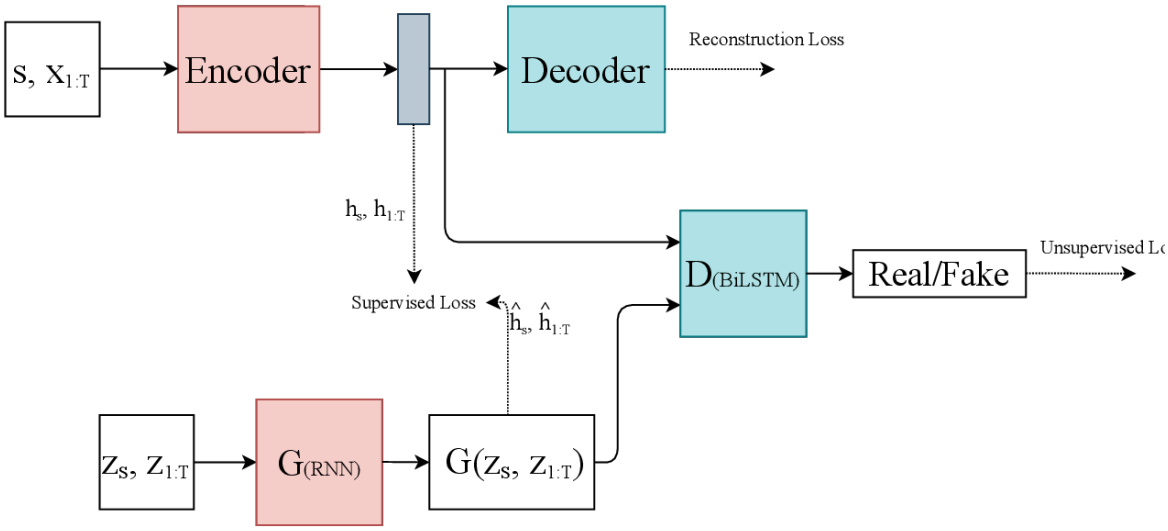


Figure 2.13: TimeGAN architecture. TimeGAN combines an autoencoder architecture with a GAN architecture. It consists of four networks: an Encoder, a Decoder, an RNN generator G and a Bidirectional LSTM (BiLSTM) discriminator D. The encoders task is to encode tuples of $S, X_{1:T}$ to a latent representation. S are statistical features of the data, such as a category, X are the temporal features as a sequence, and T is the sequence length. $(Z_S, Z_{1:T})$ are the noise vectors given as input to G, which generates $G(Z_s, Z_{1_T})$. $h_s, h_{1:T}$ are the latent output from the Encoder, and $\hat{h}_s, \hat{h}_{1:T}$ are the synthetic latent vectors of $G(Z_s, Z_{1_T})$. The figure is borrowed from (Brophy et al., 2021), Figure 11 on page 12.

problem of capturing temporal dependencies in time-series data. When creating SigCWGAN, Ni et al. (2020a) aimed for the model to also be able to predict for "tails of data", by which they mean data that are sparse in the dataset. To capture this, they integrate GANs with the "signature of a path" extraction, for which they create a discriminative metric C-Sig-W_1. They also propose a generator, called Autoregressive Feed Forward Neural Network (AR-FNN).

The Autoregressive Feed Forward Neural Network (AR-FNN) generator aims to capture auto-regressive processes. The model is a three layer feed forward neural network. The network is designed to predict the next step of a given sequence. To generate a sequence, the process is repeated over and over where for every new iteration the previously predicted sample is added to the condition.

Instead of an RNN networks as discriminator which is commonly seen, they

propose a C-Sig-W_1 metric. This aims to capture the temporal dependencies in time series, and not only the data distribution which can be a limitation of other discriminators. They show that the C-Sig-W_1 metric works well to capture the auto-correlation functions of the data.

Ni et al. has also implemented a Conditional Wasserstein GAN (CWGAN) (Ni et al., 2020b), which does not use their "signature of the path" metric C-Sig-W_1. They implemented it using AR-FNN as both the generator and the discriminator. In addition, Ni et al. implemented a recurrent version of CWGAN, which uses RNNs for both the generator and discriminator. This version is called Recurrent Conditional Wasserstein GAN (RCWGAN).

In 2020 Hazra and Byun (2020) presented SynSigGAN, a GAN designed for generating any kind of synthetic biomedical signal. Similarly to what we address in this thesis project, Hazra and Byun address the issue that when using medical data in research there is often a confidentiality concern regarding the data. Therefore, generating synthetic data with the same properties as the training data would have many usages in research. Often though, medical data with certain properties can be limited, and the design of SynSigGAN is therefore made to be able to generate data of similar properties using only a small set of real data.

The SynSigGAN generator is a bidirectional grid LSTM. A GridLSTM (Kalchbrenner et al., 2015) is an improved version of LSTM that can deploy cells along any dimension of the network, not only the temporal. The discriminator is a one dimensional CNN. SynSigGAN was tested on four types of biomedical signals: electrocardiogram, electroencephalogram, electromyography and photoplethysmography. Hazra and Byun show in their evaluation that SynsigGAN performs better than existing models for their problem.

Since the field of GANs generating time-series data is still very much in the research phase, it remains in the literature to agree upon what evaluation metrics to use to best evaluate the generated data, according to Brophy et al. (2021).

Due to this, the models are compared to problem-specific measures made by the researchers in the literature. When researchers then compare their model to other models, many tend to stick to the self-made, domain-specific evaluation metrics, hence the comparisons is only domain specific and cannot necessarily be said to apply in other research questions or with other data (Brophy et al., 2021).

2.3.2 Transformer GANs

The popular Transformers (Vaswani et al., 2017) and BERT (Devlin et al., 2018) have the recent years been attempted used in GANs for a range of tasks. Transformer GANs have been attempted used in music generation (Muhamed et al., 2021a; N. Zhang, 2020), trajectory forecasting (Giuliari et al., 2021; Lv et al., 2021), and even for non-sequential tasks within computer vision (Jiang et al., 2021; Lin et al., 2018). Jiang et al. (2021) shows how the use of two transformers as the adversarial networks of a GAN can generate realistic images, and they propose this as a method as opposed to using CNNs in GANs for computer vision tasks.

For trajectory forecasting, Lv et al. (2021) made a transformer GAN for predicting the possible future trajectories of multiple pedestrians in a scene, for usages such as self-driving cars and robots. Their model is able to generate several trajectories at once.

Muhamed et al. (2021a) recently were able to generate high quality, realistic music using a Transformer GAN. They trained their model in the GAN fashion, using a Transformer as the generator and a BERT as the discriminator. When making humans score their Transformer GAN generated music to the current state-of-the-art Transformers for music generation, their music was shown to be preferred by the test subjects.

2.3.3 GANs for eye-gaze and foraging trajectories

Using ANNs to replicate eye-gaze trajectories is not a new concept. In 2011, years before GANs were proposed as a way to train generative models, Y. Zhang et al. (2011) made a neural network for predicting human interest spots in images. The network was trained to predict the next point following a given sequence from a fixation period in an eye-gaze trajectory. To generate a trajectory, they repeated the generation process while periodically adding saccades to imitate the saccadic movement from one fixational period to the next.

Pan et al. (2017) proposed SalGAN, which uses a CNN based GAN to predict saliency maps of where in an image a person would look. They were only concerned with predicting where in an image people would look, and not in which order, making it very different from the problem addressed in this thesis project.

Assens et al. (2018) introduced PathGAN, which is a conditional RNN GAN for predicting where in an image people will look and in what order. PathGAN generates trajectories, making it more similar to what we wish to achieve compared to SalGAN. However, the image that is given as condition for the PathGAN model to predict the eye-gaze trajectory is a very central part of this model. However, the foraging process of the eyes, which is what we aim to capture in this thesis project, is not a focus of PathGAN.

Moving away from eye-gaze trajectories onto foraging trajectories, which has similarities with human eye-gaze trajectories when foraging for visual information (subsection 2.1.1), are Roy et al. (2021) who use GANs with LSTMs and CNNs in various combinations to generate the trajectories of animals foraging for food (Roy et al., 2021). Unlike the previously mentioned networks that predicted eye-gaze trajectories based of where in an image a person would look, the project by Roy et al. can be considered more similar to the problem we address in this thesis, due to the similarity of foraging trajectories and eye-gaze trajectories when searching for information.

As shown, GANs have been used in replicating eye-movements of humans looking at images to predict where they will look (Assens et al., 2018), and GANs have also been used to replicate trajectories of animals foraging for food (Roy et al., 2021). However, as far as we are aware, there has not been conducted any work in replicating eye-gaze trajectories as sequences by the use of Generative Adversarial Networks for the purpose of replicating the process of which our eyes search for information.

Chapter 3

Methodology

This project aims to test a range of GAN models for time-series to see if they can be suitable for generating realistic eye-gaze trajectories of the eyes' foraging process. In order to study the limitations of each model, we conduct the experiments in four stages, with increasingly complex data sets for the GAN models to replicate.

We chose to use Vector Autoregressive (VAR) generated data for our first two stages of our experiments. VAR data, both in one and two dimensions, have been used as a demonstrative example by Ni et al. (2020a) and Yoon et al. (2019a) in their comparisons of a range of models. The one dimensional VAR data has a short temporal dependence, as in that the current movement is dependant on the recent past movements. The two dimensional VAR data has both the temporal dependency and a feature dependence across the two dimensions, making the process slightly more complex.

The third data we will use is intermittent processes, which we assume will be more difficult for the models to replicate than the two dimensional VAR data. Intermittent processes consist of two different processes, saccades and fixations, which they change between periodically (Land, 2019). Replicating intermittent processes will require the models to be able to learn longer dependencies than

the two dimensional VAR data, hence be more challenging.

Finally, we will use real eye-gaze trajectories. Due to the similarities between intermittent process and eye-gaze trajectories, we assume that the models performing well for intermittent processes will be the same to perform well for real eye-gaze trajectories. These does however have a more complex process than the underlying intermittent process, so there might be discrepancies.

This chapter first explains how we generate the three datasets of synthetic data. Then we present the real eye-trajectory dataset we are using. We will then go into explaining the experiments we conduct, which GAN models we chose to use in the experiments, and which evaluation metrics we will use to evaluate the results of the different experiments.

3.1 Generating synthetic data

For the first three experiments we use two types of synthetic data: Vector Autoregressive (VAR) generated data, and intermittent processes. The VAR data we will be using has previously been used as an illustrative example both by Yoon et al. in TimeGAN (Yoon et al., 2019a), and by Ni et al. in SigCWGAN (Ni et al., 2020a) when comparing a range of different time-series GAN models. The second type of synthetic data we use are intermittent processes, which as discussed in subsection 2.1.1 is a type of process that has similarities to the process of eye-gaze trajectories when foraging for information.

For a one dimensional (1D) VAR process, each new state X_t is calculated by Equation 3.1a with Equation 3.1b where ϵ is a normal distributed variable with mean 0 and standard deviation 1. We define ϕ as the temporal correlation of the time series and σ is the feature correlation.

$$X_t = X_{t-1} + \delta X_{t-1} \tag{3.1a}$$

$$\delta X_{t-1} = \phi \times \delta X_{t-2} + \epsilon \tag{3.1b}$$

For a two dimensional (2D) VAR process, each new state $X(t), Y(t)$ is calculated by Equation 3.2a with Equation 3.2b and Equation 3.2c where ϵ is a normal distributed variable with mean 0 and standard deviation 1.

$$X_t, Y_t = (X_{t-1} + \delta X_{t-1}, Y_{t-1} + \delta Y_{t-1} \tag{3.2a}$$

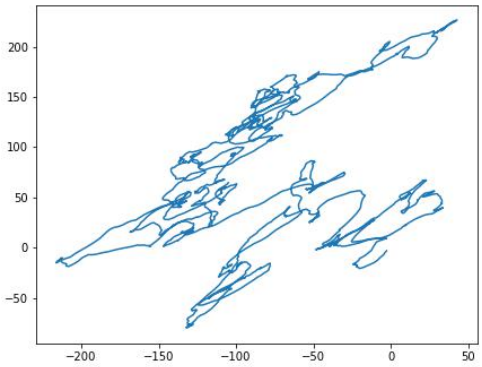
$$\delta X_{t-1} = \phi \times \delta X_{t-2} + \sigma \times \delta Y_{t-2} + \epsilon \tag{3.2b}$$

$$\delta Y_{t-1} = \phi \times \delta Y_{t-2} + \sigma \times \delta X_{t-2} + \epsilon \tag{3.2c}$$

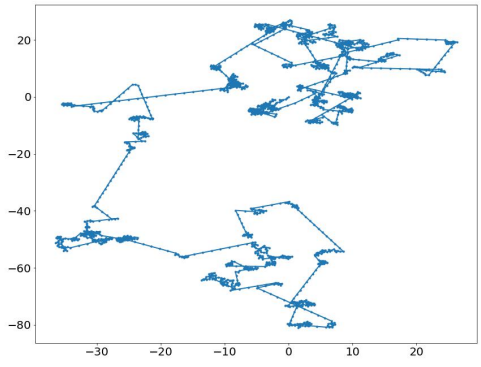
The VAR data was generated using the implementation by Ni et al. (2020a). The code to generate the data is shared in their GitHub project (Ni et al., 2020b). We generate a set of one dimensional data, using Equation 3.1, with σ = 0.8. For the second experiment we use Equation 3.2, also with σ = 0.8 and ϕ = 0.8. Figure 3.1a is an example of a one dimensional trajectory from our generated dataset, and Figure 3.1b of a two dimensional. The reason we choose 0.8 as the values for σ and ϕ is because these values were used by Ni et al. (2020a) when comparing a range of different models' performances. Therefore, by using the same values, we expect to see similar results as they did for the experiments where we use this data.

To generate the synthetic intermittent processes, the processes are started with either a fixation period F or a saccadic period S, decided on random. To determine the duration of each saccade and fixation, we define a rate λ for changing from one process to the other. λ_{SF} is the rate of transition from saccade to fixation, and λ_{FS} is the rate of transition from fixation to saccade. At τ time-interval, $P_{SF} = e^{-\lambda_{SF}\tau}$

(a) VAR 1D



(b) VAR 2D



(c) Intermittent process

Figure 3.1: One plotted example of a generated synthetic data of each of the data types used in the first three experiments. (a) VAR 1D shows the value (x axis) over time (y axis). (b) and (c) show the flattened plot of the x and y values of the data across a sequence of 2000 time steps.

is the probability of changing from a saccade to a fixation, and $P_{FS} = e^{-\lambda_{FS}\tau}$ is the probability of changing from a fixation to a saccade.

A saccade has one constant angle for the whole duration of the saccadic period, chosen at random, and the length of the saccade that was determined by P_{SF}. The step length in a saccade is determined by a constant saccadic velocity.

For a fixation every new time step in the duration, given by P_{FS}, is determined by a new random angle, and a step length. The step lengths are normally distributed

with a mean 0 and variance given by a diffusive parameter D. Figure 3.1c is a plot of a 2000 time steps long generated intermittent process from the generated training dataset.

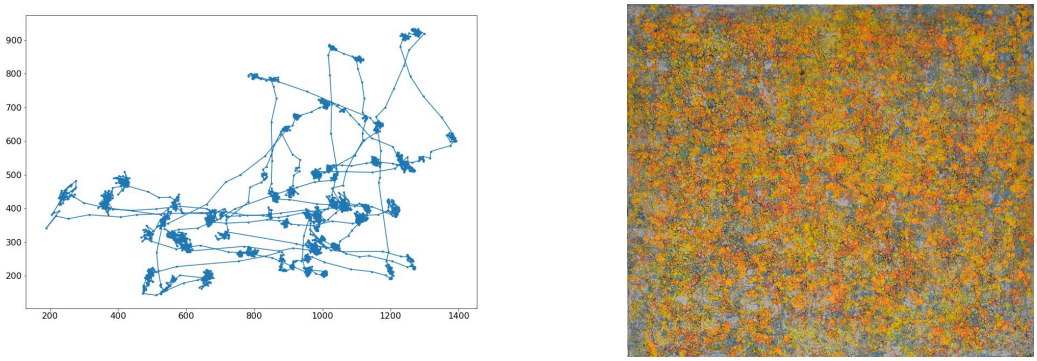
3.2 Eye-trajectory data

EyeT4Empathy database (Lencastre, 2021) was built by researchers at the AI lab at OsloMet. The dataset contains eye-movements from 60 participants, divided in two datasets based on the images and task they recorded. For this dataset they used paintings without specific motives. Participants were asked to describe what they saw in the image after looking at it for one minute. The aim was to assess how people forage for visual information in an unfamiliar scene. For collecting the data they used the eye-tracker Tobii Pro X3-120. The data has authorization by NSD, the Norwegian Centre for Research Data, allowing collection and publishing the data.

For this thesis we will only be using dataset II. Figure 3.2 shows two randomly chosen examples of trajectories from dataset II. The dataset contains a large range of features, but in this project we will only be concerned with the features 'Gaze point X' and 'Gaze point Y'. The data contains just below 15 percent nan values, which is important to consider before training the models as this can potentially negatively affect the training. In subsection 3.3.1 we describe how we fill in the missing values to reduce their potential negative impact.

3.3 Methodology

The aim of this project is to explore whether GANs are able to replicate eye-gaze trajectories. We will test a range of different time-series GANs, on four different sets of data, and evaluate how the models perform on the different data and where each of them falls short. In this section we first explain the experiments that are conducted for this project. Then we describe the models we chose to use in



(a) Trajectory from EyeT4Empathy.

(b) "Artwork 71" by the artist Rainer Gross.

Figure 3.2: EyeT4Empathy dataset II, example of trajectory. The trajectories in dataset II are the sampled trajectories from the first minute of participants looking at the seemingly random painting, after being asked to describe what they see in the painting. The aim was to capture how the participants use their eyes to forage for information in an unfamiliar scene. (a) Plot of a trajectory from EyeT4Empathy dataset II. This trajectory shows how the eyes seemingly explore the image in (b) in the same sense as the Intermittent Process, as shown in Figure 3.1c. (b) The image that the participants were asked to study. This image is the painting "Artwork 71" by the artist Rainer Gross. The image is borrowed from the EyeT4Empathy dataset, available online (Lencastre, 2021).

the experiments and their implementations. Finally, we present how we will be evaluating the different models' results for each experiment.

3.3.1 Experiments

This section describes the experiments conducted in this project. The goal is to explore a variety of existing GAN models' ability to replicate eye gaze trajectories. The experiments are conducted in four stages, with four different types of data of what we believe to be four increasingly difficult datasets to learn by a time-series GAN. The purpose is to be able to filter out the models that are not working for the simpler datasets, and to get an insight into the limitations of each model by observing where and on which data the different models start to struggle.

We will conduct four experiments with the different models implemented as described in subsection 3.3.2. The first experiment, which from now on will be

referred to as experiment A, uses one dimensional VAR data. Experiment B uses two dimensional VAR data. Experiment C uses generated intermittent processes. All data in experiments A, B and C are generated as described in section 3.1. Experiment D is the final experiment, using real eye-gaze trajectories from the EyeT4Empathy dataset (Lencastre, 2021).

For all experiments, the training of the models were done on the increments from one point in the time series to the next. For experiment A on one dimensional VAR data, which originally is a sequence of positions x, we define a new series of distances dx between the points at every time step t where $dx_t = x_{t+1} - x_t$. For experiment B, C, and D, which are all with two-dimensional data in which each point is defined by an x and a y value, we define a new series of distances dx and dy at every time step t where $(dx_t, dy_t) = (x_{t+1} - x_t, y_{t+1} - y_t)$.

For experiment D, to overcome the issues of missing values in the trajectories, we had two options: either removing the missing values, or calculate dummy values to replace them. Since the training is done on the increments between two points, removing the missing values can lead to some unnaturally high increments, which is a feature the GANs will be learning to replicate during training. To avoid this, we calculated the distances dx and dy from the previously known position (x_0, y_0) to the next known position (x_1, y_1) calculated by $(dx, dy) = (x_1 - x_0, y_1 - y_0)$, and then filled in the number n of consecutive missing values evenly in a straight line at each time step t, so that $(x_t, y_t) = (x_{t-1} + \frac{dx}{n}, y_{t-1} + \frac{dy}{n})$

During experiment A and B, the models' losses were calculated on generated sequences of 50 time steps, from given conditions of 50 time steps. For experiment C and D, the conditional GANs are fed a sequence of 50 and generate sequences of 100 time steps. Note that this is substantially more than what most of the models used in these experiments have been shown to generate in the literature. The SigCWGAN paper (Ni et al., 2020a) and TimeGAN paper (Yoon et al., 2019a) both used VAR data for comparing a range of different models, however in both these

papers the models were trained to generate only very short sequences at training time. Ni et al. generated sequences of three points from a condition of three points during training.

The reason we in this project train the models on longer sequences is because the saccadic periods and the fixation periods in the real eye-gaze trajectories lasts much longer than 3-6 time steps. For the models to learn to capture both the saccadic and fixational periods in the generated sequences, the sequences that are generated during training and fed to the discriminator should reflect this difference, at least to a certain degree. Generating longer sequences during training does though affect the time it takes to train the models, so this is a trade-off. Another trade-off for the time it takes to train is the total amount of training data. Using more data when training the models not only takes more time per iteration, but also requires more memory usage. In experiment D we used a random selection of the dataset (section 3.2) in order to make the training time and computational requirements manageable.

The sequences used in experiment A, B and C were of a total of 40000 time steps each. For experiment D we used a total of 20000 time steps of real eye-gaze trajectories. For all the experiments, the data was divided into 80% for training and 20% for evaluation. For the final evaluation in chapter 4, new data was generated, except in experiment D where we had set aside data before training to use in the final evaluation.

3.3.2 Models

All the models used in the experiments are models made for continuous sequential data. The reason behind this is that all the synthetic data and the eye trajectory data we will use in these experiments are originally represented as continuous sequences. In addition, as discussed in section 2.3, most of the state of the art models are made for sequential data, with the models made for NLP as an

exception. It is however, as we also presented in section 2.3, possible to convert various types of continuous data into a discrete representation and using them for training instead. By doing this, we could potentially use models made for discrete sequential data such as SeqGAN (L. Yu et al., 2017) and Transformer GANs (Muhamed et al., 2021a). This would require testing various ways of representing the data as discrete tokens. This possibility is further discussed in chapter 5.

The models we chose to use for the experiments are CWGAN (Ni et al., 2020b), RCGAN (Esteban et al., 2017a), RCWGAN (Ni et al., 2020b), SigCWGAN (Ni et al., 2020a), and TimeGAN (Yoon et al., 2019a). All these models are conditional GANs, whose aim is to be given a sequence which it tries to continue for a set amount of steps. The background on these models and on conditional GANs are given in subsection 2.3.1.

Our choice of using SigCWGAN was due to the AR-FNN, which was designed to work for ”tails of data”. This might make it ideal for eye-gaze trajectories. As shown in subsection 2.1.2, eye-gaze trajectories will consist of far more short increments than long ones. This is due to the small, frequent movements during the fixation periods, as opposed to the longer, faster increments in the saccadic periods. Thus, we can see the long increments to be what ... refer to as “tails of data”, which may be difficult for models to capture. In addition, Ni et al. aimed to ensure that SigCWGAN could capture temporal dependencies, which is also essential to successfully replicate eye-gaze trajectories. As described in subsection 2.1.2, in addition to the more short-termed dependency of switching between saccadic and fixational periods, the eyes have shown to not revisit recently viewed locations for a period of about 1.5 to 3 seconds (Le Meur & Liu, 2015).

We use the original implementation of SigCWGAN, by Ni et al. The code available on (Ni et al., 2020b). The implementation is done using PyTorch, in Python 3. We ran SigCWGAN 500 iterations for experiment A, 1000 iterations for experiment B, 2000 iterations for experiment C, and 4000 iterations for experiment

D. To make the comparison fair for the models, all the models were trained until they had stopped improving for several iterations, which differed a lot from model to model and is why the amount of training iterations varies a lot from model to model in our experiments.

To compare the impact of the C-Sig-W_1 metric in SigCWGAN, we also include Conditional Wasserstein GAN (CWGAN) and Recurrent Conditional Wasserstein GAN (RCWGAN) in the experiments. CWGAN is implemented similar to SigCWGAN, but does not use the C-Sig-W_1 metric. RCWGAN is the recurrent version of CWGAN, where both the generator and discriminator are RNNs instead. Both these models were implemented by Ni et al., and the implementations we used are available on (Ni et al., 2020b). CWGAN was only used for experiment A and B, where we trained it for 1000 iterations on both experiments. RCWGAN was trained for 5000 iteration for A and B, 10000 for C, and 20000 in experiment D.

TimeGAN is, as we saw in subsection 2.3.1, using a combination of GAN and autoencoder architecture. This model has shown useful in a range of tasks, and is recently used a lot as a benchmark (Ni et al., 2020a). We use the TimeGAN implementation by Ni et al. (2020a), which according to (Ni et al., 2020a) is based off the original TimeGAN code by Yoon et al. (2019a), whose original implementation is available on GitHub (Yoon et al., 2019b) in Python 3 using Tensorflow 3.

The implementation of TimeGAN used in this project is available on (Ni et al., 2020b). This implementation is done using PyTorch, in Python 3. We ran the training 10000 iterations for experiments A and B, and 20000 for C and D.

RCGAN was used as a comparison in both TimeGAN and SigCWGAN. The results by Yoon et al. showed that TimeGAN performed better than RCGAN, however the comparison by Ni et al. showed very similar results between TimeGAN and RCGAN. We therefore wanted to include RCGAN in our experiments. For RCGAN we use the implementation by Ni et al. (Ni et al., 2020b). It is based off the original RCGAN implementation by Esteban et al., whose original code is available

(Esteban et al., 2017b). We trained RCGAN 5000 iterations for experiments A and B, and 10000 iterations for experiments C and D.

3.3.3 Evaluation metrics

We defined a set of statistical measures to evaluate the results of the experiments. For each of the models in each experiment L_2 distance, Kolmogorov Smirnoff (KS) test, distance intensity of the auto-correlation profiles and the similarity profile of the auto-correlations were calculated. The first two measures, the L_2 distance and the KS test, only concerns the distribution of the data without taking into consideration the time aspect of the generated data. The auto-correlation of the increments is the simplest method to compare two samples of data which considers the time-aspect.

The L_2 distance measures the squared difference between two distributions. Let $p(x)$ be the original probability distribution function of the synthetic data and $q(x)$ the probability distribution function for the GAN generated distribution. Then, the L_2 distance d_{L_2} is given by Equation 3.3.

$$\sqrt{d_{L_2} = \sum_x (p(x) - q(x))^2} \tag{3.3}$$

The goal of the GAN generated data in these experiments is that it should have the same distribution as the original data for its respective experiment. In order to actually know whether the distribution of the real data and the GAN generated data come from the same distribution, we use a statistical test called the Kolmogorov Smirnoff (KS) test. The KS test gives us a probability value p of whether two samples of data comes from the same distribution. We then take the distance of the probability value p. The p distance is calculated by $log(1/p)$. The value can be from 0 and up, and will be larger for a smaller p value.

We set the threshold for an acceptable p distance at 95 percent similarity. This is calculated by $log(1/0.05)$, which is ≈ 1.3. A p distance value lower than this means that there is an over 95 percent chance that the samples come from the same distribution.

The two final measurements, which both regards the auto-correlations of the samples, are the difference in the profiles and the differences in absolute values. We will call the first one the "distance intensity" and the second the "similarity profile". To compute these quantities we will compare the auto-correlation functions acf of the original synthetic data $acf_o(i)$ and of the GAN generated data $acf_{GAN}(i)$. Each element represents the correlation of the increment of the process with the i-th previous increment in the time-series.

$$d_{int} = \sqrt{\sum_{i=0}^{i=N} (acf_o(i) - acf_{GAN}(i))^2} \tag{3.4}$$

The distance intensity d_{int} of the auto-correlations are calculated by Equation 3.4. The distance varies between 0 and 1, where 0 means a very little distance, and 1 means a high distance. If the result is 0, this means that the absolute value of the correlations is similar.

$$s_{prof} = \frac{\sum_{i=0}^{i=N} (acf_o(i) \times acf_{GAN}(i))^2}{\sqrt{\sum_{i=0}^{i=N} (acf_o(i) \times acf_o(i))^2} \times \sqrt{\sum_{i=0}^{i=N} (acf_{GAN}(i) \times acf_{GAN}(i))^2}} \tag{3.5}$$

Similarity profile s_{prof} of the auto-correlations is measured by Equation 3.5, where acr is the auto-correlation function, o is the original data, and GAN is the generated data. If the similarity profile is 1 it means that the auto-correlations have a similar profile, and if it is 0 they have a very different profile. Negative values are also possible, but rare. To check if the similarity is statistically significant, we define a similarity threshold at 99 percent, which is calculated for each sample. If the similarity profile is higher than 99 it is very high, and we can say confidently that

the profiles match better than random auto-correlation functions. Note though that this does not necessarily mean that the generated data is good or realistic, only that it is certainly better than random data.

Chapter 4

Results

This section will present the results of each of the four experiments conducted for this thesis. Table 4.1 includes all the results from all the models that were trained, in each of the four experiments. For each model, ten samples of 2000 time steps were generated upon finishing training, to use in the evaluation. Each cell in Table 4.1 gives the average and the standard deviation of the ten generated samples for the respective metric, for the respective model, as well as the one smallest result and the one highest result among the ten samples. The results marked as bold are the best results for that metric in the respective experiment. For the similarity distance of the auto-correlation functions all values where the average had an accuracy of over 99 percent are marked as bold.

For the evaluation of experiment A, we used the series' of generated increments and compared them to an original series of VAR 1D, represented as increments, which is the same representation dx that the models were trained on as described in subsection 3.3.1 . For the rest of the experiments, for which the data was two dimensional, the evaluation was done using the combined increments of dx and dy at every time step. We call this combined increment ds, so the two-dimensional sequences, both original and generated, were thus represented as $ds_t = \sqrt{dx_t^2 + dy_t^2}$ for the evaluation of experiments B, C, and D.

For each of the experiments, a table of figures showing the results of one randomly chosen sample per model is provided, like Table 4.2 which is the result table for experiment A. These tables are all at the end of this chapter. For each of the tables, the first row is a plot of a generated time series per model. The second row is the distribution of increments, dx in experiment A and ds in the other experiments. The third row is the auto-correlation profiles of each sequence. To compare the generated samples to how they ideally should be, the plots in all three rows also shows the results for one sample of the type of data used for training in that respective experiment.

4.1 Experiment A - VAR one dimension

In this experiment, five GAN models were trained to replicate VAR 1D data. The training data was generated as described in section 3.1, and Figure 3.1a is a plot of a sequence of 2000 point of VAR 1D data.

From Table 4.2 it becomes apparent that, at least for this one sample, CWGAN fails to generate a realistic sequence. The sequence itself, shown in the first row, does not look as expected, and the histogram in the second row shows that the distribution of increments for this generated sequence is skewed towards negative increments. The auto-correlation profile also appears to be further from the auto-correlation profile of the original data compared to the rest of the models. From Table 4.1 we can confirm that the auto-correlation similarity indeed is smaller than for the rest of the models. SigCWGAN does however have a higher distance intensity than CWGAN. Common for all the models is that they seem to have a tendency of favouring smaller increments than the original data.

		SigCWGAN	TimeGAN	RCGAN	RCWGAN	CWGAN
Experiment A VAR 1D	L_2	**0.163 ± .009**	0.199 ± .001	0.183 ± .008	0.17 ± .01	0.39 ± .04
		0.15 0.177	0.182 0.216	0.169 0.194	0.153 0.191	0.314 0.437
	p	**25 ± 2**	37 ± 4	35 ± 5	31 ± 6	163 ± 18
		20.562 28.01	32.263 44.942	27.522 42.505	23.906 45.039	127.611 191.428
	distance	0.869 ± .008	0.34 ± .07	0.31 ± .08	**0.26 ± .09**	0.5 ± .2
	intensity	0.851 0.881	0.254 0.456	0.149 0.434	0.056 0.388	0.223 0.889
	similarity	0.58 ± .05	**0.989 ± .007**	**0.994 ± .004**	**0.982 ± .009**	0.4 ± .2
	profile	0.476 0.652	0.971 0.996	0.985 0.998	0.968 0.99	0.182 0.74
Experiment B VAR 2D	L_2	0.733 ± .004	0.241 ± .009	**0.09 ± .02**	0.21 ± .02	0.206 ± .006
		0.724 0.739	0.222 0.252	0.05 0.133	0.178 0.231	0.194 0.216
	p	inf	167 ± 14	**24 ± 12**	130 ± 21	50 ± 10
		inf inf	143.42 187.843	4.541 51.285	92.707 160.669	30.965 64.176
	distance	0.446 ± .08	0.283 ± .07	0.7 ± .1	**0.18 ± .07**	0.991 ± .0
	intensity	0.255 0.56	0.2 0.429	0.468 0.93	0.068 0.318	0.991 0.991
	similarity	0.87 ± .03	**0.994 ± .005**	**0.8 ± .1**	**0.992 ± .005**	**0.143 ± .002**
	profile	0.83 0.927	0.984 0.999	0.465 0.951	0.984 0.998	0.14 0.146

Continued on next page

Continued		SigCWGAN	TimeGAN	RCGAN	RCWGAN	CWGAN
Experiment C Intermittent Process	L_2	0.69 ± .01 0.677 0.707	0.57 ± .01 0.547 0.582	0.60 ± .01 0.584 0.623	**0.55 ± .03** 0.492 0.616	-
	p	inf inf inf	inf inf inf	inf inf inf	inf inf inf	-
	distance intensity	0.6 ± .2 0.201 0.816	0.4 ± .1 0.177 0.527	0.6 ± .2 0.154 0.789	**0.4 ± .2** 0.025 0.722	-
	similarity profile	**0.89 ± .07** 0.79 0.999	**0.992 ± .005** 0.983 0.997	**0.991 ± .004** 0.982 0.999	**0.96 ± .03** 0.899 0.989	-
Experiment D EyeT4Empathy data	L_2	0.32 ± .0 0.32 0.32	0.32 ± .0 0.32 0.32	0.32 ± .0 0.32 0.32	0.32 ± .0 0.32 0.32	-
	p	inf inf inf	inf inf inf	inf inf inf	inf inf inf	-
	distance intensity	0.984 ± .004 0.973 0.987	0.5 ± .1 0.413 0.709	0.95 ± .02 0.888 0.973	**0.5 ± .2** 0.14 0.725	-
	similarity profile	0.1 ± .1 -0.071 0.348	**0.92 ± .04** 0.847 0.97	**0.6 ± .1** 0.423 0.723	**0.92 ± .06** 0.823 0.978	-

Table 4.1: Results from all four experiments. Ten generated samples of 2000 time steps for each model were evaluated. Each cell in this table has four values. On the top it shows the average and the standard deviation of the generated samples for the respective metric, and on the bottom of each well we have the minimum and maximum value among the ten samples. Here, "L_2" is the L_2 distance, "p" is the p distance of the KS test, "distance intensity" is the distance intensity of the auto-correlation function, and "similarity profile" is the similarity profile of the auto-correlation function.

By just looking at the single samples in Table 4.2, TimeGAN and RCGAN appear to be the ones performing the best. In Table 4.1 though, when evaluating ten generated samples, SigCWGAN has the strongest performance on the L_2 distance with 0.163 ± .009 and on the KS test with a p distance of 25 ± 2. For the distance intensity RCWGAN performed the best with a score of 0.18 ± .07. On the L_2 distance all models except CWGAN performed similarly. On the p distance, all models failed to perform below the 1.3 acceptance limit.

4.2 Experiment B - VAR two dimensions

Experiment B included five models trained on VAR 2D data. For this experiment, RCGAN performed well regarding the L_2 distance with a value of 0.093 ± .004. RCGAN also performed the best among the models for the p distance, but the score of 25 ± 12 is still much higher than the acceptance limit. Even the lowest score among RCGANs samples of 4.541 is too high. For the auto-correlation distance, which also concerns the time aspect of the data, RCGAN performed worse than tree of the other models.

As expected from seeing the results of experiment A, CWGAN had a very poor performance in this experiment also. It performed a little bit better than on experiment A, according to the evaluation metrics in Table 4.1, but from the example shown in Table 4.3 it is very apparent that this GAN did not work well in this experiment.

SigCWGAN was the only one of the models in this experiment to not perform over the acceptance limit of 99 percent for the similarity distance. None of the models were anywhere near the acceptance limit for the KS test.

4.3 Experiment C - Intermittent process

RCWGAN outperformed the other three models in this experiment, with an L_2 of 0.55 ± .03, a distance intensity of 0.4 ± .2, and a similarity profile of 0.96 ± .03. The similarity profiles for all models were over our set threshold for 99 percent, meaning that all the models were able to generate data that had a closer auto-correlation profile to the intermittent processes than random data would have had. From the second row in Table 4.4 it becomes even more apparent that the models favour smaller increments than the original data. The original data has a distribution of mostly smaller increments, but then also has a peak of longer increments which are the saccades in the intermittent process. From these histograms of the samples from the models it seems that none of the models were able to capture this distribution.

The leftmost plot in the first row of Table 4.4 shows an example of a 2000 steps long sequence of original intermittent processes like the ones the models were trained to generate, and the rest of the first row shows one example of a sequence generated by each different model after completing the training. SigCWGAN is the only one of the models that seems to have to some degree captured the saccadic jumps' constant direction.

4.4 Experiment D - Eye-gaze trajectories

In the final experiment we trained four of the models on eye-gaze trajectories from the EyeT4Empathy dataset II.

RCWGAN was the model to perform the best in this experiment for the auto correlation measured with a distance intensity of 0.5 ± .2 and similarity profile of 0.91 ± .06 TimeGAN almost had a distance intensity as low as RCWGAN, but SigCWGAN and RCGAN had much higher distance intensity with 0.984 ± .004 and 0.95 ± .02 respectively.

All the models had an infinite high p distance, which means that there is an almost non-existing probability that the generated data came from the same distribution as the real eye-gaze trajectory data.

The reason the L_2 distance for this experiment are the same for all in Table 4.1 is that the bins that were calculated puts all the data within the same bins, since all the models' generated data has much smaller increments than the real data. This gives the impression that the models might have only produced constant increments, which is not the case. The second row of Table 4.5 shows how the increments are being placed into the same bin, and hence appears to be the same for all the models.

Table 4.6 demonstrates the differences in the increments for one sample for each of the models. It is apparent that both the dx, dy, and ds increments of the samples from each of the models are in fact not the same for all models.

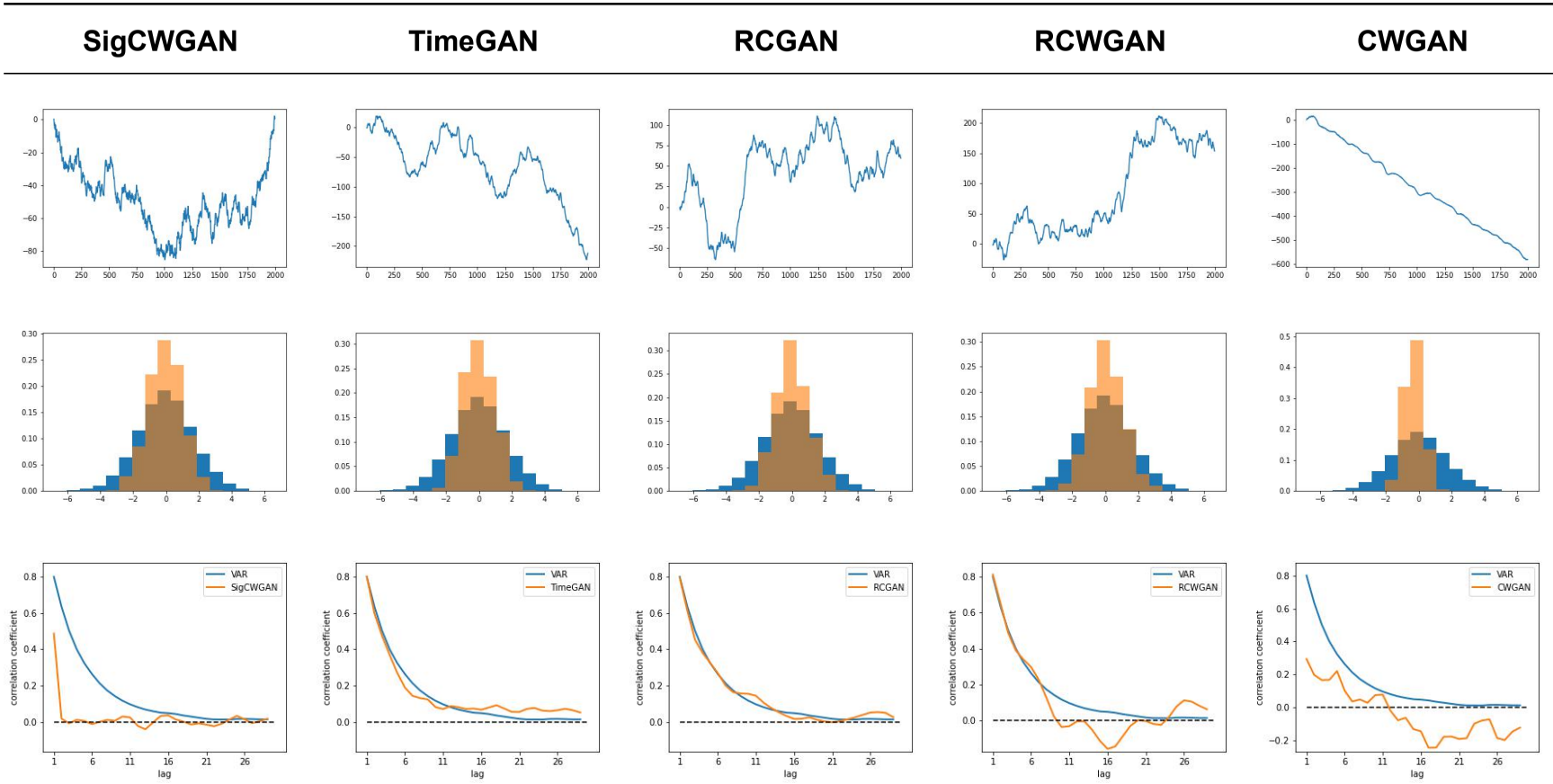


Table 4.2: Examples of generated trajectories from experiment A, along with the distributions of the increments, and the auto-correlation profile. The first row shows one example of a generated trajectory of 2000 time steps per model, after being trained on one dimensional VAR data. The second row shows the trajectories' distribution of increments compared to an example of real VAR 1D data. The bottom row shows the trajectories' auto-correlation profile of the increments, compared to a sample of real data.

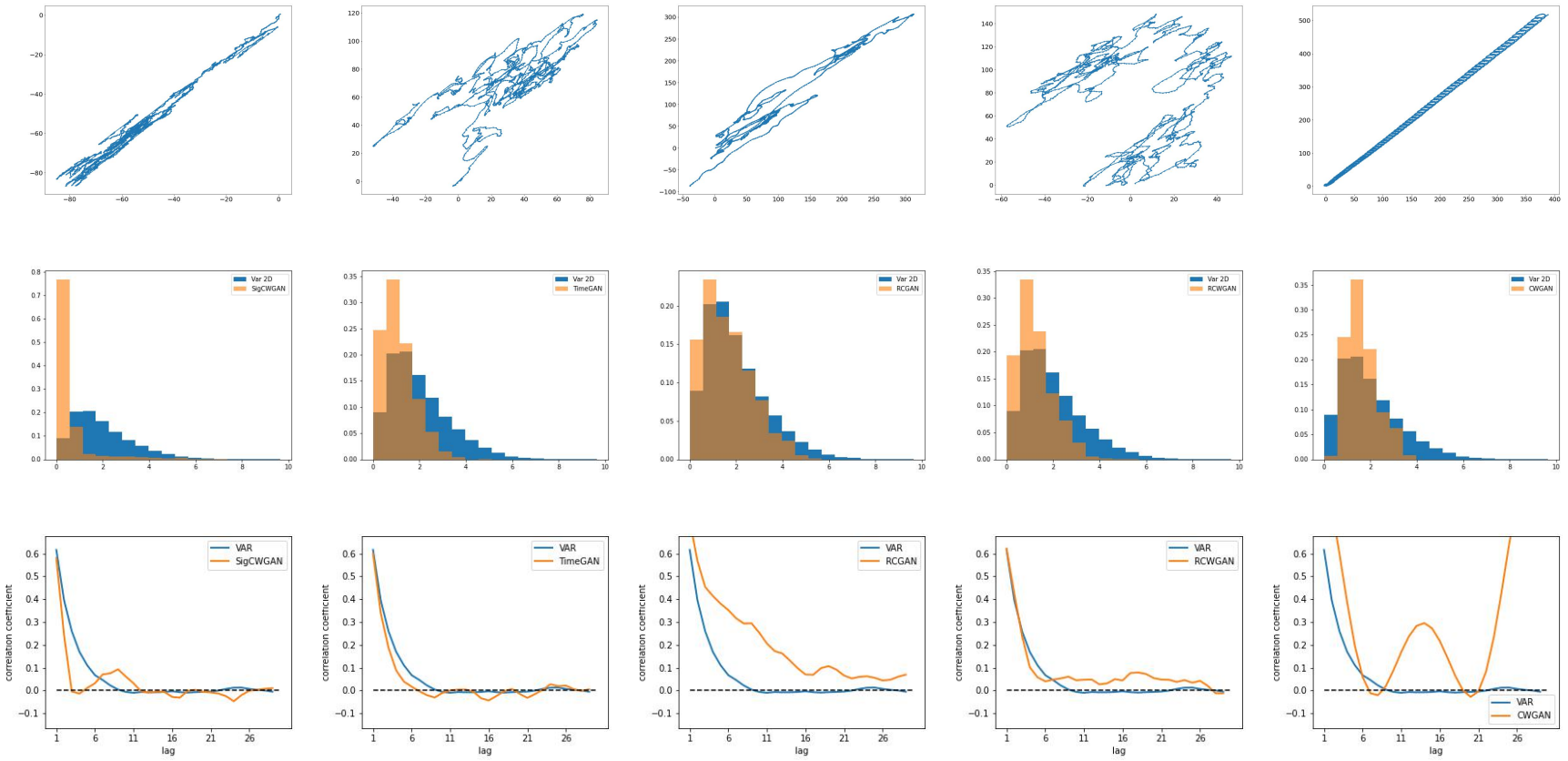


Table 4.3: Examples of generated trajectories from experiment B. The first row is a plot of one example of a generated trajectory of 2000 time steps per model, after being trained on two dimensional VAR data. The second row shows the trajectories' distribution of increments compared to an example of real VAR 2D data. The bottom row shows the trajectories' auto-correlation profile of the increments, compared to a sample of real data.

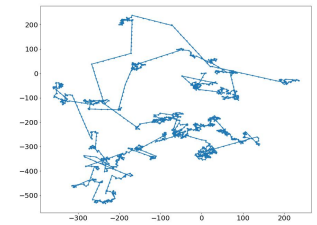
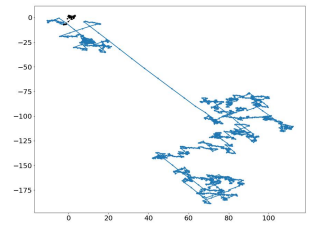
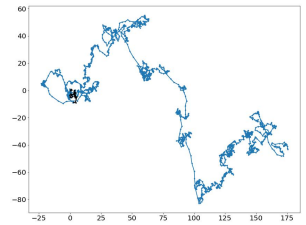
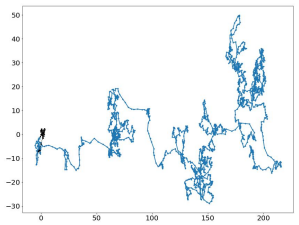
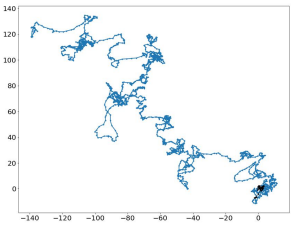
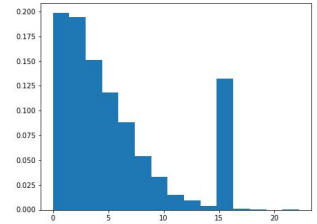
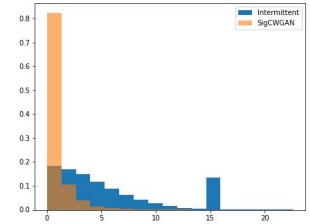
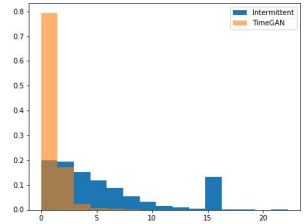
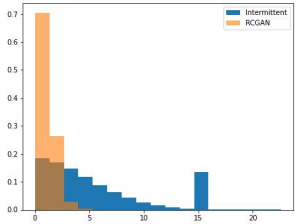
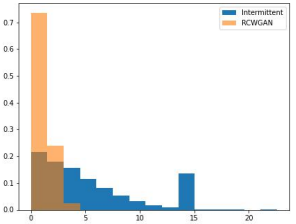
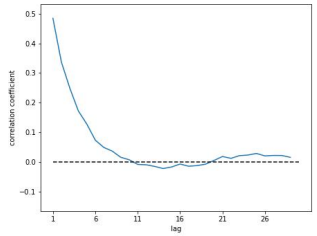
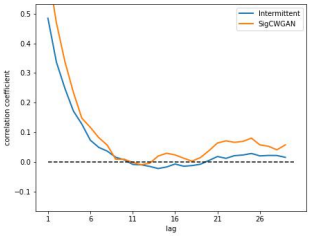
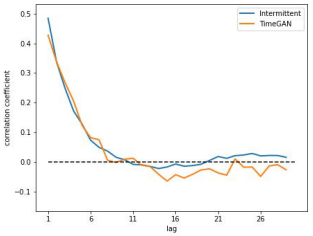
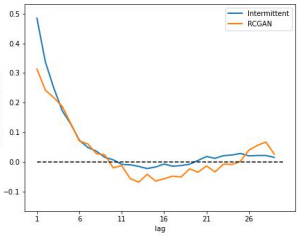
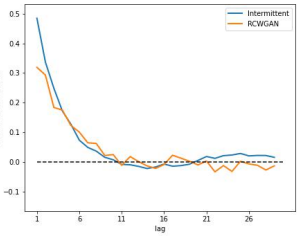
Intermittent	SigCWGAN	TimeGAN	RCGAN	RCWGAN
				
				
				

Table 4.4: Examples of generated trajectories from experiment C. The first column shows an example of 2000 time steps from the original intermittent process data. The models were given sequences of 50 points of intermittent process data at generation, which are the dotted black section of the plots. The first row is a plot of one example of a generated trajectory of 2000 time steps per model. The second row shows the trajectories' distribution of increments compared to the real sample. The bottom row shows the trajectories' auto-correlation profile of the increments, compared to the real sample.

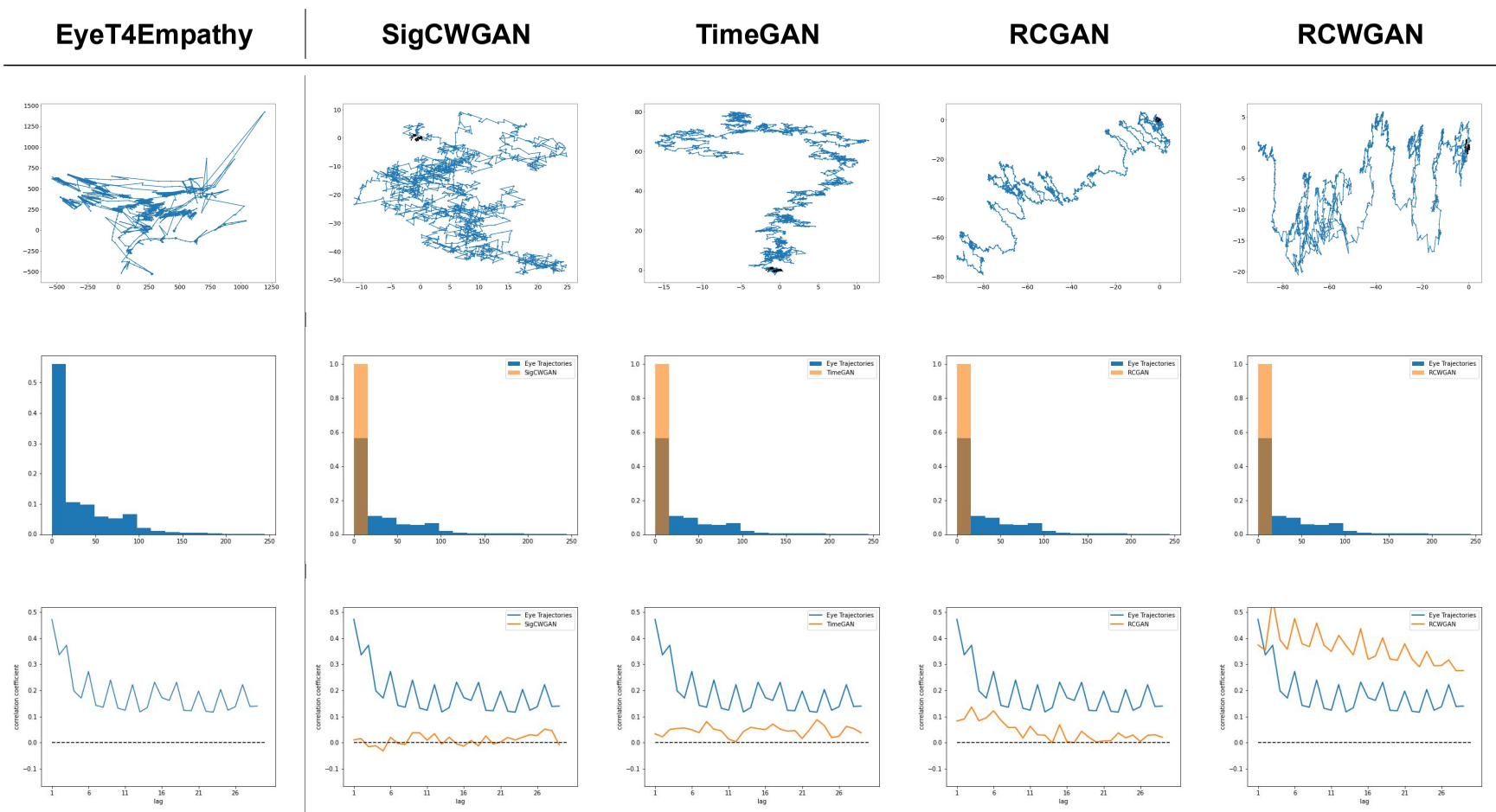


Table 4.5: Examples of generated trajectories from experiment D. The first column shows an example of 2000 time steps from the original eye-gaze trajectory data from the EyeT4Empathy dataset II (Lencastre, 2021). The models were given sequences of 50 points of eye-gaze trajectory data at generation, which are the dotted black section of the plots. The first row is a plot of one example of a generated trajectory of 2000 time steps per model. The second row shows the trajectories' distribution of increments compared to the real sample. The bottom row shows the trajectories' auto-correlation profile of the increments, compared to the real sample.

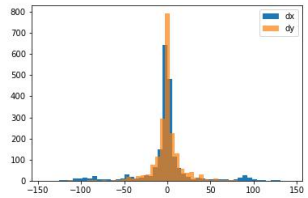
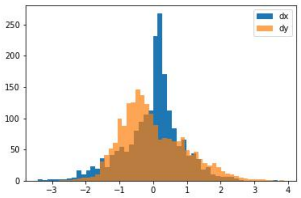
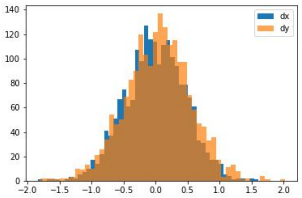
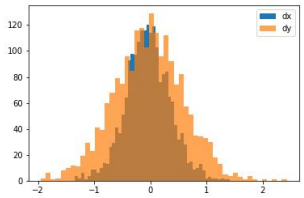
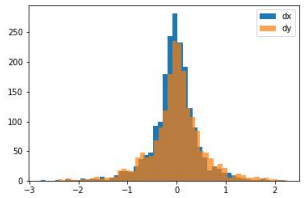
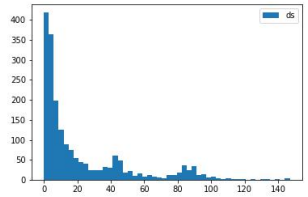
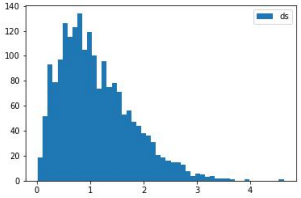
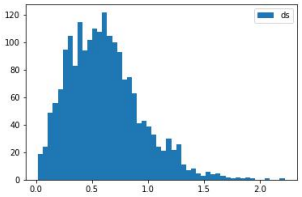
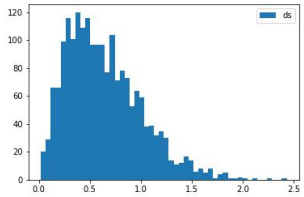
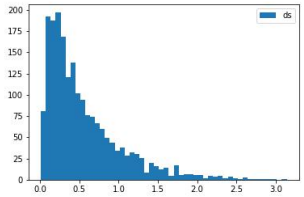
EyeT4Empathy	SigCWGAN	TimeGAN	RCGAN	RCWGAN
				
				

Table 4.6: Experiment D: frequency distribution of the increments, from the same generated samples per model as used in Table 4.5. The first column shows an example of 2000 time steps from original eye-gaze trajectory data. The first row is cumulative histograms of dx and dy for the samples, as defined in subsection 3.3.1. The second row is a cumulative histogram of the ds for the samples, as described in the beginning of chapter 4

Chapter 5

Discussion and conclusions

In this thesis we have explored the possibility to generate new, unique eye-gaze trajectories of the eyes' foraging process using time-series GAN models. Being able to generate datasets of realistic synthetic eye-gaze trajectories could be used in research within statistics and artificial intelligence for applications in for example psychology and medicine. We hypothesized that it would be possible to train existing state-of-the-art GANs for time-series to replicate eye-gaze trajectories, capturing the advanced properties of the human gaze when searching for information in an unfamiliar scene or image.

The experiments were divided into four stages, which we referred to as experiments A, B, C, and D. For the first three experiments we generated three different datasets of processes of increasing complexity. We wanted to get an understanding of the limitations of each of the models. To achieve this we experimented with data of increasingly complex processes that we assumed would be increasingly challenging for the models to replicate, before doing a final experiment using real eye-gaze trajectories of the eyes' foraging process. We assumed that the models would be able to perform similarly well for the real eye-gaze trajectories in experiment D as they did for the intermittent processes in experiment C, due to the hypothesis in the literature that the eyes' foraging process

follows an intermittent process (Land, 2019).

For the experiments we tested five different time-series GAN architectures: Recurrent Conditional GAN (RCGAN); Conditional Wasserstein GAN (CWGAN); Recurrent Conditional Wasserstein GAN (RCWGAN); Conditional Sig-Wasserstein GAN (SigCWGAN); and Time-series GAN (TimeGAN) (subsection 3.3.2). The methods we used in these experiments are conditional GANs which work by giving the generator a short sequence of a real eye-gaze trajectory, and let the generator continue this sequence for a given length. We trained the models to generate a continuance of an eye-gaze trajectory given a short sequence of 50 time-steps of a real eye gaze trajectory from the EyeT4Empathy (Lencastre, 2021) dataset II. To evaluate the models' performances we used two evaluation metrics to evaluate the data distribution, and two metrics to evaluate the temporal correlation.

We expected SigCWGAN to perform better than RCGAN and TimeGAN, based on the comparisons on VAR data by Ni et al. (2020a). We also expected that TimeGAN might perform slightly better than RCGAN, based on the comparisons by Yoon et al. (2019a). There were however some differences in both of their comparisons compared to ours, the main being that in their evaluations the models had only been trained on generating short sequences. In our experiments however, we used longer conditions and generated longer sequences during training.

5.1 Discussion

5.1.1 Findings

For experiment A, using one dimensional VAR data, we found that SigCWGAN performed best on the distribution metrics, L_2 distance and p distance of the KS test. However, SigCWGAN had the highest distance intensity of the auto-correlation function. RCWGAN performed the best on the temporal dependence metrics, distance intensity and similarity profile of the auto-correlation, and was a close

second on the distribution of increments.

These findings are not in alignment with the findings of Ni et al. (2020a). When they used 1D VAR data generated with the same value for the temporal correlation ϕ as we used in our experiments, they found that SigCWGAN outperformed RCGAN and TimeGAN on the distance of the auto-correlation function. Other than this, our findings in experiments A align with the findings of Ni et al., such as our finding that RCGAN performed slightly better than TimeGAN on our metrics.

When using two dimensional VAR data in experiment B, we had expected similar results as for the first experiment. We instead found RCGAN to be performing the best on our metrics on the distribution of increments, L_2 distance and p distance. On the distance intensity of the auto-correlation function however, RCGAN only performed better than CWGAN, while RCWGAN had the smallest distance.

For two dimensional VAR data where $\sigma = 0.8$ and $\phi = 0.8$, same as in our experiment B, (Ni et al., 2020a) found that SigCWGAN performed better than TimeGAN and RCGAN on all their five evaluation metrics. Yoon et al. (2019a) used a predictive score and a discriminative score to compare TimeGAN to among others RCGAN. They found that TimeGAN performed the best in replicating VAR data with the same values and σ and ϕ as ours, and RCGAN came out as second. However, we found that RCGAN has a smaller L_2 distance and p distance than TimeGAN, but TimeGAN had a smaller distance intensity of the auto-correlation function than RCGAN.

For both experiments A and B CWGAN fell short, and was not able to perform as well as any of the other four models. This was as expected as CWGAN is non-recurrent. Based on the poor results in these two experiments, we chose not to continue using CWGAN in experiment C and D.

For experiment C we had expected the models to perform poorer than they did in experiment B. We anticipated intermittent processes to be harder for the GANs to learn to generate than two-dimensional VAR data, due to the switching between the

saccadic and fixational processes. Surprisingly, the models did not perform much poorer, but in fact performed better for a few of the evaluation metrics.

Interestingly, it seems that all the four models in experiment C were able to, at least to a certain degree, capture the alternation between fixations and saccades, which we could see in the first row of Table 4.4 in the chapter 4. This alternation is, as we discussed in subsection 2.1.1, the most central property of the intermittent processes, so it would be a great achievement for the models to properly capture it. All four models produced a distribution of increments which is preferring much shorter increments than in the intermittent process data. This is consistent with previous findings sawing that machine-learning models have a tendency of underestimating large increments (Lind et al., 2017).

Our assumption that the models which performed well for intermittent processes in experiment C would be the same models to work well for the real eye-gaze trajectories in experiment D. We found this to partly be the case. None of the models were able to reach the aim of replicating eye-gaze trajectories to a degree where they are indistinguishable from real ones. RCGAN and TimeGAN were the two models with the smallest distance intensities of the auto-correlation function, and this was also the case for experiment D. SigCWGAN did not have a satisfactory similarity profile of the auto-correlation function, something the three other models were able to achieve.

Even though our results show the generated data to not be very similar to the real data for all models in experiments D, we did have some interesting observations. In the plotted example of auto-correlation of RCWGAN, it is apparent that RCWGAN was able to achieve a similar shape as the real data even though the distance of the auto-correlation was high. This may be an indication of RCWGANs potential in replicating eye-gaze trajectories. The plotted trajectories for all four models used in both experiments C and D also show that the generated samples indeed show a tendency of switching between shorter and longer increments, which

may be a sing that they may indeed have a potential of capturing the switching between saccades and fixation of the real eye-gaze trajectories.

5.1.2 Limitations

We have identified these as the primary limitations who can have impacted the results of this project.

In order to capture both saccadic and fictional processes of the eye trajectories, we chose to train the models in experiments C and D by giving them sequences of 50 points and generating sequences of 100 time steps, which increased the training time. In addition, due to limited time and computational resources, we chose to only use a subset of the EyeT4Empathy dataset 2, which made the training computationally feasible. We expect the models to have had better results, if we were able to increase the generated time steps for each sequence during training, and the total amount of training data.

Data can be represented in several different ways, as discussed in subsection 2.3.1. In each of these experiments only one representation was used. To limit the scope of the experiments of this project, we chose to only use models created for continuous data. The reason being that both the synthetic and real data used in these experiments are all originally represented continuously. Nevertheless, as discussed in section 2.3 the same data can be represented both in a continuous and a discrete manner. For the synthetic data and the eye-gaze trajectories this would involve having to transform the data into discrete tokens, which must also be possible to revert to the continuous representation for later usage. In NLP and music generation several methods for changing the original data into discrete representation. In NLP it is common to use multi-dimensional vectors, and in music generation we saw examples of tokenizing based on tone duration, pitch, etc. For the type of trajectories there is no standard way of converting, hence we would need to try different representations. This was not tested in this project due to the very

limited time aspect of this master thesis.

5.1.3 Strengths

Based on literature review, in subsection 2.3.3, we found research aiming to predict where in an image humans will look. Assens et al. (2018) used conditional GANs, where the image is the condition. They were not concerned of the foraging process of the eyes, as our project is. GANs have also been used to generate foraging trajectories of animals foraging for food (Roy et al., 2021), which can be seen as a more similar research question to ours, but still from quite a different domain. To our knowledge there is no research on using time-series GANs to replicate eye-gaze trajectories of the foraging process of the human gaze.

There is no consensus in the literature on which metrics to use when comparing time-series GANs (Brophy et al., 2021). We chose four evaluation metrics which we believe can fit all time-series with temporal dependencies, in order to compare the underlying statistical processes behind the sequences of the original data and the generated data. Two of the metrics concerns only the distribution of the data, while the other two measures the auto-correlation and thus captures the temporal dependencies of the data.

5.2 Conclusion and future work

The time-series GAN models tested in our experiments all fell short of being able to generate realistic eye-gaze trajectories. Some of the models showed promising results when generating VAR and Intermittent processes, which does give us an indication that with some further adjustments, those models might be able to generate realistic eye-gaze trajectories.

In this project we tested time-series GANs with a continuous representation of the data. Even though the results of the models in our experiments fell short of

generating realistic eye-gaze trajectories, we believe that time-series GANs have a potential of generating eye-gaze trajectories. We propose to study the possibility of changing the representation of the data into a discrete manner. If so, other state-of-the-art GAN models using RNNs such as SeqGAN (L. Yu et al., 2017), or QuantGAN (Wiese et al., 2020) could be tested, or Transformer or BERT based GAN models. Specifically, we have faith in the architecture of Symbolic Music Generation with Transformer-GANs by Muhamed et al. (2021a), whose implementation is publicly available as a GitHub project (Muhamed et al., 2021b).

Bibliography

Antoniades, C. A., Xu, Z., Mason, S. L., Carpenter, R. H. S. & Barker, R. A. (2010). Huntington's disease: changes in saccades and hand-tapping over 3 years. *Journal of neurology*, *257*(11), 1890–1898.

Ariel, G., Rabani, A., Benisty, S., Partridge, J., Harshey, R. & Be'Er, A. (2015). Swarming bacteria migrate by Lévy Walk. *Nature communications*, *6*(1), 1–6.

Assens, M., Giro-i-Nieto, X., McGuinness, K. & O'Connor, N. E. (2018). PathGAN: Visual scanpath prediction with generative adversarial networks. *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 0.

Bargary, G., Bosten, J. M., Goodbourn, P. T., Lawrance-Owen, A. J., Hogg, R. E. & Mollon, J. D. (2017). Individual differences in human eye movements: An oculomotor signature? *Vision research*, *141*, 157–169.

Bénichou, O., Loverdo, C., Moreau, M. & Voituriez, R. (2011). Intermittent search strategies. *Reviews of Modern Physics*, *83*(1), 81.

Berkovsky, S., Taib, R., Koprinska, I., Wang, E., Zeng, Y., Li, J. & Kleitman, S. (2019). Detecting personality traits using eye-tracking data. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–12.

Brockmann, D. & Geisel, T. (1999). Are human scanpaths Levy flights?

Brockmann, D. & Geisel, T. (2000). The ecology of gaze shifts. *Neurocomputing*, *32*, 643–650.

Brophy, E., Wang, Z., She, Q. & Ward, T. (2021). Generative adversarial networks in time series: A survey and taxonomy. *arXiv preprint arXiv:2107.11098*.

Burrell, J. R., Hornberger, M., Carpenter, R. H. S., Kiernan, M. C. & Hodges, J. R. (2012). Saccadic abnormalities in frontotemporal dementia. *Neurology*, *78*(23), 1816–1823.

Cartuyvels, R., Spinks, G. & Moens, M.-F. (2021). Discrete and continuous representations and processing in deep learning: looking forward. *AI Open*, *2*, 143–159.

Caswell, I. & Liang, B. (2020). Recent Advances in Google Translate. Retrieved May 9, 2022, from https://ai.googleblog.com/2020/06/recent-advances-in-google-translate.html

Chauhan, H., Prasad, A. & Shukla, J. (2020). Engagement Analysis of ADHD Students using Visual Cues from Eye Tracker. *Companion Publication of the 2020 International Conference on Multimodal Interaction*, 27–31.

Chernyavskiy, A., Ilvovsky, D. & Nakov, P. (2021). Transformers:" The End of History" for NLP? *arXiv preprint arXiv:2105.00813*.

Chung, J., Gulcehre, C., Cho, K. & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Credidio, H. F., Teixeira, E. N., Reis, S. D. S., Moreira, A. A. & Andrade Jr, J. S. (2012). Statistical patterns of visual search for hidden objects. *Scientific reports*, *2*(1), 1–6.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C. & Yang, Y.-H. (2018). Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Esteban, C., Hyland, S. L. & Rätsch, G. (2017a). Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*.

Esteban, C., Hyland, S. L. & Rätsch, G. (2017b). RGAN. Retrieved May 7, 2022, from https://github.com/ratschlab/RGAN

Gensler, A., Henze, J., Sick, B. & Raabe, N. (2016). Deep Learning for solar power forecasting—An approach using AutoEncoder and LSTM Neural Networks. *2016 IEEE international conference on systems, man, and cybernetics (SMC)*, 2858–2865.

Giuliari, F., Hasan, I., Cristani, M. & Galasso, F. (2021). Transformer networks for trajectory forecasting. *2020 25th International Conference on Pattern Recognition (ICPR)*, 10335–10342.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Grace, P. M., Stanford, T., Gentgall, M. & Rolan, P. E. (2010). Utility of saccadic eye movement analysis as an objective biomarker to detect the sedative interaction between opioids and sleep deprivation in opioid-naive and opioid-tolerant populations. *Journal of Psychopharmacology*, *24*(11), 1631–1640.

Greff, K., Srivastava, R. K., Koutń, J., Steunebrink, B. R. & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, *28*(10), 2222–2232.

Hazra, D. & Byun, Y.-C. (2020). SynSigGAN: Generative adversarial networks for synthetic biomedical signal generation. *Biology*, *9*(12), 441.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hou, X., Shen, L., Sun, K. & Qiu, G. (2017). Deep feature consistent variational autoencoder. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1133–1141.

Jiang, Y., Chang, S. & Wang, Z. (2021). Transgan: Two pure transformers can make one strong gan, and that can scale up. *Advances in Neural Information Processing Systems*, *34*.

Jordan, M. I. & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Kalchbrenner, N., Danihelka, I. & Graves, A. (2015). Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Kaplanyan, A. S., Sochenov, A., Leimkühler, T., Okunev, M., Goodall, T. & Rufo, G. (2019). DeepFovea: Neural reconstruction for foveated rendering and video compression using learned statistics of natural videos. *ACM Transactions on Graphics (TOG)*, *38*(6), 1–13.

Kasneci, E., Kasneci, G., Appel, T., Haug, J., Wortha, F., Tibus, M., Trautwein, U. & Gerjets, P. (2021). TüEyeQ, a rich IQ test performance data set with eye movement, educational and socio-demographic information. *Scientific Data*, *8*(1), 1–14.

Kingma, D. P. & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kröger, J. L., Lutz, O. H.-M. & Müller, F. (2019). What does your gaze reveal about you? On the privacy implications of eye tracking. *IFIP International Summer School on Privacy and Identity Management*, 226–241.

Lambiotte, R., Salnikov, V. & Rosvall, M. (2015). Effect of memory on the dynamics of random walks on networks. *Journal of Complex Networks*, *3*(2), 177–188.

Land, M. (2019). Eye movements in man and other animals. *Vision research*, *162*, 1–7.

Le Meur, O. & Liu, Z. (2015). Saccadic model of eye movements for free-viewing condition. *Vision research*, *116*, 152–164.

LeCun, Y., Bengio, Y. et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995.

Lencastre, P. (2021). EYET.zip. https://doi.org/10.6084/m9.figshare.17049761.v1

Lev, A., Braw, Y., Elbaum, T., Wagner, M. & Rassovsky, Y. (2020). Eye Tracking During a Continuous Performance Test: Utility for Assessing ADHD Patients. *Journal of Attention Disorders*, 1087054720972786.

Li, J., Song, Y., Zhang, H., Chen, D., Shi, S., Zhao, D. & Yan, R. (2018). Generating classical chinese poems via conditional variational autoencoder and adversarial training. *Proceedings of the 2018 conference on empirical methods in natural language processing*, 3890–3900.

Lin, C.-H., Yumer, E., Wang, O., Shechtman, E. & Lucey, S. (2018). St-gan: Spatial transformer generative adversarial networks for image compositing. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 9455–9464.

Lind, P. G., Vera-Tudela, L., Wächter, M., Kühn, M. & Peinke, J. (2017). Normal behaviour models for wind turbine vibrations: Comparison of neural networks and a stochastic approach. *Energies*, *10*(12), 1944.

Lv, Z., Huang, X. & Cao, W. (2021). An improved GAN with transformers for pedestrian trajectory prediction models. *International Journal of Intelligent Systems*.

Mardanbegi, D., Killick, R., Xia, B., Wilcockson, T., Gellersen, H., Sawyer, P. & Crawford, T. J. (2018). Effect of aging on post-saccadic oscillations. *Vision research*, *143*, 1–8.

Mikolov, T., Joulin, A., Chopra, S., Mathieu, M. & Ranzato, M. (2014). Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*.

Mogren, O. (2016). C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*.

Muhamed, A., Li, L., Shi, X., Yaddanapudi, S., Chi, W., Jackson, D., Suresh, R., Lipton, Z. C. & Smola, A. J. (2021a). Symbolic Music Generation with Transformer-GANs. *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*(1), 408–417.

Muhamed, A., Li, L., Shi, X., Yaddanapudi, S., Chi, W., Jackson, D., Suresh, R., Lipton, Z. C. & Smola, A. J. (2021b). Transformer-gan. Retrieved March 25, 2022, from https://github.com/amazon-research/transformer-gan

Ni, H., Szpruch, L., Wiese, M., Liao, S. & Xiao, B. (2020a). Conditional sig-wasserstein gans for time series generation. *arXiv preprint arXiv:2006.05421*.

Ni, H., Szpruch, L., Wiese, M., Liao, S. & Xiao, B. (2020b). Conditional-Sig-Wasserstein-GANs. Retrieved March 28, 2022, from https://github.com/SigCGANs/Conditional-Sig-Wasserstein-GANs

Pan, J., Canton, C., McGuinness, K., O'Connor, N. E., Torres, J., Sayrol, E. & Giro-i-Nieto, X. (2017). SalGAN: Visual Saliency Prediction with Generative Adversarial Networks. *arXiv*.

Perneczky, R., Ghosh, B. C. P., Hughes, L., Carpenter, R. H. S., Barker, R. A. & Rowe, J. B. (2011). Saccadic latency in Parkinson's disease correlates with executive function and brain atrophy, but not motor severity. *Neurobiology of disease*, *43*(1), 79–85.

Rao, Q. & Frtunikj, J. (2018). Deep learning for self-driving cars: chances and challenges. *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*, 35–38.

Rhee, I., Shin, M., Hong, S., Lee, K., Kim, S. J. & Chong, S. (2011). On the levy-walk nature of human mobility. *IEEE/ACM transactions on networking*, *19*(3), 630–643.

Rigas, I. & Komogortsev, O. V. (2017). Current research in eye movement biometrics: An analysis based on BioEye 2015 competition. *Image and Vision Computing*, *58*, 129–141.

Roy, A., Bertrand, S. L. & Fablet, R. (2021). Generative Adversarial Networks (GAN) for the simulation of central-place foraging trajectories. *bioRxiv*.

Salehinejad, H., Sankar, S., Barfett, J., Colak, E. & Valaee, S. (2017). Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*.

Sarkar, A. & Cooper, S. (2021). Generating and blending game levels via quality-diversity in the latent space of a variational autoencoder. *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, 1–11.

Schuster, M. & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, *45*(11), 2673–2681.

Scott, N., Green, C. & Fairley, S. (2016). Investigation of the use of eye tracking to examine tourism advertising effectiveness. *Current Issues in Tourism*, *19*(7), 634–642.

Sims, D., Humphries, N., Bradford, R. & Bruce, B. (2012). Lévy flight and Brownian search patterns of a free-ranging predator reflect different prey field characteristics. *Journal of Animal Ecology*, *81*(2), 432–442.

Steffens, M., Becker, B., Neumann, C., Kasparbauer, A. M., Meyhöfer, I., Weber, B., Mehta, M. A., Hurlemann, R. & Ettinger, U. (2016). Effects of ketamine on brain function during smooth pursuit eye movements. *Human brain mapping*, *37*(11), 4047–4060.

Thompson, V. A. (2021). Eye-tracking IQ: Cognitive capacity and strategy use on a ratio-bias task. *Cognition*, *208*, 104523.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998–6008.

Viswanathan, G., Afanasyev, V., Buldyrev, S., Murphy, E., Prince, P. A. & Stanley, E. (1996). Lévy flight search patterns of wandering albatrosses. *Nature*, *381*(6581), 413–415.

Wadhera, T. & Kakkar, D. (2019). Eye Tracker: An Assistive Tool in Diagnosis of Autism Spectrum Disorder. *Emerging trends in the diagnosis and intervention of neurodevelopmental disorders* (pp. 125–152). IGI Global.

Wang, L., Zhang, W. & He, X. (2019). Continuous patient-centric sequence generation via sequentially coupled adversarial learning. *International Conference on Database Systems for Advanced Applications*, 36–52.

Wedel, M. & Pieters, R. (2017). A review of eye-tracking research in marketing. *Review of marketing research*, 123–147.

Wei, W., Wu, H. & Ma, H. (2019). An autoencoder and LSTM-based traffic flow prediction method. *Sensors*, *19*(13), 2946.

Wiese, M., Knobloch, R., Korn, R. & Kretschmer, P. (2020). Quant gans: Deep generation of financial time series. *Quantitative Finance*, *20*(9), 1419–1440.

Yan, X.-F. & Ye, D.-Y. (2015). Improved bacterial foraging optimization algorithm based on Levy flight. *Computer System & Applications*, *24*(3), 124–132.

Yan, Z., Pei, M. & Su, Y. (2017). Children's empathy and their perception and evaluation of facial pain expression: An eye tracking study. *Frontiers in Psychology*, *8*, 2284.

Yoon, J., Jarrett, D. & der Schaar, M. (2019a). Time-series generative adversarial networks.

Yoon, J., Jarrett, D. & der Schaar, M. (2019b). TimeGAN. Retrieved May 7, 2022, from https://github.com/jsyoon0823/TimeGAN

Yu, L., Zhang, W., Wang, J. & Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. *Proceedings of the AAAI conference on artificial intelligence*, *31*(1).

Yu, Y., Si, X., Hu, C. & Zhang, J. (2019). A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation*, *31*(7), 1235–1270.

Zhang, N. (2020). Learning adversarial transformer for symbolic music generation. *IEEE Transactions on Neural Networks and Learning Systems*.

Zhang, Y., Zhao, X., Fu, H., Liang, Z., Chi, Z., Zhao, X. & Feng, D. (2011). A time delay neural network model for simulating eye gaze data. *Journal of Experimental & Theoretical Artificial Intelligence*, *23*(1), 111–126.