University of Amsterdam

MSc Artificial Intelligence
Master Thesis

# Generating spiking time series with Generative Adversarial Networks:

an application on banking transactions

by

Luca Simonetto

11413522

September 2018

36 ECTS
February 2018 - August 2018

| | |
|---|---|
| *Supervisors:* | *Assessor:* |
| Dr. Amir Ghodrati | Prof. Cees Snoek |
| Prof. Efstratios Gavves | |
| Floris den Hengst | |

ING Netherlands

# Acknowledgements

*I would like to thank my UvA supervisors Efstratios Gavves and Amir Ghodrati, who supervised my work and helped me in finding elegant solutions to the problems I encountered during my thesis work, always with a smile.*

*Thanks to my ING supervisor Floris den Hengst, who was always there when I needed help, not letting me down a single time and always available for some weekend beers with our colleagues.*

*I would also like to thank the extraordinary people that I encountered during my two years of master, many of which I can now call friends and that have made me love the time spent together.*

*Thanks to my family, supporting me no matter what, being there in the rough moments, cheering me for every accomplishment and always waiting for me with a smile when I came back to Italy.*

*Finally, thanks to Valeria, the most amazing and unexpected person that ever entered my life. You showed me how beautiful the world can be together with someone you love, and I will never be grateful enough for it.*

# Abstract

The task of data generation using Generative Models has recently gained more and more attention from the scientific community, as the number of applications in which these models work surprisingly well is constantly increasing. Some examples are image and video generation, speech synthesis and style transfer, pose guided image generation, cross-domain transfer and super resolution. Contrarily to such tasks generating data coming from the banking domain poses a different challenge, due to its atypical structure when compared with traditional data and its limited availability due to privacy restrictions.

In this work, we analyze the feasibility of generating spiking time series patterns appearing in the banking environment using Generative Adversarial Networks. We develop a novel end-to-end framework for training, testing and comparing different generative models using both quantitative and qualitative metrics. Finally, we propose a novel approach that combines Variational Autoencoders with Generative Adversarial Networks in order to learn a loss function for datasets in which good similarity metrics are difficult to define.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Generating data that has similar characteristics as real data using Machine Learning techniques has been an important area of research since the advent of generative models: such approaches leverage hidden semantic features of the input data in order to generate data samples that are realistic enough to be indistinguishable from real samples. Given the improvements made in recent years, namely with the advent of Generative Adversarial Networks [Goodfellow et al., 2014], a new era of data generation commenced.

## 1.1 Problem presentation

The main application in which Generative Adversarial Networks have proven to work well has been image generation [Karras et al., 2017], thanks to the strong spatial relationships of the inputs along with the quickly verifiable generation quality, either with well-known scores or with a simple visual inspection. With limited knowledge required, these models have been able to generate a large number of diversified images, going from simple hand-drawn digits to faces and objects [Radford et al., 2015], resulting in constant improvements and innovations for this type of tasks. Along with images, other research areas have been explored, such as speech synthesis [Donahue et al., 2018] and video generation [Vondrick et al., 2016], again thanks to the strong relationships of the inputs resulting in robust ways of comparing the quality of the generation and an easier data distribution modeling. Having access to a method that is able to produce realistic looking and behaving data can help other research tasks as well, for example by giving a quick way of obtaining more data when some datasets are scarce or not easily accessible [Antoniou et al., 2017], and can help in quickly assessing the quality of a model that operates on the real data without needing to have the latter on hand.

While building Generative Models that work well with the abovementioned types of data is becoming easier, the same cannot be said for more alternative representations that do not express clear relationships or that cannot be easily understood by researchers not having a deep domain knowledge. This work approaches the task of generating data coming from the banking environment, defined as real-valued spiking time series representing monetary transactions of users during a fixed period of time. Differently from typical time series, the data used in our work exhibits a peculiar structure: each feature is either part of a flat region or forms a single spike, resulting in greater difficulty in both understanding the data and developing good generative models to generate it.

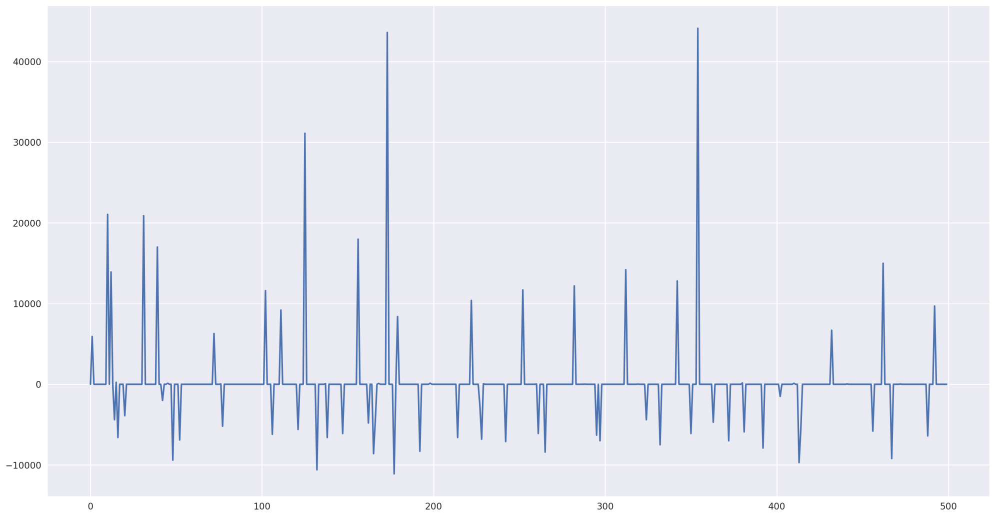

Figure 1.1: Example of a sparse spiking real-valued time series.

Figure 1.1 shows an example of a real-valued spiking time series taken from the dataset used. While some patterns can be visually recognized, most of the behaviors are not of immediate understanding, meaning that an expert is needed to correctly work with this type of data.

## 1.2 Research questions

Having stated the problem that needs to be solved, we define three research questions, that we will try to answer in our work:

- *How can we generate real-valued spiking time series coming from the banking domain?* As this is a data generation task applied to a relatively unseen type of time series, we first want to understand whether this patterns can actually be generated.

- *How can we evaluate the performance of models generating real-valued spiking data?* Given the peculiarity of the domain on which this task is proposed, traditional evaluation metrics cannot be applied, resulting in the need of defining a new method for evaluating the quality of generated datasets.

- *How can we eliminate the need for defining a similarity metric for generative models that require it?* Defining a good similarity metric is difficult without domain knowledge, and developing a method that removes such requirement would result in an easier task for researchers, and the ability to keep working with known models such as Variational Autoencoders.

## 1.3 Contribution

Based on the above-defined research questions, we can identify three main contributions in our work

First, we show how Generative Adversarial Networks can be applied in the task of generating real-valued spiking time series, using convolutional and deconvolutional methods. We apply the Improved Wasserstein GAN model and show that it can learn the data distribution and produce acceptable results, similarly to an alternative formulation that aims at reducing mode collapse and enforces diversification of the latent space. We propose an alternative internal architecture of the generative models used, that allows to generate spiking time series using one-dimensional convolutions and deconvolutions.

Second, we propose a framework for evaluating different generative models, allowing both quantitative and qualitative evaluations of the results. Traditional evaluation metrics used in the literature are not applicable to this problem, resulting in the need of a novel evaluation method. For the quantitative evaluation, we define an additional task in which datasets generated by the various models are compared, by looking at the scores produced by a classifier that is required to distinguish real from generated data. For the qualitative evaluation, we define a visualization task in which bitmaps representations of the generated datasets are compared, allowing a quick analysis of the datasets' quality.

Third, we define a new model that combines a traditional Variational Autoencoder and an Improved Wasserstein GAN critic, resulting in a generative approach that automatically learns a good similarity metric for the VAE model. We show that a generative model that requires a pre-defined similarity metric to be able to be trained can still be used with new datasets that would otherwise make the model collapse into generating a single output. By minimizing the loss of a GAN critic the generative model can now gradually improve its effectiveness at correctly modeling the latent space to accommodate for the input distribution, arriving at generating samples with a quality similar to the other state-of-the-art GAN models proposed.

## 1.4 Thesis structure

In Chapter 2, background knowledge on the used methods is given, along with the abbreviations used to define the various models later in this work. Chapter 3 gives an overview on both the data that has been used and the general experimental procedure followed, along with an indication on the motivation and purpose in the use of the specific methods. Chapter 4 gives an overview of the literature linked to our work, to give an indication of the current progress made in the research field.

In Chapter 5 and 6 the setup for this work and the results obtained are presented, along with an explanation of the metrics chosen to measure the quality of a specific approach. Chapter 7 gives a summary our work, along with final conclusions and possible future directions for this work, the latter given in Chapter 8.

# Chapter 2

# Background

In this chapter, a general understanding of the concepts used in this work is given. Section 2.1 first gives an overview of the concepts regarding the internal architecture used in our approaches (2.1.1), and then describes the theoretical formulations of the generative models that have been experimented with. Section 2.2 gives details regarding the evaluation methods used, both quantitative (2.2.1) and qualitative (2.2.2).

## 2.1 Generative models

### 2.1.1 Convolutional Neural Network (CNN)

When features of the data exhibit spatial relationships, such as pixels in an image, or frames of a video, Convolutional Neural Networks usually [Lecun et al., 1998] are employed. This particular deep learning model applies convolutions and reduction operations to the input volume to extract progressively higher level representations of the data. By using local connections and weight sharing in the internal convolutional kernels, this model achieves translation-invariance, reducing the amount of preprocessing needed on the training dataset along with greatly improved training times compared to traditional Neural Networks.

This model combines two different type of layers in succession, convolutional and pooling layers: the first is used to extract features from the input, while the second is used to reduce its dimensionality in order to both reduce the number of parameters used and to provide an abstraction mechanism for the model. After a number of convolution+pooling operations, this approach often employs fully connected layers, in order to combine local features from the first layers with the global view from the second layers.

Given an input volume $\mathbf{x}$ with $d \times d \times m$ dimensions and a convolutional layer with $k$ kernels each one with size $l \times l$ and depth $f$, the output of the convolution can be expressed as $f$ feature maps each one of dimensionality $d - l + 1$. This feature maps will be then reduced using a pooling method, that usually takes the maximum value of a sliding window of predefined size. If the size of the sliding window is 2, the height and width of the input volume will be halved.

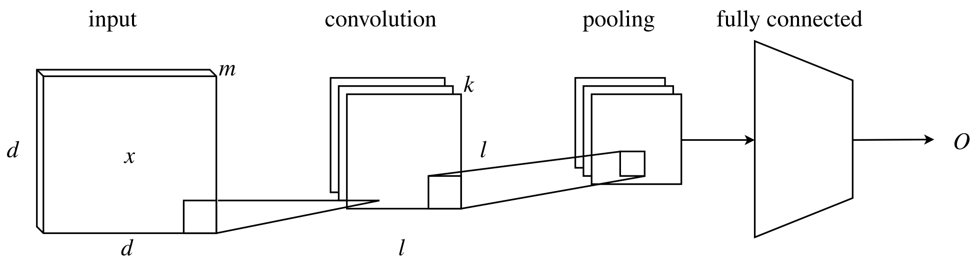

Figure 2.1: Architecture of a Convolutional Neural Network that uses only one convolution+pooling layer.

Figure 2.1 shows the architecture of a Convolutional Neural Network, that uses only one convolutional layer, followed by a pooling layer and a fully connected layer. For simplicity, the dimensions of $\mathbf{x}$ have been kept the same, and the number of filters $f$ has been kept to 1. Convolutional Neural Networks have proven to be useful in many image-related tasks, and in this work, their convolutional concept is used to process time series.

### 2.1.2 Variational Autoencoder (VAE)

First, an Autoencoder is a Neural Network architecture used to learn encoded representations of the input data in an unsupervised manner. To solve this task, an autoencoder is composed by two networks, an Encoder network $E$ with internal parameters $\theta$ and a Decoder network $D$ with internal parameters $\phi$: the first network encodes the input into a smaller encoding space $z$ and the second network takes such encoding as an input and transforms it back to the original. By enforcing a restriction on the dimensionality of the encoding, an Autoencoder is forced to learn an efficient representation of the inputs, in order to be able to reproduce each sample from a scarcer source of information. Such a model is trained by enforcing a loss $\mathcal{L}$ that usually is defined as the Mean Squared Error between the input $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$

$$\mathcal{L} = MSE(\mathbf{x}, \tilde{\mathbf{x}})$$

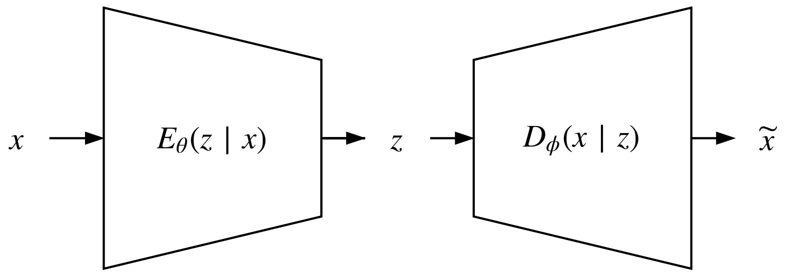

Figure 2.2: Architecture of an Autoencoder.

Figure 2.2 shows the architecture of an Autoencoder: the input $\mathbf{x}$ is passed through the encoder $E$, that outputs an encoding $z$ for the decoder $D$ that is tasked with recreating the input $\mathbf{x}$ as closely as possible with its output $\tilde{\mathbf{x}}$.

A Variational Autoencoder [Kingma and Welling, 2013] is an alternative formulation of the traditional Autoencoder approach that provides a probabilistic interpretation of the latent space observations: the encoder network $E$ now outputs two parameters $\mu_z$ and $\sigma_z$ used in sampling a point in the Gaussian distribution modeled by $D$. Optimal values for the parameters $\theta$ and $\phi$ are now found by minimizing a loss function $\mathcal{L}_{VAE}$ that incorporates both the similarity of the input with its reconstruction $\mathcal{L}_{sim}$, and the KL divergence $\mathcal{L}_{KL}$ of the decoder's modeled distribution and a prior Gaussian distribution $p(z)$.

$$\mathcal{L}_{VAE} = \mathcal{L}_{sim} + \mathcal{L}_{KL} = \mathcal{L}(\mathbf{x}, \tilde{\mathbf{x}}) + KL(D_\phi(x|z) \| p(z))$$

The intuitive way of obtaining a value for $z$ from $\mu_z$ and $\sigma_z$ ($z = N(\mu_z, \sigma_z)$) doesn't have a computable gradient and makes training using Stochastic Gradient Descent impossible: to circumvent this issue a variable $\epsilon \sim N(0,1)$ is added to the model, used to make the sampling operation differentiable by multiplying it to $\sigma_z$.

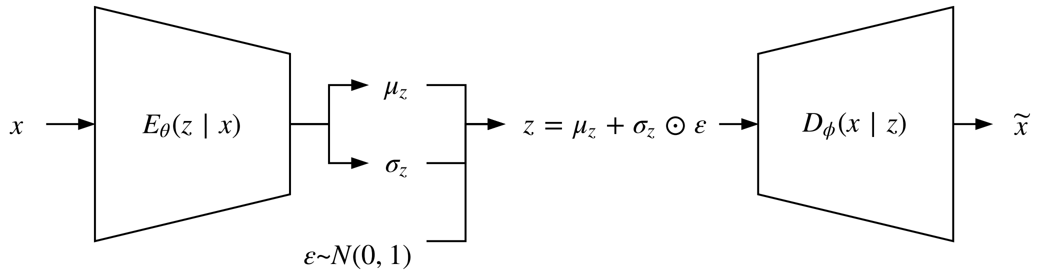

Figure 2.3: Architecture of a Variational Autoencoder.

The resulting model can be seen in Figure 2.3, where $z$ is obtained by adding a value $\epsilon$ to the model sampled from a Gaussian distribution. As with a traditional

Autoencoder, the flow is kept equal, with the difference that now the encoded $z$ is calculated from the values of $\mu_z$, $\sigma_z$ and $\epsilon$, and the model is now able to be used in a generative manner.

### 2.1.3 Generative Adversarial Network (GAN)

A Generative Adversarial Network [Goodfellow et al., 2014] is a deep Neural Network architecture that uses an adversarial training process to ideally approximate any dataset distribution and allow data generation by sampling from such approximation. The model is composed by a generator network $G$ and a discriminator network $D$ with parameters respectively $\theta$ and $\phi$. As with Variational Autoencoders, the generating part of the model (previously called Decoder) is conditioned on a sampled Gaussian variable $z \sim p(z)$ $(G_\theta(x|z))$. The generator network tries to generate samples that follow the real distribution of the data $p(x)$, while the discriminator tries to distinguish between real data $\mathbf{x}$ and generated data $\tilde{\mathbf{x}}$ by outputting a class probability $D_\phi(x) \in [0, 1]$.

Training is done simultaneously on both $G$ and $D$ using an adversarial setting, specifically by alternating updates of $G$ with updates of $D$: $D$ is trained to maximize the probability of assigning the correct class to both real data and generated data, while $G$ is trained to maximize $D$ uncertainty by minimizing $log(1-D(G(z))$. This results in the minmax game with value function $V(G, D)$:

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

At the optimal point of the training, the ideal discriminator $D$ is unable to distinguish real data from fake data $(D(x) = 0.5)$, meaning that the ideal generator $G$ successfully approximated the real data distribution $p(x)$: this point will be reached when the two networks reach a Nash equilibrium of the minmax game.

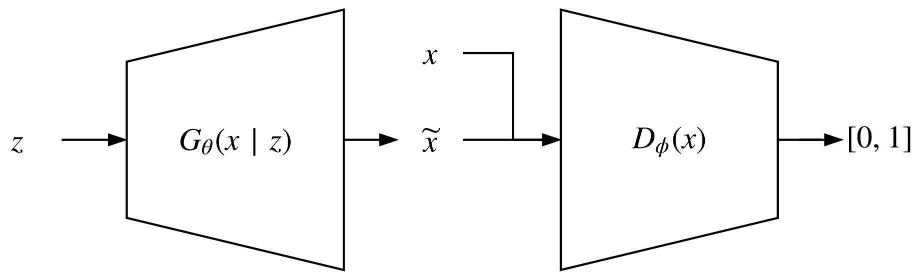

Figure 2.4: Architecture of a Generative Adversarial Network.

The explained GAN architecture is shown in Figure 2.4: the generator model $G$ takes an input a sample $z$ and generates an output $\tilde{\mathbf{x}}$, that will then be used as an input for the discriminator model $D$ along with real samples $\mathbf{x}$. For each sample, the discriminator outputs a value between [0, 1], that will then be used for the adversarial training process.

Due to the fact that the optimal generator and discriminator are ideal, in real setups reaching the Nash equilibrium doesn't mean that the generator matched the real data distribution, but only that the discriminator has reached its maximum classification capabilities for this kind of setup. For this reason, more advanced models have been developed by using alternative formulations, such as the ones discussed in Subsection 2.1.4 and 2.1.5.

### 2.1.4 Wasserstein GAN (WGAN)

The Wasserstein GAN [Arjovsky et al., 2017] uses the training methodology and concepts introduced by the original GAN, but adds the following ideas:

- The loss to be minimized by the discriminator is now the Wasserstein distance between the ground truth and its output. The Wasserstein distance, also called Earth Mover's distance (EM distance), can be interpreted as the "mass" movement that is required for transforming a distribution into another. As the Wasserstein distance is continuous and unbounded, the discriminator now outputs an unbounded value.
- The discriminator is now called critic ($C$), as its output now isn't a probability but an approximation of the Wasserstein metric between the real and generated data distributions.
- In order to use this new distance we need to ensure that some particular constraints are satisfied (K-Lipschitz continuity). For this reason, the weights of the network $C$ are now clipped in a small range of values $[-c, c]$.

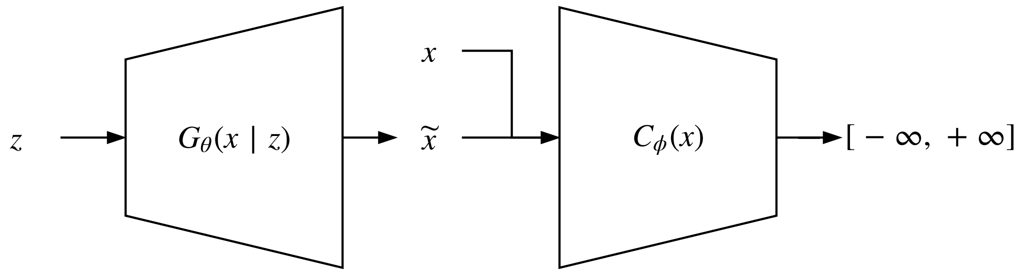

Figure 2.5: Architecture of a Wasserstein GAN.

Those improvements allow for more stable training and a correlation between losses and samples quality (not possible in the original approach), resulting in better results. Figure 2.5 shows the new WGAN formulation, allowing to analyze the difference between it and the original GAN formulation, namely the distinction between $D$ and $C$, and the critic's range of outputs.

### 2.1.5 Improved Wasserstein GAN (WGAN-GP)

Another improvement to the original formulation is to add an additional idea to the Wasserstein GAN model, transforming it in what is known as Improved Wasserstein GAN [Gulrajani et al., 2017]. Weight clipping in the critic is a poor method for ensuring Lipschitz continuity, as the clipping value affects both the speed of convergence and the magnitude of the gradients: too high-value results in a longer time for the weights to reach their clipping limit, while a too small value can result in vanishing gradients. Also, weight clipping reduces the capacity of $C$ to the point that only simpler functions can be learned. The Improved WGAN approach ensures Lipschitz continuity by taking note of the fact that a differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere. This is enforced by penalizing the weights of the critic when the norm of the gradients moves away from 1.

Enforcing a gradient penalty (from which the alternative name of the model WGAN-GP) ensures much faster convergence and a bigger capacity for the critic, resulting in this approach being preferred over the standard WGAN and GAN. Some improvements have also been discussed ([Wei et al., 2018]), but are out of the scope of this work.

## 2.2 Evaluating models

Given the novelty of the task proposed in this work and the difficulties in standard evaluation procedures, we propose an evaluation method that combines both quantitative and qualitative metrics. Details regarding one of the models used in the quantitative evaluation are given in Subsection 2.2.1, while the qualitative evaluation procedure is proposed in Subsection 2.2.2.

### 2.2.1 Support Vector Machine (SVM)

Given its use as an evaluation method to assess the final performance of the proposed generative models, a theoretical formulation is given. A Support Vector Machine [Cortes and Vapnik, 1995] is a supervised method of the maximum margin classifiers family, in which the objective is finding a hyperplane that best separates the samples of two classes. This hyperplane is found by maximizing the margin between points belonging to a class and the decision boundary of the model, namely the perpendicular distance between the points and the hyperplane.

From the fact that SVMs are inherently binary classifiers, when multiple classes are present usually a collection of SVMs are trained for each pair of classes, with the output being the most voted class.

Given that the majority of problems cannot be solved by linearly separating the samples, Support Vector Machines offer the possibility of using nonlinear kernels, projecting the data into a higher dimensional space where a hyperplane search could be more feasible.

The decision boundary determined by an SVM method is determined using support vectors, samples of the data that lie close together and need their distance maximized. Real-world datasets are typically not completely separable, even when using an appropriate kernel. To accommodate for this, SVMs allow for examples from the opposing class through a soft margin classification process. The soft margin concept is used in the Hinge Loss function, defined as:

$$\mathcal{L} = \sum_{i} \max(0, 1 - \mathbf{y}_i(\mathbf{w}^T \mathbf{x}_i + \mathbf{b}))$$

where $\mathbf{x}_i$ is the $i^{th}$ training sample, $\mathbf{y}_i$ is the class of the $i^{th}$ sample, $\mathbf{w}$ and $\mathbf{b}$ are the internal weights and biases of the model.

### 2.2.2 Bitmap representation of time series

Used as a qualitative evaluation procedure, bitmaps generated from time series allow quick visual comparison between samples and datasets. Proposed by Kumar et al. [2005] as a method to better work with large time series datasets, this method allows even out-of-domain users to roughly visually assess whether two or more samples are similar or not to each other. This method transfers the information regarding the frequency of a particular sub-sequence of values to a predefined pixel in a bitmap image, using as its input the symbolic representation (SAX representation [Lin et al., 2007]) of the time series.

The steps taken to generate a bitmap from a particular input are:

- *Generation of the discretized representation of the time series*: this step takes as input a real-valued signal and discretizes each timestep into $n$ equal sized intervals. The number of intervals is chosen beforehand and determines the granularity of the subdivision, higher intervals resulting in a better representation of the original signal but with a lower generalization capability and vice versa. Each interval is identified by a different letter, resulting in an "alphabet" of characters that will determine the new representation. In order to ease the process of bitmap generation, $n$ is usually chosen to be a perfect square, such as 4, 9 etc., allowing square images to be created.

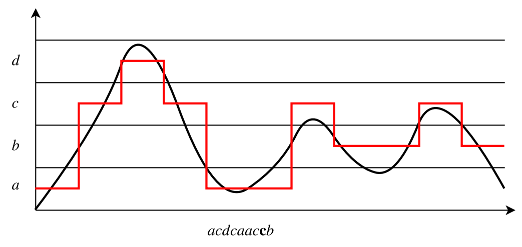

Figure 2.6: Discretization phase of a time series ($n = 4$). The time series is shown in black, the discretization is shown in red.

Figure 2.6 shows a visualization of the discretization process, where the representation with $n = 4$ of a real-valued time series is shown in red. The resulting discretized time series is *acdcaacbbcb*.

- *Count of sub-words in the discretized signal using a sliding window approach*: in this step a sliding window of length $L$ is passed through the discretized time series, outputting a series of words with relative counts indicating how many times a word occurred. For example, for the code *acdcaacbbcb* and a sliding window of length 2, the result would be $ac = 2, cd = 1, dc = 1, ca = 1, aa = 1, cb = 2, bb = 1, bc = 1$.

- *Bitmap grid generation*: in order to populate each pixel in the bitmap, a structured subdivision of the 2D space is defined, by taking into account both the length $L$ of the sliding window and the size $n$ of the alphabet used. The bitmap is divided into $n$ quadrants each one representing a starting letter and containing sub-quadrants representing a possible word. Each subdivision iteration is called Level.

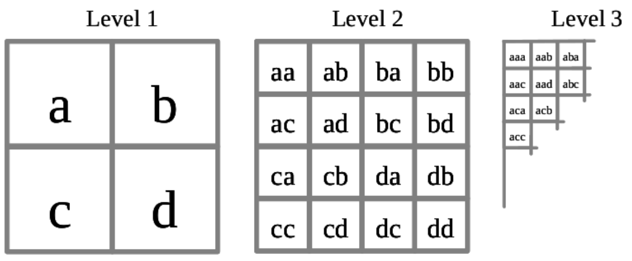

Figure 2.7: Levels of subdivision of a timeseries bitmap based on an alphabet with $n = 4$.

Figure 2.7 shows an example of a subdivision of a bitmap for a time series, using an alphabet of 4 letters and a sliding window of length 1, 2 and 3 respectively. Each element of the grid will result in a colored pixel.

- *Bitmap colorization*: the final step, consists in coloring each pixel with the value being the number of times the respective word has been encountered in the timestep. After this step, all the values are normalized in order to obtain an image that can be shown by either using grayscale or other color gradients.

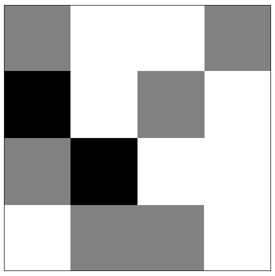

Figure 2.8: Examples of bitmap generated from the time series *acdcaacbbcb*, using an alphabet of $n = 4$ characters and a sliding window of length $L = 2$.

Figure 2.8 shows the resulting bitmap when this method is applied to the time series *acdcaacbbcb* using $n = 4$ and $L = 2$. Although not very informative due to the short input, application on longer time series allows quick semantic separations using only visual inspection.

# Chapter 3

# Approach

In this chapter, we give a description of the approach applied in this work. First, in Section 3.1 we present the dataset used, along with details regarding various choices made and steps taken to make it suitable for our task. Section 3.2 gives an overview of the generative models that proposed in this work, namely two different GAN implementations and our novel approach. In Section 3.3 we give a description of the comparison process, giving details regarding both the quantitative and qualitative evaluations that have taken place.

## 3.1 Dataset

After an initial research process aimed at determining the best dataset to use for the proposed task, the most fitting one has been chosen as being the Berka dataset [Berka and Sochorova, 1999], a relational database containing anonymized Czech bank transactions, account info, and loan records released for the PKDD'99 Discovery Challenge. This dataset has then been parsed in order to extract a list of 4500 bank accounts each one defined as a list of daily inbound or outbound transactions spanning from the $1^{st}$ of January 1993 to the $31^{st}$ of December 1998. This resulted in effectively 2190 values for each one of the 4500 bank accounts.

Each value in this dataset can be classified into three different types: saving, expense, and flat. Savings occur when an amount of money gets transferred into an account (positive value), while an expense occurs when it gets transferred from the account (negative value). When neither of those transaction types occurs in a particular day, the time series has a value of 0, defined as flat for later reference. Figure 3.1 shows one of the 4500 different parsed time series appearing in the Berka dataset: it can be seen how it's vastly different from typical time series, due to the spiking behavior that brings the values to zero after every time a spike occurs. This results in a much harder understanding of the patterns and

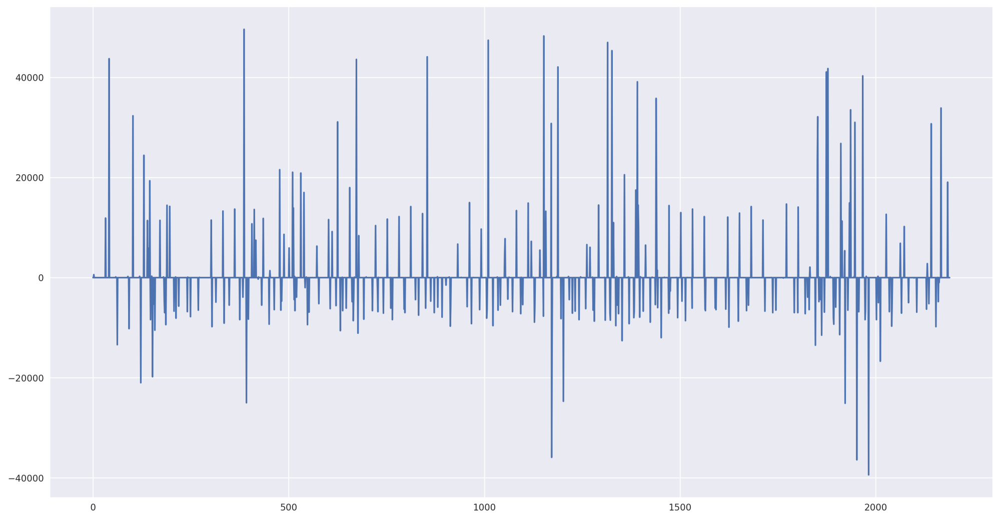

Figure 3.1: Sample from the 4500 parsed time series.

inter-relationships of the time series values, possibly resulting in a harder data generation task. Subsection 3.1.1 gives an overview of the main feature of this dataset, namely its spikiness.

### 3.1.1 Spikiness

With regards to the spikiness of the data, there are 810449 nonzero values, out of the total 9855000 possible values. This results in 8.22% of values appearing as either a saving or an expense, leaving the remaining 91.77% as zero values.

By analyzing the ranges of values for both savings and expenses, a better overview of the dataset can be done. First, every value appears in the range [-87400.0, 74812.0], with a mean of 689.72 and a standard deviation of 11595.909.
Figure 3.2 shows the counts for each nonzero value appearing in the Berka dataset: it can be seen as the distribution resembles that of a heavily skewed distribution. The green line indicates the mean value, while the red lines indicate one standard deviation from the mean.

### 3.1.2 Data preparation

In the financial world, having data that spans many years in the future is not usually required. In addition, longer time series would require longer training times, along with lower performance due to the bigger amount of information required to be learned. Finally, this dataset used as it is would provide enough

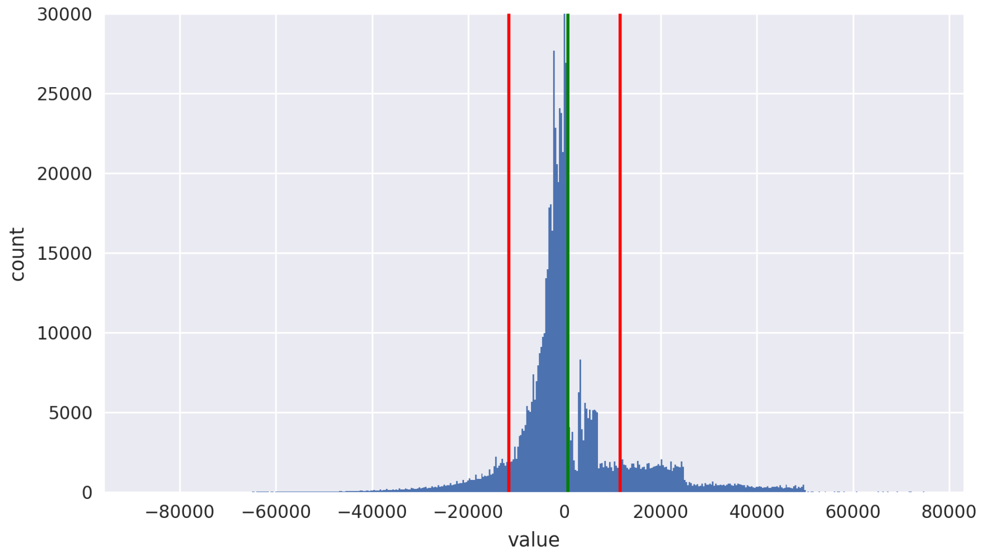

Figure 3.2: Histogram of the nonzero values appearing in the dataset.

samples for a complete training process. For this reasons, we have decided to limit the total length of the input data to a smaller time frame, resulting in an interval of 3 months per sample. This resulted in a new dataset composed of 108000 samples with 90 timesteps each

## 3.2 Generative models

Having defined the dataset and the preprocessing steps applied to it, we can now define the generative models used in this work. First, in Subsection 3.2.1, we motivate the choices made regarding the general approach followed to tackle the problem. Following this, in Subsection 3.2.2 the models used and architectural choices made for each one are explained.

### 3.2.1 Initial architectural choices

An important step in this work has been the initial choice on the model architecture to use: by keeping the same architecture among multiple models, the differences in performance due to different architectures can be removed, resulting in fewer variables contributing to different results. A preliminary investigation indicates that recurrent architectures, such as LSTMs, seem to be a sub-optimal choice for this type of data: when dealing with time series data, LSTM provides a quick and reliable architecture due to their theoretical strengths against inputs with one dimensional relationships. In this case, however, having sparse spikes to be memorized results in big fluctuations of traditional loss functions due to the big error coming from even the slightest shift in the positioning of the generated spikes. Moreover, training LSTM architectures is much slower in most cases, and coupling this architecture with the Generative Adversarial Network approach would slow down convergence even further. Finally, LSTM architectures suffer from long time dependencies, meaning that longer time series will be harder to model using this approach.

Traditional architectures that use fully connected layers have the problem of requiring many parameters: while they are easy to implement and don't require many hyperparameters, this architecture suffers from the high number of internal trainable parameters. As Generative Adversarial Networks need to be trained longer than other methods due to their adversarial setting, having a slow architecture translates in much longer training time. Along with this factor, fully connected architectures would lose spatial information of the input time series, resulting in less information that the model can use to learn the data distribution.

For the above reasons, we have opted to approach the task of generating time series with the use of one-dimensional convolutions: compared to recurrent neural networks, this architecture is faster to train, scales better as the time series length increases and is still able to exploit the spatial information of the inputs. Compared to feed-forward architectures, convolutional architectures require less trainable parameters and don't lose spatial information, resulting in faster training and supposedly better generation.

Generative Models usually work by combining two sub-models in which one acts as the final generator. As they usually operate similarly to each other but in reverse, we apply deconvolution and upsampling operations for the generator and convolutions and max-pooling for the supporting model. This architectural choice removes unnecessary factors that could lead to different performances, as discussed above.

### 3.2.2 Approaches

Here we define the approaches for spiking time series generation using the Generative Adversarial Network formulation: the first one simply is an application of the Improved Wasserstein GAN, while the second allows the critic to make decisions using more information from the generator model. the third approach defines our new model that uses a Variational Autoencoder that learns a similarity metric given by an Improved Wasserstein GAN critic.

**Improved Wasserstein GAN (WGAN-GP)**: as indicated in Section 1.3, this work uses a particular version of the GAN approach, in particular, the Wasserstein GAN with Gradient Penalty (WGAN-GP). While keeping the original formulation where a generator and a discriminator network compete between each other to optimize their loss, convergence speed and generation quality are improved thanks to the alternative loss for the discriminator (now called critic) and the added constraints. To keep the experiments as unbiased as possible, the prior chosen for the generation has been settled to a Gaussian distribution, the same as the one for the VAE model and the other GAN approaches.

**WGAN-GP with packed inputs**: given preliminary experiments and the particular structure of the data, we noticed that one problem that could arise during training is having a collapsed generator: this issue is caused by the lack of contextual information to the critic, that can make a prediction only by looking at one sample at a time. For this reason, a simple technique proposed by [Lin et al., 2017] has been used: as the critic doesn't keep any memory between different training mini-batches, determining that the generator model has collapsed only given an output is almost impossible, resulting in a generative model that learns to produce one or few samples that are good enough just to fool the critic. This phenomenon can be mitigated by increasing the amount of information presented to the critic model, in this case by stacking one or more real or generated samples to the input at each training step. An important thing to note is that the added samples to the critic's inputs are coming from the same generative process as the original inputs, meaning that real samples are added to real inputs and generated samples are added to generated inputs. This gives the critic a broader view on the generative capabilities of its competitor and is able to make better decisions on the base of its new knowledge.

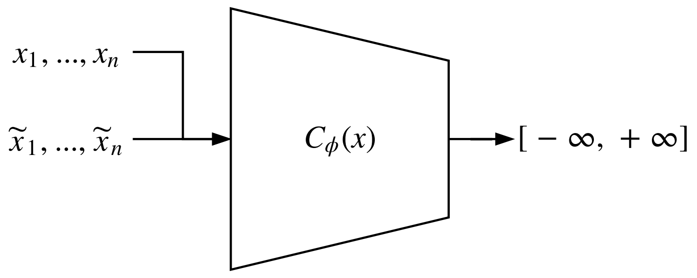

Figure 3.3: Architecture showing how packing changes the input for the WGAN critic.

Figure 3.3 shows how this idea is implemented: by stacking more samples to the traditional real or fake input, the critic is now able to make more informed decisions. As this approach only adds dimensions to the input, the rest of the model is kept the same, meaning that both the training times and the number of parameters are not too affected.

**Variational Autoencoder with learned similarity metric**: from the first preliminary experiments done, we have noted that an issue exists when generative models that use a pre-defined similarity metric are trained: the losses used forced the approaches to try and minimize as much as possible to overall penalization given, without generating spikes in order to avoid loss increases. We then associated this behavior with the wrong choice of similarity metric, even though multiple different losses have been tried.

In order to combat this phenomenon, we propose an application of a Variational Autoencoder that is capable of finding a suitable loss by learning one from scratch, similarly to the work of [Larsen et al., 2015]. Our approach works by transferring the task of determining what loss needs to be minimized to an Improved Wasserstein GAN critic, that gradually learns better representations of the inputs to be able to minimize its GAN loss. By training both the VAE and the critic models together, the data generation process is able to progressively improve as the training continues. Architecturally, the combined model is composed by a traditional VAE, in which an input is encoded into a mean vector $\mu_z$ and a variance vector $\sigma_z$ by an encoder network $E$ with parameters $\theta$. The decoded sample is generated by sampling a latent vector $z$ using the encoding vectors and a random vector $\epsilon$ in a Variational Autoencoder fashion, and passing the result in a decoder network $D$ with parameters $\gamma$. Along with this, an Improved Wasserstein GAN critic is used, of which the theorization details have been described in Section 2.1.5. The training process of this combined model consists of two phases, the first one being the Variational Autoencoder training and the second one being the critic model training:

The Variational Autoencoder model is trained by minimizing similarity loss (Mean Squared Error) $\mathcal{L}_{sim}$ between the output of the critic model when a real sample $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$ are given. By being able to minimize this error, the VAE model is able to produce reconstructions that have the same features for the critic model, that with an ideal critic would correspond to having learned a perfect encoding for the dataset. Additionally, this model is also tasked to minimize a novelty loss $\mathcal{L}_{nov}$, for which only the decoder $D$ is used: by applying the traditional loss defined when training a GAN generator, we can optimize the decoder model to fool the critic using only sampled vectors. With an optimal critic model, this would translate into having learned the dataset distribution.

The two abovementioned losses, $\mathcal{L}_{sim}$ and $\mathcal{L}_{nov}$ , are combined into a single loss $\mathcal{L}$, by using a weighting parameter $\gamma \in [0, 1]$ that allows to choose to give more importance to the autoencoding properties of the model, or its ability to generate novel samples. If $\gamma$ is equal to 0, the training translates to a standard GAN training, while if it's equal to 1, the training translates to a VAE training.

$$\mathcal{L} = \gamma\mathcal{L}_{sim} + (1-\gamma)\mathcal{L}_{nov}$$

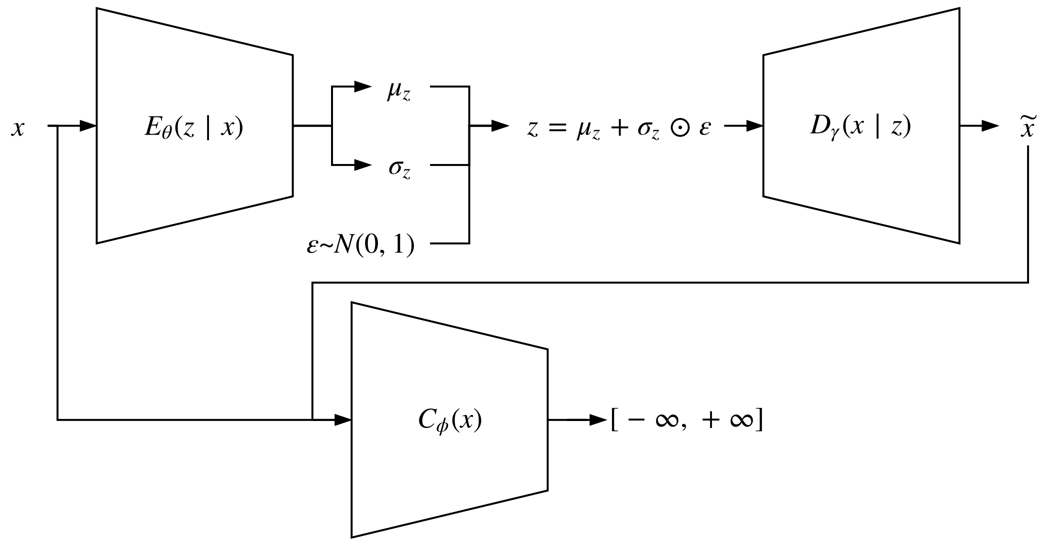

Figure 3.4: Architecture for the VAE with learned similarity metric, in particular for the VAE training phase.

Figure 3.4 shows the architecture of the proposed model, in particular when the VAE model is being trained: in this case both an input $\mathbf{x}$ and its reconstruction $\tilde{\mathbf{x}}$ are separately given to the critic model, in order to obtain two outputs and calculate the similarity loss $\mathcal{L}_{sim}$. At the same time, the decoder $D$ is trained in a GAN fashion, using sampled inputs $\mathbf{z}$.

In order to force the VAE model to minimize the correct losses, the critic model is trained as in a usual GAN setup, in order to minimize the errors made on the inputs. This allows the combined model to continue improving in an alternated way, ideally resulting in a Variational Autoencoder that is able to maximize a perfect critic's uncertainty.
Figure 3.5 shows the critic's training phase, in which the computed gradients are stopped before being backpropagated through the VAE model.

To ensure that the proposed model is correctly trained and converges towards the optimal state, we can analyze the various situations in which the model could be during training:

- *The critic's outputs for both the input sample and its reconstruction are the same, but their value is wrong*: in this case, the problem lies in the critic,

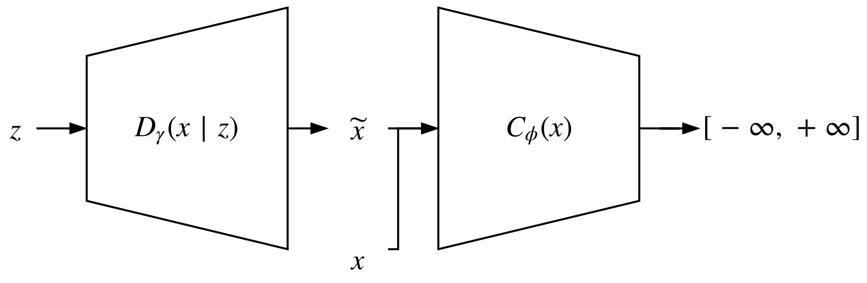

Figure 3.5: Architecture for the VAE with learned similarity metric, in particular for the critic training phase.

that provides a wrong estimation of the Wasserstein distance for the samples. This problem is solved during the critic's training phase, in which it gets optimized in distinguishing real from generated data.

- *The critic's outputs for the input sample and its reconstruction are different*: in this case, the problem lies in the VAE, that has trouble optimizing the loss given by the critic. This problem is solved during the VAE's training phase, in which the difference between the outputs of the critic for inputs and reconstructions are minimized.
- *The outputs for both the input sample and its reconstruction are the same, along with a correct value*: this case could happen when either the critic doesn't have the predictive power to distinguish real samples from generated samples, or when the VAE model has learned to approximate the real data distribution. In the first case, more training of the combined model should gradually improve the final generation capabilities.

Given the above architectural and model details, we can state some benefits in using our proposed model with respect to its alternative formulation proposed in [Larsen et al., 2015]: first, the use of an Improved Wasserstein GAN critic should improve both the convergence speed and quality of the results when compared with a standard GAN discriminator. Second, our approach needs to minimize a similarity metric given by only the output of the critic model, removing the need for complex manipulation of the hidden-representations emerging in the critic. Third, the fact that we are minimizing differences between two Wasserstein distances (WGAN critic) instead of two probability distributions (GAN discriminator) alleviates the problems related with output saturation and restricted range of values.

Finally, one benefit of using this model when compared with traditional GANs, is the type of loss optimization done: with traditional GANs, we are training the generator to fool the discriminator, but without exact information on how to do it. In our case, however, we are forcing the VAE model to minimize the differences in scores for the critic, pushing it to learn better discriminative features as quick as possible. This sustained training methodology should force the critic into converging faster, and reduce the amount of iterations needed for the the model to be able to generate good results.

Regarding the proposed approach more generally, using a Variational Autoencoder instead of a simple generator should achieve a better latent space structure, given the additional constraints imposed on the model. Our approach also allows using Variational Autoencoders when good similarity metrics are absent or are difficult to define, broadening the range of application of generative models based on similarity metrics.

## 3.3 Comparison framework

Given preliminary experiments in which simple generative models are compared, we have noted that typical evaluation metrics are not appropriate in this setting: the fact that spiking time series are a very niche type of data translates in difficulties in determining what is a good performance metric to use. A factor that restricts even more the choice of a good score to use is the fact that this dataset comes from a very specific domain, resulting in problems when determining which data characteristics to weight more with respect to others, such as spike height, spacing, presence or absence of spikes etc. Traditional evaluation metrics used to determine the quality of GANs, such as the Inception Score [Salimans et al., 2016] cannot be applied to our dataset, as it doesn't consist of proper images. Furthermore, more work has been done in order to define how good traditional scoring metrics are, resulting in the insight that many of them are insufficient for giving strong conclusions on the performance of a generative model [Shmelkov et al., 2018]. For this reason, we propose an evaluation framework that can be applied in an end-to-end fashion, determining the quality of a generative model by the means of two distinct evaluation procedures, one quantitative and one qualitative. In this way, the models can be compared in multiple ways, where each one approaches the problem from a different view.

### 3.3.1 Quantitative evaluation

The quantitative evaluation process proposed, similarly to [Esteban et al., 2017], aims at giving as a result a series of scores for each generative model by determining how good the generated data is when used for some particular tasks. For our work, these scores are given by training additional models on a classification task, using a mixture of real and generated samples as training dataset. By analyzing their performance, multiple metrics can be calculated and will be used when determining the efficacy of the proposed generative models.

**Dataset of origin classification task**: in this task, a classifier is required to determine whether a sample belongs to the original dataset or the generated one. If the generative model is able to closely recreate the original distribution, the classifier will have a lower performance, due to the less frequent dissimilarities between the real and generated samples. Given a trained model, we decided to use *accuracy* and *f1_score* as output metrics, that will then be used for comparison. In order to give a stronger measure of performance from the evaluation framework, we decided to employ two different models in parallel: one being a feedforward Neural Network and the other being a Support Vector Machine. As with all the other architectures involving Neural Networks, to keep the overall consistency, the convolutional approach has been kept for the first classifier.

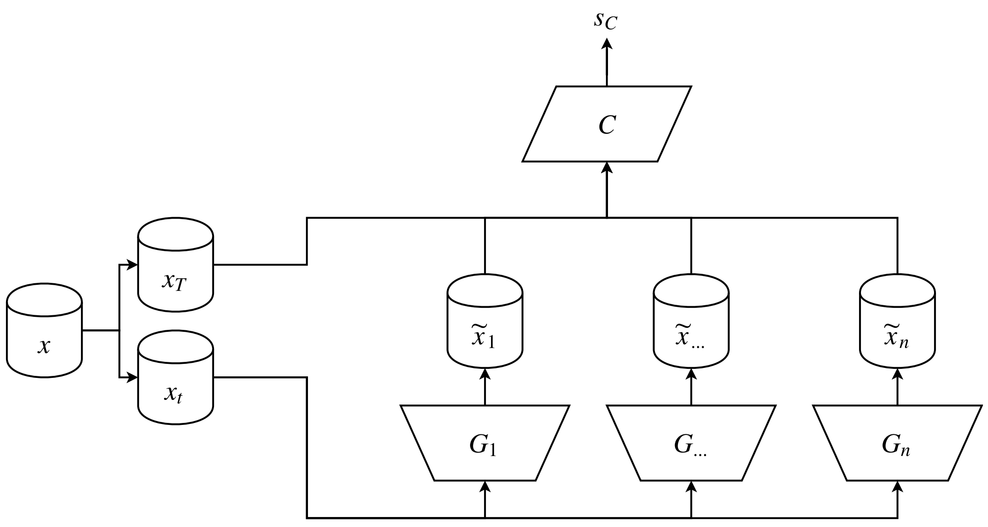

Figure 3.6: Overview of the comparison framework with quantitative classification task: every generative model is trained using the real data, and a mixture of left-out real data and generated data is used for training the classifiers.

Figure 3.6 shows the structure of the comparison framework, specifically for the proposed classification tasks: a training set $\mathbf{x}_t$ composed of samples taken from the real dataset is used to train the different generative models $G_1, ..., G_n$, that at the end of the training process output a generated dataset $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n$. This generated datasets, along with a test dataset $\mathbf{x}_T$ coming from the real data, are separately used for a classification task $C$ outputting a series of scores $s_C$. Such scores are then used for comparing the generative performance of the various models proposed.

### 3.3.2 Qualitative evaluation

Along with the quantitative evaluation methodology described above, we also define a qualitative evaluation method, that allows a supporting analysis of the results. This evaluation is done by analyzing the bitmaps generated using the samples from each generated dataset and comparing them to the bitmaps generated using the samples from the held out test set. In order to ease the comparison work, all the bitmaps coming from the same dataset are aggregated in order to be able to analyze an entire dataset at a time: in order to do so, the mean and std bitmaps are calculated, by using each bitmap pixel's value throughout the dataset. After this operation, the number of bitmaps to consider is decreased from around 200k to 12 in our case, two for each dataset. Although removing details regarding the behavior of every single time series, this method defines an efficient way to make comparisons, even when no domain knowledge is given to the user.

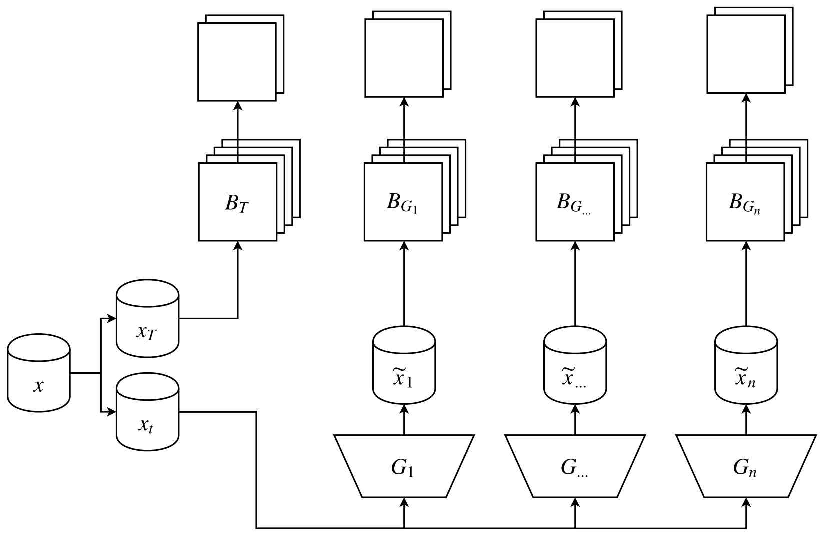

Figure 3.7: Overview of the comparison framework with qualitative classification task: each dataset is used to generate one bitmap for each time series, then combined to obtain a mean and an std bitmap.

Figure 3.7 shows the structure of the comparison framework, specifically for the qualitative evaluation process. Each generated dataset $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_n$, along with the test set $\mathbf{x}_T$ is used to generate one bitmap for each time series $B_{G_1}, ..., B_{G_n}$ and $B_T$. The bitmaps generated from the same dataset are then aggregated together in two bitmaps, one indicating the mean and the other indicating the standard deviation.

# Chapter 4

# Related work

In this Chapter, we give a brief description of the work related to ours.

Since the advent of Generative Adversarial Networks [Goodfellow et al., 2014], constant improvements in the field of data generation have been made. By proving a strong generative alternative to traditional models such as Variational Autoencoders proposed by [Kingma and Welling, 2013], GANs have gained more and more attention from the scientific community especially in the field of image generation. For example, tasks such as cartoon images generation [Jin et al., 2017], image inpainting [Demir and Ünal, 2018] and text to image synthesis [Reed et al., 2016] proved the effectiveness of the new approaches, driving more researchers in such directions.

Given the success on tasks related to image generation, research has been done on various other fields, such as music [Mogren, 2016] and text generation [Fedus et al., 2018]: also, thanks to the spatial correlation of nearby values in time series, approaches employing RNNs for generation of this type of data have been developed. The work of [Esteban et al., 2017] uses Recurrent Conditional GANs, that add class information coming from the time series to aid in the training of the model. It exploits the recurrent nature of RNNs and can be thought of as the work that tries to solve a task that is the most similar to ours.

As noted in Chapter 1, we use an improved version of the traditional GAN [Goodfellow et al., 2014]: such model uses a Wasserstein loss measure proposed by [Arjovsky et al., 2017] and adds constraints on the gradients as defined by [Gulrajani et al., 2017]. Subsequent work has been done to further improve the training process like in [Wei et al., 2018], but as this work approaches a new domain proof of concept, we decided to not include it.

With regards to the learned similarity metric proposed in this work, we refer to the work of [Larsen et al., 2015], proposing this technique by combining a standard GAN and a VAE model. Contrary to our work, this approach uses intermediate representations of the discriminator for the similarity metric calculation, simplified in our work by simply taking the critic's output.

Regarding the evaluation of the performance of GANs, many different metrics have been defined when working with images, such as the Inception Score (IS) [Salimans et al., 2016]: this techniques work well with images, but fall short when the data is of a different type. For this reason, a new technique proposed by Esteban et al. [2017] allows a model to be quantitatively assessed by the means of a proxy model, that is required to do a task for which quantitative metrics are known: our work takes this idea and applies it in a domain for which no labels for different samples are given, resulting in different tasks to be defined.

Finally, qualitative evaluation of generative models is relatively easy when working with images: techniques such as latent space interpolation and nearest neighbor comparison are easy to use, given the visual representability of the data that can be immediately checked. When working with time series, this methods cannot be applied as easily, especially when the data comes from a very specific domain such as ours. For this reason, visualization techniques such as bitmaps generation from time series [Kumar et al., 2005] have been developed, easing the work of the researcher by allowing a quick semantic subdivision of the samples.

# Chapter 5

# Experimental setup

In this chapter, a detailed explanation of the experiments done to assess the performance of our proposed approaches is given, along with all the architectural details, hyperparameters used and evaluation setup. First, Section 5.1 defines all the preprocessing steps taken to obtain usable data, Section 5.2 indicates the architectural details regarding the generative models used in the experiments, along with the hyperparameters chosen. Section 5.3 lists the architectures used for the models used in the comparison framework.

## 5.1 Dataset

As indicated in Section 3.1, the initial dataset is composed by 108000 time series each one containing 90 time steps, where the 90 elements account for 3 months worth of transactions. In order to use such dataset, we normalized it in the $[-1, 1]$ range: as already mentioned, this dataset contains some outliers with very high positive and negative values, and that would bring almost every other value to 0 if simple normalization had been done. To mitigate this problem, we clipped every value in the range given by the $1^{st}$ and $99^{th}$ percentiles ($-7300$ and 11739 respectively), to preserve as much as the variance as possible while considerably shrinking the range of possible values. After this step, we normalized the values in the $[-1, 1]$ range using standard methods.

## 5.2 Generative Models

In this section the details regarding the generative models used are shown. First, in Subsection 5.2.1 we define the baseline models that will be compared with our work. In Subsection 5.2.2 we define the convolutional approach architecture that will be used in all the models employing Neural Networks, with specific architectural details shown in Subsection 5.2.3. All the hyperparameters chosen for each model's training are defined in Subsection 5.2.4.

### 5.2.1 Baselines

In order to be able to determine the quality of the generative model proposed in this work we couple our methods with two other approaches, namely handcrafted generation and Variational Autoencoder, allowing comparisons using the evaluation framework defined in Section 3.3.

**Handcrafted generation**: first, a fully handcrafted method for generating time series has been developed. This model would represent the alternative approach in which an expert is given the task of generating the data given some specific knowledge. Each time series is generated by placing a spike at every timestep with a probability given by the probability of a spike occurring in the original dataset. When a particular time step is chosen as containing a spike, a random choice between it being positive or negative is made (50/50 split). Finally, the spike's value is determined by a Gaussian sampler that takes the mean and standard deviation of the values of the dataset at that timestep as parameters. Although naive, this model is an intuitive generation process that could come to mind when dealing with this type of data.

**Variational Autoencoder**: second, we implemented a Variational Autoencoder by using the convolutional and deconvolutional ideas presented in Section 3.2. This approach, being well understood and studied by the research community, poses a solid comparison method that is usually seen as one of the first choices in many data generation experiments. As this method uses a fixed similarity metric to determine how close two time series are from each other, we needed to choose one. For our experiments, the Mean Squared Error loss has been used (MSE), mainly due to its widespread application in many different tasks. From initial experimentation, it has been found that this particular type of data poses a strong obstacle to VAE's ability to converge, even when other loss functions are used. Developing a problem-tailored similarity metric would require domain knowledge that we don't possess, along with resulting in a model too dependent on the type of data used.

### 5.2.2 Base architecture

Opting for a convolutional approach to tackle this task resulted in both a decrease in learnable parameters and scalability of the approach: in the case where a longer or shorter time series is used, a simple addition or deletion of a convolutional layer will allow using all the models in the same fashion. The proposed architecture is divided into two separate components, a convolutional block and a deconvolutional block.

**Convolutional block**: used when the model needs to process an input time series, either for the VAE encoder or the GAN critic.

| layer | layer type |
|---|---|
| 0 | Conv1D |
| 1 | Activation(LeakyReLU) |
| 2 | MaxPooling1D |

Table 5.1: Convolutional block architecture.

Table 5.1 shows the details of the convolutional block used for the generative models: as we are working with time series, both the Convolution and the MaxPooling layers are of the 1D variant. Between them, a LeakyReLU activation function is used, to add a nonlinearity to the model defined as:

$$f(x, \alpha) = \begin{cases} \alpha x & \text{for x} < 0 \\ x & \text{for x} \geq 0 \end{cases}$$

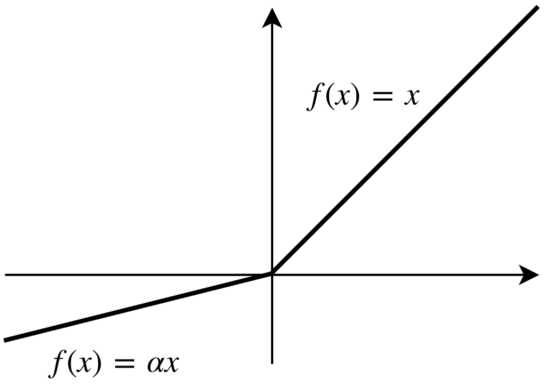

Figure 5.1: LeakyReLU activation function.

Figure 5.1 shows a plot of the chosen LeakyReLU activation function, where it can be seen how, unlike traditional ReLU activations, when $x < 0$ the function differs from 0. This function has been chosen to avoid the neuron "dying" in traditional ReLUs, meaning that the output is consistently 0. The $\alpha$ parameter indicated is defined a priori, and its value is indicated in Table 5.3.

**Deconvolutional block**: used when the task is the opposite of the above, meaning that from an input with lower dimensionality the model is required to produce a full time series as an output.

| layer | layer type |
|---|---|
| 0 | Conv1D |
| 1 | Activation(LeakyReLU) |
| 1 | UpSampling1D |

Table 5.2: Deconvolutional block architecture.

Table 5.2 shows the architecture of a deconvolutional block: very similar to the convolutional one, this block uses an UpSampling1D layer to double the size of the current input, after having been processed by a Conv1D layer with LeakyReLU activation function.

**Batch Normalization**: in order to speed up the training of the generative models, along with improved convergence, we added a Batch Normalization layer between the Convolutional and Activation layers, when the generative model specifically allowed its use in their theoretical formulation. This addition will be noted in the following tables with the keyword *batchnorm*.

**Common architecture parameters**: for all the models, we decided to use the same parameters when they are conceptually similar or identical, such as the latent space dimensionality or the convolution parameters.

| Parameter | value |
|---|---|
| Latent space dimensionality | 2 |
| Latent-Conv intermediate dimensionality | 15 |
| Conv1D kernel_size | 32 |
| Conv1D strides | 3 |
| MaxPooling pool_size | 2 |
| UpSampling size | 2 |
| LeakyReLU alpha | 0.2 |

Table 5.3: Common architectural parameters used in the generative models proposed.

The second entry in Table 5.3, *Latent-Conv intermediate dimensionality* refers to the intermediate dimensionality in which the input from the latent space is brought, before being passed to the subsequent deconvolutional blocks: this ensures that an alteration to the dimensionality of the latent space doesn't require modifications of the subsequent layers, and abstracts the input.

### 5.2.3 Model specific architectures

Having defined the basic convolutional and deconvolutional blocks used as a baseline, along with notes regarding the Batch Normalization layer used and the common hyperparameters, we can define the architectures for all the generative models proposed in this work.

**Handcrafted generation**: defined as a fully handcrafted model, this approach doesn't require any of the abovementioned blocks, as the output is generated by analyzing the statistics of the source dataset and not using Neural Networks. In order to generate time series, this model calculates for each of the 90 time steps the probability of a spike, along with the mean and the variance of both positive and negative spikes. The data generation follows Algorithm 1
Algorithm 1 shows the procedure for generating a spiking time series $\mathbf{x}_{res}$, given an input dataset $\mathbf{X}$: for each time step, if a sampled random number $p_{spike}$ is below the spike probability $\hat{\mathbf{x}}_i$, a choice is made between placing a positive or a negative spike ($p_{positive}$). Then, the value for that element is sampled from a Gaussian distribution with mean and std given by the dataset's statistics for either the positive ($\bar{\mathbf{x}}_{p,i}$, $\tilde{\mathbf{x}}_{p,i}$) or negative ($\bar{\mathbf{x}}_{n,i}$, $\tilde{\mathbf{x}}_{n,i}$) spikes.

**VAE**: the Variational Autoencoder model is the second generative approach used for comparison. The complete architecture can be divided into two separate parts, being the encoder and the decoder models.
Tables 5.4 and 5.5 show the architectures of both the VAE encoder and decoder: for the first, it's important to note how the convolutional output is flattened, in order to be further passed in the Network's fully connected layers, and for the second is important to note the transitioning fully connected layer between the input and the first convolutional layer, with fixed dimensionality as explained above. Finally, it's also important to note the final convolutional layer before the fully connected layer that acts as an output: by using a kernel size of 1 we can transform the output to one that's compatible with the dense layer, allowing also to generate time series with a different length if needed without resulting in a totally different architecture. Due to the nature of the inputs, time series with multiples of 30 timesteps should be used (to always represent multiple numbers of months), and our implementation allows different lengths to be tried if necessary.

**WGAN-GP**: our first generative model employing GANs, in this approach the

**Algorithm 1** Handcrafted generation

```
  Generation of time series given the statistics of the input dataset.
  $\bar{\mathbf{x}}_p$ is the mean vector for positive spikes
  $\bar{\mathbf{x}}_n$ is the mean vector for negative spikes
  $\tilde{\mathbf{x}}_p$ is the std vector for positive spikes
  $\tilde{\mathbf{x}}_n$ is the std vector for negative spikes
  $\hat{\mathbf{x}}$ is the probability vector
  $\mathbf{x}_{res}$ is the output vector
  each one has a dimensionality of (1, 90)
Input: dataset $\mathbf{X}$ with dimensionality (N, 90)
Output: generated time series $\mathbf{x}_{res}$
  calculate $\bar{\mathbf{x}}_p$, $\bar{\mathbf{x}}_n$, $\tilde{\mathbf{x}}_p$, $\tilde{\mathbf{x}}_n$, $\hat{\mathbf{x}}$
  set $\mathbf{x}_{res}$ with the normalized 0 value
  for $i$ in range(90) do
    $p_{spike}$ = random(0, 1)
    if $p_{spike} < \hat{\mathbf{x}}_i$ then
      $p_{positive}$ = random(0,1)
      if $p_{positive} \geq 0.5$ then
        $s$ = random_normal($\bar{\mathbf{x}}_{p,i}$, $\tilde{\mathbf{x}}_{p,i}$)
      else
        $s$ = random_normal($\bar{\mathbf{x}}_{n,i}$, $\tilde{\mathbf{x}}_{n,i}$)
      end if
      $\mathbf{x}_{res,i} = s$
    end if
  end for
  return $\mathbf{x}_{res}$
```

generator model architecture is equal to the standard VAE decoder architecture. The critic is similar to the VAE encoder, with the difference that in this case the output is a single value and after convolutions the number of fully connected layers is increased: this increases the capacity of the critic, allowing for better predictions and training convergence.
Table 5.6 shows the architecture of the WGAN-GP critic, different from the VAE encoder (Table 5.4) only after the last convolutional layer: while the encoder needs to generate a location on the latent space from which to sample, the critic model is required to output a measure of how the input is realistic. Another thing to note is that the WGAN-GP critic has a bigger capacity when compared to the WGAN generator (Table 5.7), suggested from the authors [Gulrajani et al., 2017] in order to improve the training of the combined model. Also, Batch Normalization is not used in the critic, as it is discouraged by the authors ([Gulrajani et al., 2017]).

| layer | layer type | layer parameters | output shape |
|---|---|---|---|
| 0 | Input | | (90,) |
| 0 | Conv block | batchnorm | (45, 32) |
| 1 | Conv block | batchnorm | (23, 32) |
| 2 | Conv block | batchnorm | (12, 32) |
| 3<br>4<br>5 | Conv1D<br>Activation(LeakyReLU)<br>Flatten | batchnorm | (384,) |
| 6<br>7 | Dense<br>Activation(Tanh) | neurons:128, batchnorm | (128,) |
| 8:z_mean<br>8:z_log_var | Dense<br>Dense | neurons:2<br>neurons:2 | (2,)<br>(2,) |

Table 5.4: VAE encoder architecture.

| layer | layer type | layer parameters | output shape |
|---|---|---|---|
| 0 | Input | | (2,) |
| 0<br>1 | Dense<br>Activation(LeakyReLU) | neurons:15, batchnorm | (15,) |
| 2 | Deconv block | batchnorm | (30, 32) |
| 3 | Deconv block | batchnorm | (60, 32) |
| 4 | Deconv block | batchnorm | (120, 32) |
| 5<br>6 | Conv1D<br>Activation(LeakyReLU) | batchnorm, kernel_size:1 | (120,) |
| 7<br>8 | Dense<br>Activation(Tanh) | neurons:90 | (90,) |

Table 5.5: VAE decoder architecture.

**WGAN-GP with packing**: this model is equal to the WGAN-GP one, with the only difference being in the dimensionality of the inputs of the critic. By applying packing, the dimensionality of the inputs increases by the packing degree applied. In our case, we opted for adding a single sample to the input, resulting in a packing degree of 2 and an input dimensionality of $(90, 2)$. The choice of the packing degree has been made in accordance with the experimental results of [Lin et al., 2017], in which the authors proved it to bring the biggest increase in generation quality along with minimal increase in complexity.

**VAE with l.s.m.**: as this model is a combination of an Improved WGAN and

| layer | layer type | layer parameters | output shape |
|---|---|---|---|
| 0 | Input | | (90,) |
| 0 | Conv block | | (45, 32) |
| 1 | Conv block | | (23, 32) |
| 2 | Conv block | | (12, 32) |
| 3<br>4<br>5 | Conv1D<br>Activation(LeakyReLU)<br>Flatten | | (384,) |
| 6<br>7 | Dense<br>Activation(LeakyReLU) | neurons:50 | (50,) |
| 8<br>9 | Dense<br>Activation(LeakyReLU) | neurons:15 | (15,) |
| 10 | Dense | neurons:1 | (1,) |

Table 5.6: WGAN-GP critic architecture.

| layer | layer type | layer parameters | output shape |
|---|---|---|---|
| 0 | Input | | (2,) |
| 0<br>1 | Dense<br>Activation(LeakyReLU) | neurons:15, batchnorm | (15,) |
| 2 | Deconv block | batchnorm | (30, 32) |
| 3 | Deconv block | batchnorm | (60, 32) |
| 4 | Deconv block | batchnorm | (120, 32) |
| 5<br>6 | Conv1D<br>Activation(LeakyReLU) | batchnorm, kernel_size:1 | (120,) |
| 7<br>8 | Dense<br>Activation(Tanh) | neurons:90 | (90,) |

Table 5.7: WGAN-GP generator architecture.

a VAE, the resulting architecture is just a combination of an Improved WGAN critic and the abovementioned VAE model. For this reason, refer to Table 5.4 and 5.5 for details regarding the VAE model, and Table 5.6 for details regarding the Improved WGAN critic model.

### 5.2.4 Models hyperparameters

Having defined all the architectures for the models used in this work, we can define the hyperparameters used in the training process. In order to eliminate bias for one model or the other, we decided to keep the values that appear in multiple models equal throughout all the experiments.

| **Parameter** | **value** |
|---|---|
| Batch size | 64 |
| Epochs | 1'000'000 |
| Generator iterations (WGAN-GP models) | 1 |
| Critic iterations (WGAN-GP models) | 5 |
| Generator/VAE lr | 0.001 |
| Critic lr (WGAN-GP models) | 0.001 |
| Gradient penalty weight (WGAN-GP models) | 10 |
| $\gamma$ (VAE with l.s.m.) | 0.5 |
| Lr schedule | step decay |
| Lr decay factor | 0.5 |
| Lr decay steps | 250'000 |
| Optimizer | Adam |
| Adam $\beta_1$ (WGAN-GP models) | 0 |
| Adam $\beta_1$ (VAE) | 0.9 |
| Adam $\beta_2$ (WGAN-GP models) | 0.9 |
| Adam $\beta_2$ (VAE) | 0.999 |

Table 5.8: Common hyperparameters used during training for every generative model.

Table 5.8 defines all the hyperparameters chosen during the experimental process: (WGAN-GP models) means that it's common to all the models linked to a GAN training process (WGAN-GP, WGAN-GP with packing, VAE with l.s.m.), while VAE with l.s.m. and VAE refer to a hyperparameter used only in the indicated model. The common optimizer used for all models is the Adam optimizer [Kingma and Ba, 2014].

## 5.3 Evaluation framework

In this section, parameters regarding the comparison models used to score the performance of the generative models presented in this work are defined (Subsection 3.3.1), along with details regarding the qualitative evaluation of the generated datasets using time series bitmaps comparison (Subsection 3.3.2).

One important detail regarding the evaluation procedure is that given time constraints on our work, each generative model is trained once: this results in a single source of generated data for each approach proposed, lowering the statistical significance of the overall results. To balance this phenomenon, we have increased the total amount of iterations (Table 5.8), resulting in generative models that had more time to learn the data distribution.

### 5.3.1 Quantitative evaluation

When evaluating generative models using our comparison framework, two datasets are needed: a real and a generated dataset: for the real dataset, a held out 30% of the entire data is taken, and for the generated dataset an equal amount of samples from last saved version of the generative model are selected, being with the model trained for the total amount of epochs. This combined dataset is then split into a 70%-30% fashion to allow both training and testing of the models.

**Classification task**: defined as the performance of a classifier in distinguishing real from fake data, this task is accomplished using two separate types of models: a Neural Network classifier and a Support Vector Classifier (SVC), the latter a particular type of a Support Vector Machine (SVM) used for this type of tasks. Each model used in the qualitative evaluation is trained 10 times on 10 different splits of the combined data. The performance of each trained classifier is determined by two different scores, *accuracy* and *F1 score*.

The Neural Network classifier architecture is closely related to the architectures of the other generative models employing Neural Networks used in this work: the main feature extraction method is a series of convolutional blocks ending in a one-value output layer.

| **layer** | **layer type** | **layer parameters** | **output shape** |
|---|---|---|---|
| 0 | Input | | (90,) |
| 0 | Conv block | | (45, 32) |
| 1 | Conv block | | (23, 32) |
| 2 | Conv block | | (12, 32) |
| 3 | Conv1D | | |
| 4 | Activation(LeakyReLU) | | |
| 5 | Flatten | | (384,) |
| 6 | Dense | neurons:1 | |
| 7 | Activation(Sigmoid) | | (1,) |

Table 5.9: Neural Network classifier architecture.

Table 5.9 shows the architecture of the Neural Network Classifier: the differing part from the above architectures is the sigmoid activation function, that allows two-class discrimination of the input time series. Again, to be able to pass the features extracted by the convolutional layers to the final fully connected layer, we flatten the output of the last convolutional block. The training process for this model uses the Adam optimizer [Kingma and Ba, 2014] with parameters suggested from the original authors (lr=0.001, $\beta_1$=0.9, $\beta_2$=0.999). To obtain accurate results from the classifier, an Early Stopping learning rate schedule has been used, allowing the model to be trained until scores calculated on a validation dataset don't shows signs of improvements for 3 consecutive epochs. For this schedule, an held out validation dataset using 20% of the total training samples is used.

The Support Vector Classifier uses a standard implementation of the sklearn library, with default hyperparameters. In our implementation the RBF kernel has been used.

### 5.3.2 Qualitative evaluation

With respect to qualitative evaluation, we approach the task of comparing different generative models by looking at the bitmaps generated from their produced datasets. The parameters set for this task are the following:

- *Alphabet size $n$*: chosen to be equal to 4, as an initial analysis of the data revealed that the values in the time series take few ranges only, and don't cover the entire [-1, 1] range. Having an alphabet of 4 letters gives enough flexibility to distinguish different datasets, but it's not too broad, as 9 or 16 would be. The latter choice would result in a much sparser bitmap, as there will be close to no values appearing in the intermediate ranges of values.

- *Sliding window length $L$*: for the experiments done, a length of 4 has been chosen, as longer windows would generate a bitmap with too many pixels reducing the possibilities of obtaining similar images to analyze, and shorter windows would fail to capture any similarities.

Regarding the qualitative evaluation procedure, two comparison experiments are defined, both using the abovementioned bitmaps:

- *Mean bitmap comparison*: by averaging the bitmaps generated from a particular dataset, a mean bitmap can be calculated: by comparing such images coming from different datasets, we can visually determine which generated dataset has the closest properties to the original one.

- *Std bitmap comparison*: along with mean bitmap analysis, the bitmap indicating the std for each pixel can be generated, allowing to determine which changes in the bitmaps composing a dataset are more prominent.

This evaluation procedure could allow users without domain knowledge to easily determine if two datasets have big differences in their composition, and along with quantitative results, a more informed analysis can be made.

# Chapter 6

# Experimental results

In this chapter, we present the results obtained in our experimentation, for both the quantitative and qualitative evaluations procedures. Section 6.1 gives our initial thoughts on the generated data, with insights on the results for the different models. Section 6.2 reports the results obtained in the quantitative evaluation process, and Section 6.3 describes the visual comparisons done between the real and generated datasets.

## 6.1 Preliminary evaluations

After all the generative models have been trained for the maximum amount of epochs, we can take an initial look at the generated samples. This data can then be compared with the samples in the real dataset to gather some initial insights.

**Random samples comparison**: an initial analysis can be done by looking at samples coming from the test dataset, compared to the samples generated by each of the generative models proposed. This gives a rough visualization of the performance of each generative model is, we plot 8 randomly generated samples for each dataset.
Figure 6.1 shows some random samples from the test set: we can see how the sporadic spikes are followed by flat regions, and finding a structure in this type of data is hard for users not having direct expertise on this domain. The spikes seem to almost always lie in the extreme regions of the allowed values, with sporadic smaller spikes in some cases.
Figure 6.2 shows some examples of time series generated using the handcrafted approach. It can be easily seen how the flatness is maintained, along with the general magnitude of the spikes. We can also see that the spikes are much more sporadic, when compared to the real samples in Figure 6.1.

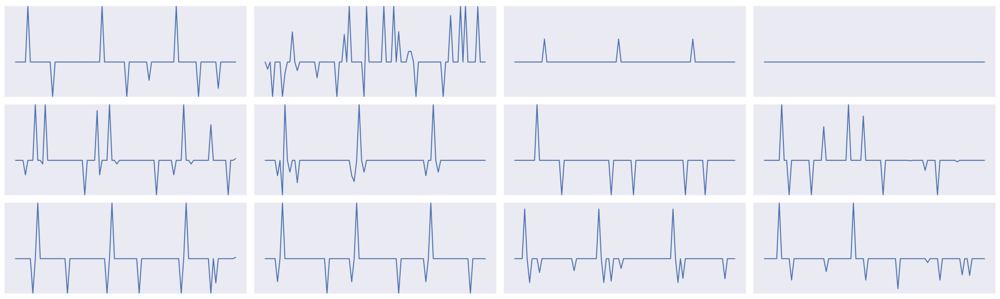
Figure 6.1: Real samples from the test set.

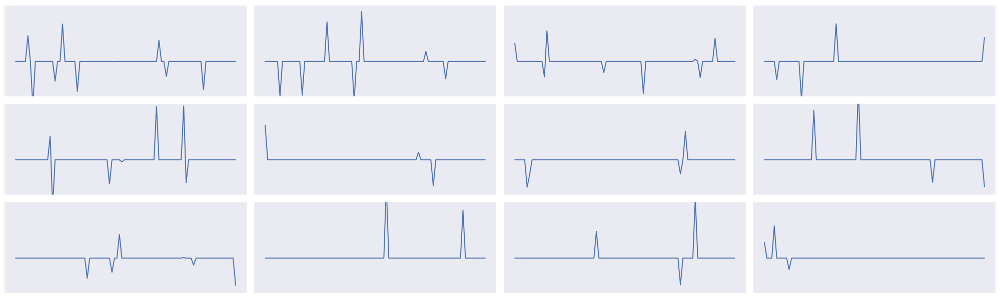
Figure 6.2: Handcrafted samples.

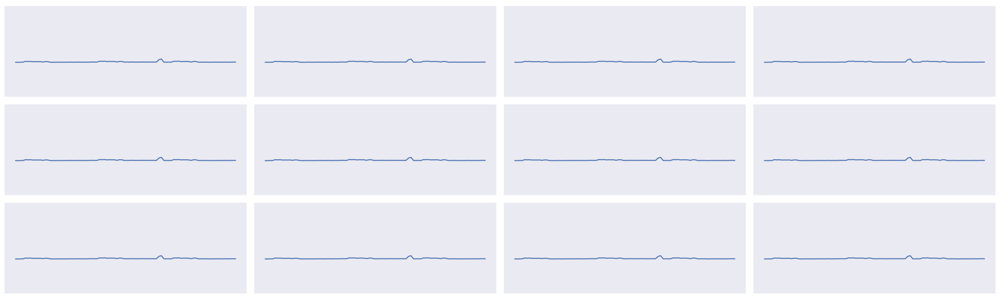
Figure 6.3: VAE samples.

Figure 6.3 shows some samples generated from the Variational Autoencoder model: it can be immediately seen how the model collapsed on generating a time series with values around the mode of the data, occurring for every training experiment done with this model. This issue arises from the fact that the model is trying to minimize a bad loss for this type of problem: given that the objective of this work is to build a generative approach that is as much ”data agnostic” as possible, refining the loss metric to fix this problem would drive the model to be too data dependent. When other standard similarity metrics are used (MSE, MAE, Poisson, Cosine Similarity etc.), the final result is the same. By giving a high penalty to any mismatch between the position of the spikes, the VAE model ends up at generating a value that is the closest to the mode for each time step, reducing the total amount of penalization.

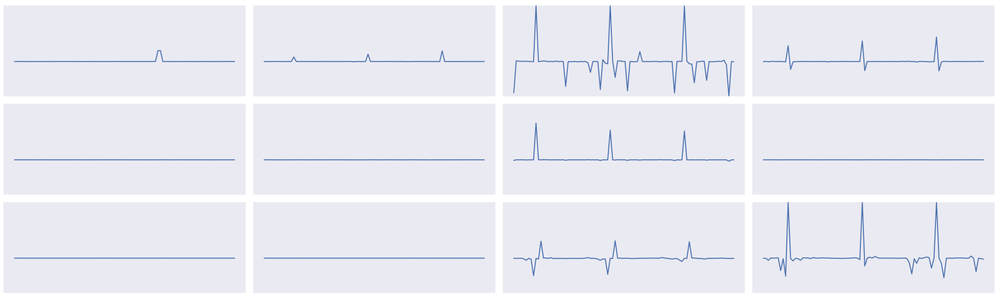
Figure 6.4: WGAN-GP samples.

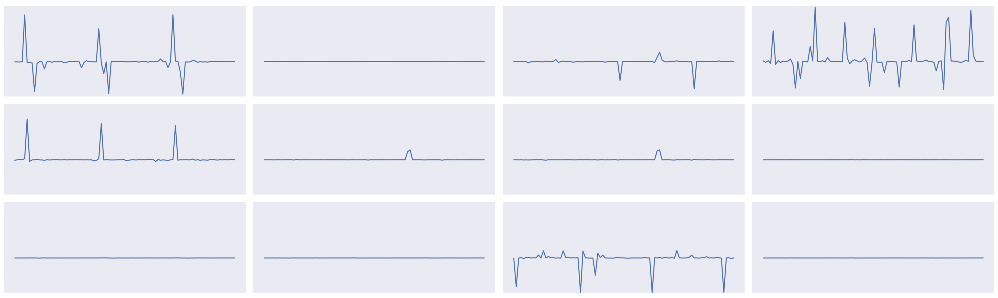
Figure 6.5: WGAN-GP (packed inputs) samples.

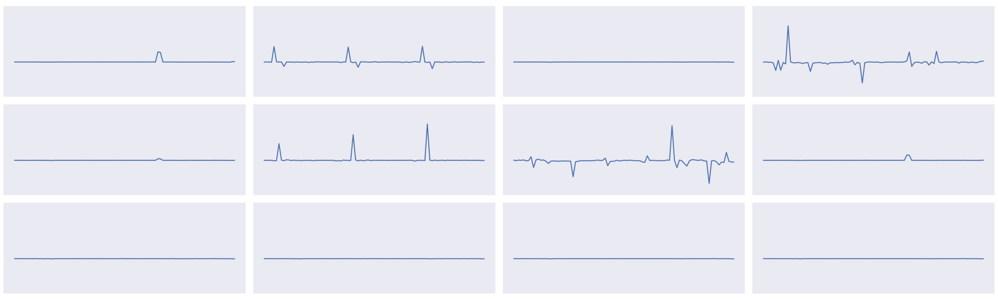
Figure 6.6: VAE with learned similarity metric samples.

Figures 6.4, 6.5 and 6.6 show some random samples obtained respectively using the WGAN-GP, WGAN-GP with packed inputs and VAE with learned similarity metric generative models. We can see how the models appear to have correctly learned how to produce realistic looking samples, although it can be seen how the amount of flat transactions is higher than usual, suggesting some mode collapse, and that the flat areas are not as good as the ones from the original dataset.

The first problem interestingly appears also in the Improved WGAN model with packed inputs, specifically chosen for its ability to reduce mode collapse and diversify the input: this fact could suggest that packing the inputs to the critic model doesn't help the generation. The VAE with l.s.m. model seems to also produce many flat transactions, but the general quality of the non-flat samples seems a bit inferior compared to the other two GAN models.

Flat areas not correctly generated could pose a problem when the data is used for the quantitative evaluation, as non-flat areas of the data signify a transaction: for this reason, a classifier could focus on checking for this small details in order to correctly classify the inputs. Another problem that could arise is when a model used for real-world applications is used, as it could see many low-valued transactions instead of flat regions, resulting in wrong behaviors.

With this first preliminary inspection, we can conclude that the Variational Autoencoder model wasn't able to correctly converge, and the GAN models seem to produce decent results with some issues in producing flat regions. The Variational Autoencoder with learned similarity metric model seems to perform similarly to the other two GAN models, although the spikes don't seem as pronounced.

**Post-processing**: after some initial experimentation, and the results shown above, we have noted that the fact that trained generative models are not able to generate perfectly flat areas can ease the work of a classifier that is required to distinguish real from generated data: by just looking at flat regions, the classifier can more easily determine what is the original dataset, thus removing the need of checking more important semantic features of the data. For this reason, we decided to add an additional post-processing step, in which the range of values that are at most $\lambda$ less or more than the normalized zero, is substituted with the zeroed value. This can help in determining which features the classifier determines as most important.

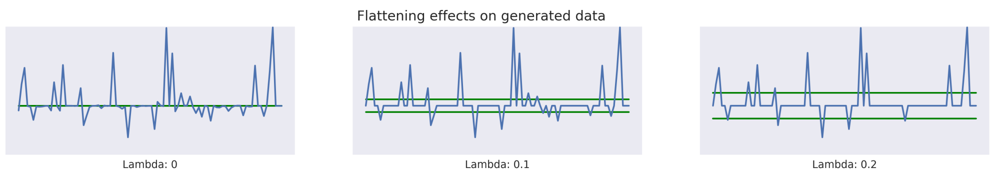

Figure 6.7: Effects of different $\lambda$ values on a generated sample.

Figure 6.7 shows the effects of using the flattening technique proposed with varying values of the flattening range $\lambda$, 0, 0.1 and 0.2 respectively: while the main features of the sample remain the same, we can see how the overall quality is im-

proved with increased lambda, whose range is indicated by the two green lines. A classifier tasked to determine which dataset produced the above samples now cannot rely on minor generation issues as before, and it's required to learn better semantic features of the input.
Given those observations, we will include different values for $\lambda$ in the following experiments, namely from 0.0 to 0.2 with increases of 0.05 for each step. This technique is applied to all the generative models apart from the handcrafted dataset, as it's not affected by the described issues.

## 6.2 Quantitative results

As noted in chapter 3, we define a classification task for which two additional models would be trained to distinguish a real dataset from a generated one. This task would output 4 different scores for each dataset, 2 scores for each of the two evaluating models used. As noted in Section 6.1, the reported scores will be presented with varying values for the parameter $\lambda$, in our experiments set to 0.0, 0.05, 0.1, 0.15 or 0.2.

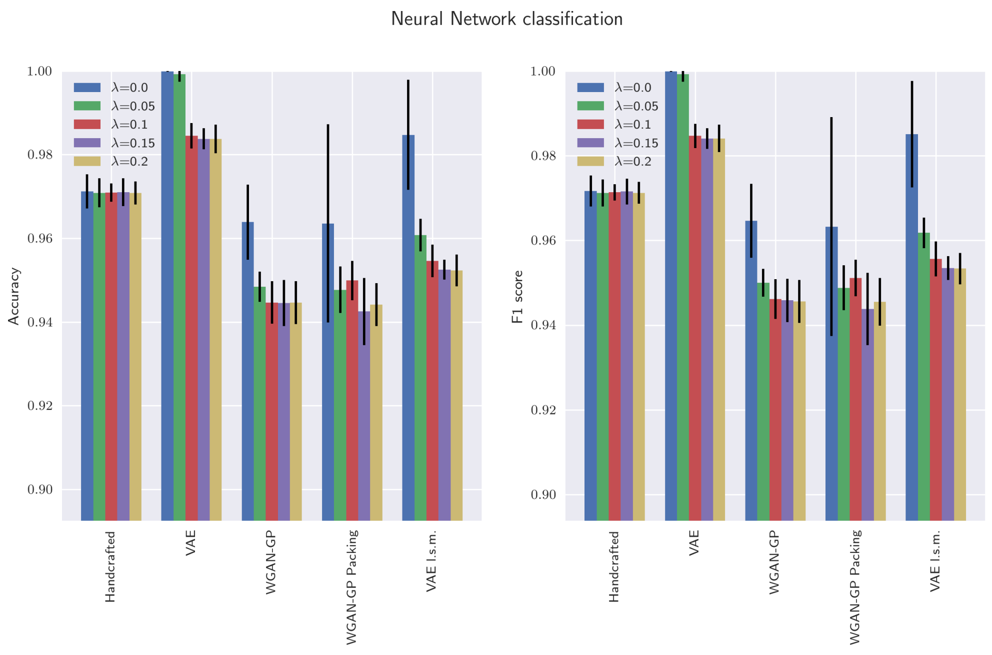

Figure 6.8: Neural Network classification scores using variable values for $\lambda$.

Figure 6.8 shows the classification scores for the Neural Network model trained on each dataset. Each bar indicates the mean score over the 10 runs, and the error line on each bar indicates the standard deviation calculated from the 10 runs. Each color corresponds to a different value for the $\lambda$ parameter: when $\lambda$ is set to 0.0 (blue), meaning no post-processing, it can be immediately seen how the model achieves perfect accuracy on the data generated by the VAE, due to the model collapsing on generating only one sample.

Again, when $\lambda$ is set to 0.0 (blue), we can notice how the worse performing model is the VAE with learned similarity metric: as noted in Section 6.1, this model seems to produce lower quality results, allowing the classifier to easily recognize the samples generated with this method with an accuracy of around 98.5%. Better results are obtained by the handcrafted generation process ($\sim$ 97%), that also obtains lower variance in the scores. Slightly lower, thus better, scores are obtained by the remaining two generative models, the two variants of the WGAN-GP approach. Although very similar in the mean score ($\sim$ 96.5%) the variant using packed inputs has a much higher variance in scores.

When $\lambda$ is increased and the datasets are post-processed, we can see a noticeable decrease in the scores obtained by the classifier, meaning higher difficulty in solving the task: every generative model achieves better results, apart from the handcrafted method, due to the samples not suffering from flatness problems. This is especially noticeable with the VAE with l.s.m., where the decreases in Accuracy and F1 score are of around 3%. Along with lower scores, also a reduction in the variance of the various runs can be noticed. This factor allows enforcing the fact that our post-processing leads to improved results. Some benefits of dataset post-processing are also seen in the collapsed VAE model, in this case because the data becomes almost perfectly flat and the classifier has less information to use. Between all the generative models using the WGAN-GP architecture, the Variational Autoencoder with l.s.m. model seem to achieve slightly worse results, although of only 1%.

This evaluation task suggests that generative models employing GANs seem to produce viable results, and the inclusion of a post-processing step does indeed help the data in becoming more similar to the real one. Our proposed model seem to again achieve comparable results to the other two GAN models, meaning that it was in fact able to avoid mode collapse and learn a good data representation. Stronger conclusions can be made if similar results appear in the SVM classification task.

Figure 6.9 shows the results of the classification task when the Support Vector Classifier is used: first, the scores are much lower compared to the previous classifier, indicating more difficulty in separating real from generated data. Strangely, the classifier is unable to obtain higher scores when the VAE generated dataset is used: with an accuracy of around 90% similar to the one obtained with hand-

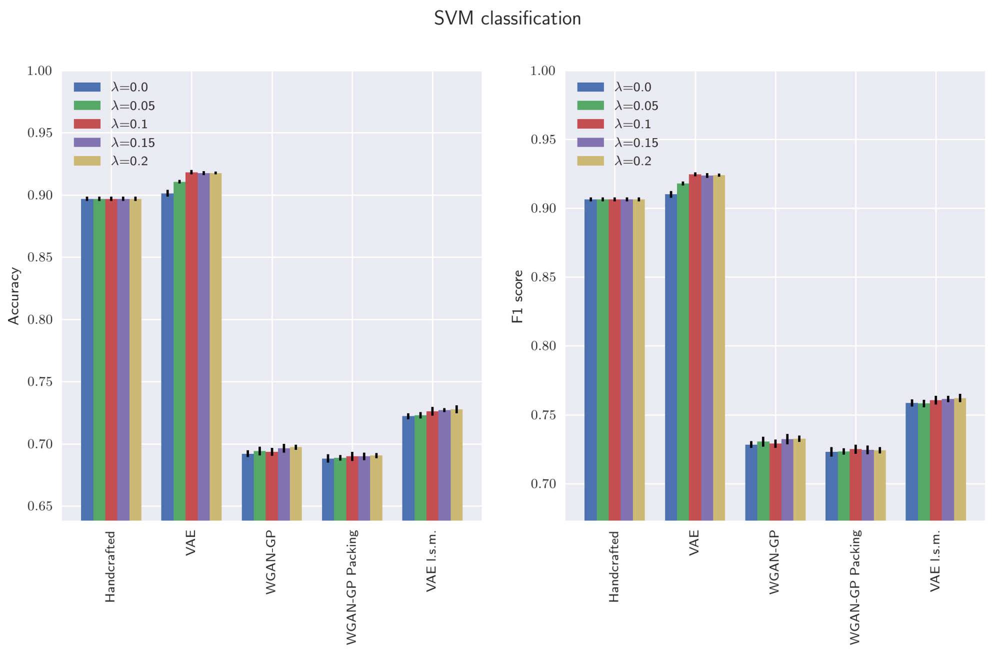

Figure 6.9: SVM classification scores using variable values for $\lambda$.

crafted data, this issue could derive from the fact that usually Support Vector Machines require more fine tuning than traditional Neural Networks, resulting in sub-optimal behaviors when trained on such a particular type of data. Similarly to previous results, the models employing the WGAN-GP architecture obtain similar scores, with the Variational Autoencoder with l.s.m. model performing slightly worse. Finally, the inclusion of post-processing steps seem to not change the results in any way, indicating that this model is not analyzing the time series given as an input in traditional ways.

Overall, the SVM results show that without fine tuning the hyperparameters of the model the performance is much lower than compared with Neural Network, and it's not able to distinguish even easy cases like the collapsed Variational Autoencoder. This problem in particular suggests that this results need to be taken with caution.

Tables 6.1 and 6.2 show the classification scores for all the datasets, with varying values of $\lambda$. For each dataset, the Accuracy and F1 scores for both the Neural Network classifier and the Support Vector Classifier are presented. Given the above plots, we decide to discuss only the Neural Network scores.
The best results obtained by the Neural Network classifier (lowest accuracy and F1 score) are achieved with the Improved Wasserstein GAN with packed inputs,

| **$\lambda$** | **Dataset** | **NN Accuracy** | **NN F1 score** |
|---|---|---|---|
| 0.0 | Handcrafted | 0.971 | 0.972 |
| | VAE | 1.000 | 1.000 |
| | WGAN-GP | 0.964 | 0.965 |
| | WGAN-GP Packing | 0.964 | 0.963 |
| | VAE l.s.m. | 0.985 | 0.985 |
| 0.05 | Handcrafted | 0.971 | 0.971 |
| | VAE | 0.999 | 0.999 |
| | WGAN-GP | 0.948 | 0.95 |
| | WGAN-GP Packing | 0.948 | 0.949 |
| | VAE l.s.m. | 0.961 | 0.962 |
| 0.1 | Handcrafted | 0.971 | 0.971 |
| | VAE | 0.984 | 0.985 |
| | WGAN-GP | 0.945 | 0.946 |
| | WGAN-GP Packing | 0.95 | 0.951 |
| | VAE l.s.m. | 0.955 | 0.956 |
| 0.15 | Handcrafted | 0.971 | 0.972 |
| | VAE | 0.984 | 0.984 |
| | WGAN-GP | 0.945 | 0.946 |
| | WGAN-GP Packing | **0.942** | **0.944** |
| | VAE l.s.m. | 0.952 | 0.953 |
| 0.2 | Handcrafted | 0.971 | 0.971 |
| | VAE | 0.984 | 0.984 |
| | WGAN-GP | 0.945 | 0.946 |
| | WGAN-GP Packing | 0.944 | 0.945 |
| | VAE l.s.m. | 0.952 | 0.953 |

Table 6.1: Neural Network Accuracy and F1 scores for the classification task.

along with a value of $\lambda$ of 0.15. For this generative model in particular, changes in the value of $\lambda$ don't result in gradual improvements, but fluctuations not noticed with other models. This fact, along with the very high variance observed when $\lambda = 0.0$, suggests that the generative model could have not learned the data distribution as well as the others. Even if this model achieves the best results, the vanilla WGAN-GP model obtains very similar scores.

**Effect of model training**: Given the results in the above experiments, another aspect of interest would be the effect of model training on the overall performance of the models proposed in this work. For this reason, we run the same type of quantitative experiment as before, but in this case, using various checkpoints oc-

| **$\lambda$** | **Dataset** | **SVM Accuracy** | **SVM F1 score** |
|---|---|---|---|
| 0.0 | Handcrafted | 0.897 | 0.906 |
| | VAE | 0.901 | 0.91 |
| | WGAN-GP | 0.692 | 0.728 |
| | WGAN-GP Packing | **0.688** | **0.723** |
| | VAE l.s.m. | 0.722 | 0.759 |
| 0.05 | Handcrafted | 0.897 | 0.906 |
| | VAE | 0.91 | 0.918 |
| | WGAN-GP | 0.694 | 0.73 |
| | WGAN-GP Packing | 0.689 | **0.723** |
| | VAE l.s.m. | 0.723 | 0.758 |
| 0.1 | Handcrafted | 0.897 | 0.906 |
| | VAE | 0.918 | 0.925 |
| | WGAN-GP | 0.693 | 0.729 |
| | WGAN-GP Packing | 0.69 | 0.725 |
| | VAE l.s.m. | 0.726 | 0.761 |
| 0.15 | Handcrafted | 0.897 | 0.906 |
| | VAE | 0.918 | 0.924 |
| | WGAN-GP | 0.696 | 0.732 |
| | WGAN-GP Packing | 0.69 | 0.724 |
| | VAE l.s.m. | 0.727 | 0.761 |
| 0.2 | Handcrafted | 0.897 | 0.906 |
| | VAE | 0.918 | 0.924 |
| | WGAN-GP | 0.697 | 0.733 |
| | WGAN-GP Packing | 0.69 | 0.724 |
| | VAE l.s.m. | 0.728 | 0.762 |

Table 6.2: SVM Accuracy and F1 scores for the classification task.

curring during model training as comparison datasets. The chosen model is the Improved Wasserstein GAN with $\lambda = 0.15$, given the scores obtained in the previous experiments. The checkpoints have been chosen at $100'000$ iteration steps, from start to end of training. Both the classifier choices, the metrics used and the number of trials are kept the same.
Figure 6.10 shows the scores obtained for various checkpoints of the Improved Wasserstein GAN with packing model during training. We can see a general decreasing trend in the results, from around 0.96% to around 0.94% for both accuracy and F1 score. This trend seems to have a slight increase towards the last 300 thousand iterations, of around 0.5%. The value of $\lambda$ has a positive effect on the variances of the results, being small enough to enforce the descending aspect

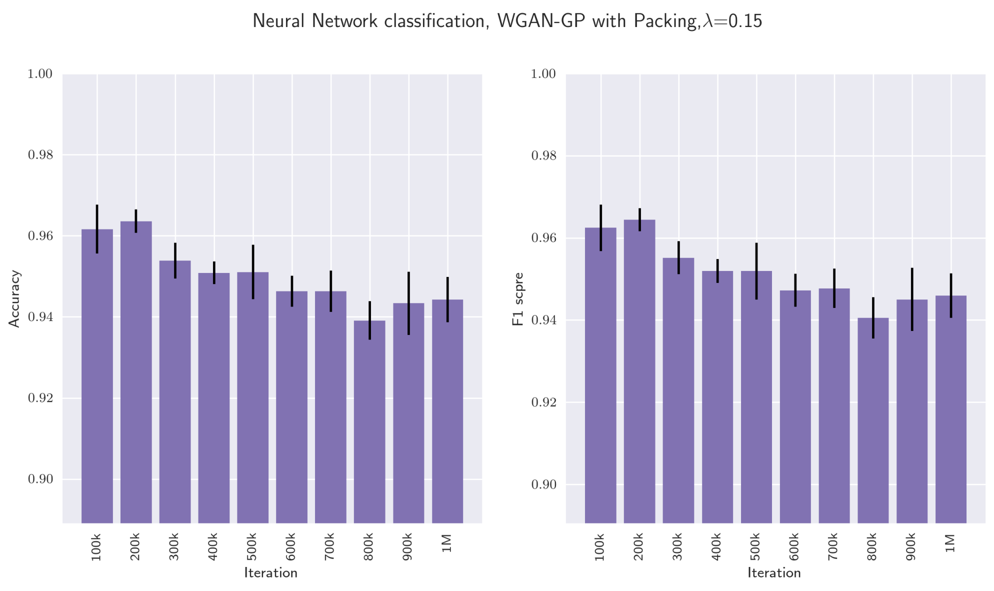

Figure 6.10: Neural Network classification scores variation during training of the WGAN-GP with packing model.

of the trend noticed.

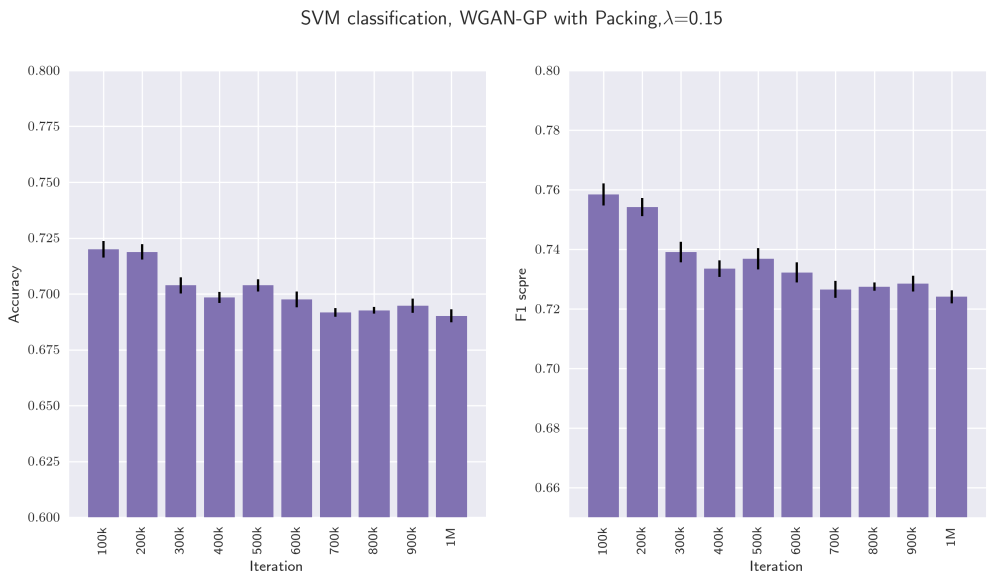

Figure 6.11: SVM classification scores variation during training of the WGAN-GP with packing model.

Figure 6.11 shows the results obtained when training the SVM model: we can still see a general decreasing trend, and as with the past experiments using the SVM model, very low variances. Although the F1 score has a different magnitude than the accuracy score, we can observe the same trend on both.

This small experiment seems to suggest that model training, in fact, helps in getting better results: the $\lambda$ parameter seems to help to obtain small variances during the whole process, and the results align with the ones obtained in the previous experiments.

## 6.3 Qualitative results

In this section, a qualitative comparison is done by visually inspecting the bitmaps generated from the time series comprising the various datasets.

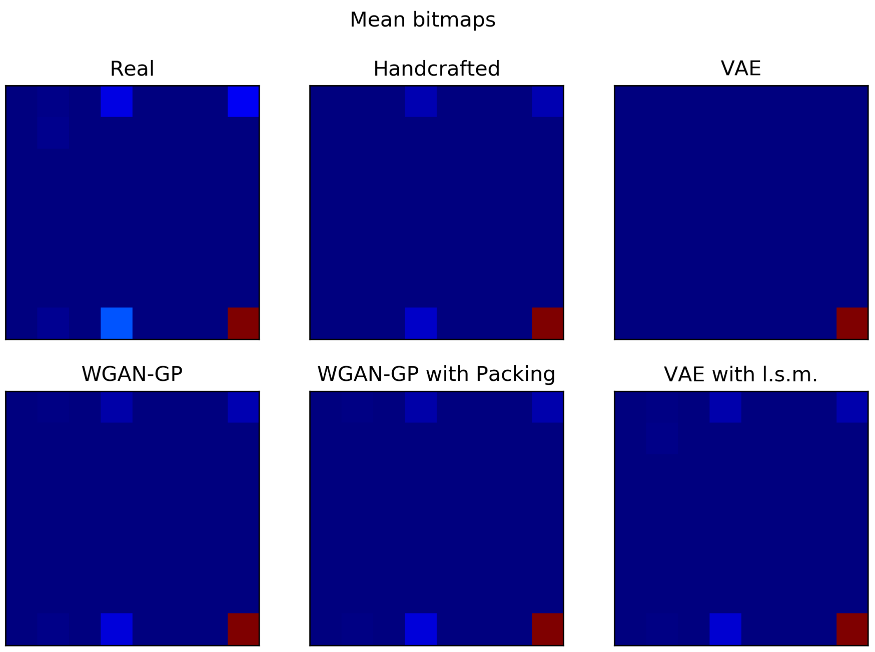

Figure 6.12: Mean bitmap for each dataset used. Colors closer to red indicate higher frequency of that sub-pattern.

Figure 6.12 shows the mean bitmaps for each dataset used in our work. Each pixel in the bitmap indicates a particular pattern, and its color indicates the frequency of such pattern. Dark blue indicates no frequency, while dark red indicates very high frequency.

By looking at the mean bitmap of the real dataset we can see how only 7 out of the 64 pixels have a value greater than zero, meaning that there are only 7 different sequences of values (words) appearing when the $[-1, 1]$ domain is divided into an alphabet of four letters. Although not being very informative, we can see how little the diversification between the pattern appearing in the time series is.

For the bitmap relative to the handcrafted data, we can notice some close resemblance, although the pixel colors are more faint, with the 3 leftmost pixels in the same position of the real dataset bitmap indicating no patterns of that type.

When looking at the VAE bitmap, even without any quantitative evaluations, we could conclude that this dataset greatly differs from the original, due to the model collapsing on generating one time series. Being able to easily distinguish different datasets without any domain knowledge allows this method to be very viable in this type of comparisons.

The lower bitmaps refer to the models proposed in this work: each one seems to closely relate to the comparison bitmap, with every feature present under the form of a lighted pixel, although sometimes not as bright.

Using only this first inspection, we are able to easily separate the VAE model from the other ones, but the information is not sufficient to give conclusions on which model is better when is compared to the others.

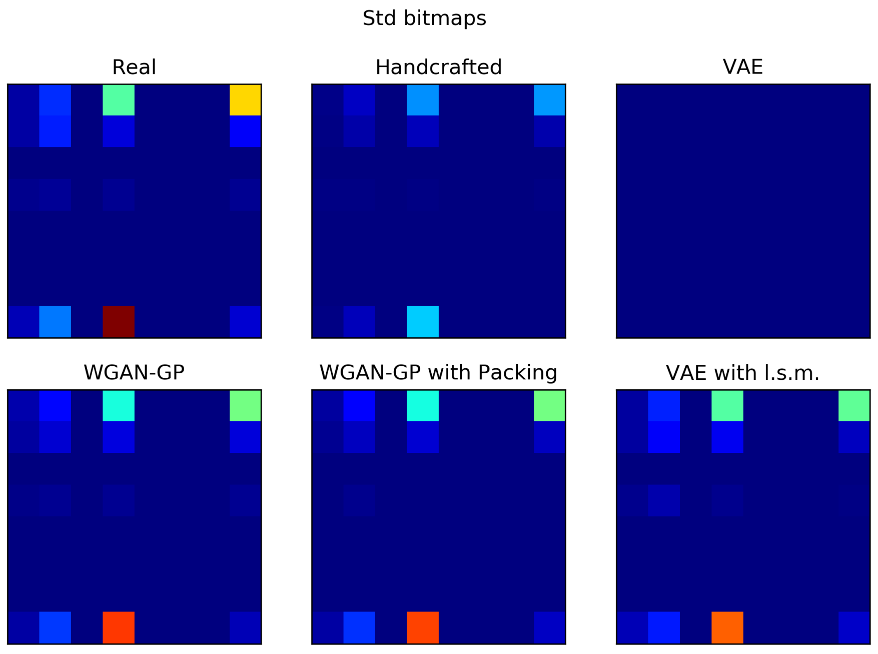

Figure 6.13: Std bitmap for each dataset used. Colors closer to red indicate higher frequency of that sub-pattern.

Figure 6.13 shows the std bitmaps for each dataset. Each pixel is now indicating the standard deviation of the pixels in the bitmaps generated a dataset's time series.

When looking at the bitmap of the real data, we can see how the amount of available information that can be used is increased. Now the variety of values for the pixels is greater, meaning that visual comparisons between this bitmap and the others should be easier.

The bitmap for the handcrafted data now exhibits some different visual features, when compared to the real data bitmap: even though the location of the lighted pixels corresponds, their colors are quite different, especially for the two in the top row. This tells us that the handcrafted dataset has much more diversification for some of its sub-patterns, more than the one in the real dataset. This visual discrepancy is easy to notice and allows a quick differentiation of the two datasets.

Similarly to the mean bitmap, the std bitmap of the VAE dataset is completely different from the real one: now, as the model is collapsed, we have no differentiation, resulting in no lighted pixels. Again, differentiation is immediate in this case.

Regarding the lower three bitmaps, we can now see how they are very similar to the bitmap generated from real data: each pixel is correctly colored, with minor color changes regarding the two brightest pixels on the top row. As with the quantitative results, we can define the most similar model to be the WGAN-GP one but closely followed by the other three.

## 6.4 Insights

Given the results obtained by our work, we can give some insights covering the dataset proposed and the generative models used.

Regarding the dataset used, we identify one main issue: we have noticed how the lack of information in the data used translated in difficulty for our understanding of the results and trouble for both generative models and quantitative evaluation models when required to understand the features of the dataset. We think that more contextual information to pair on every timestep would greatly improve the quality of the results, and ease the work of non-experts.

Another problem, that if solved could improve the performance of the proposed models, is the scarcity of samples. By splitting the initial dataset into samples we removed the information that allows the generative model to understand which time period is presented: each input indicates three months worth of transactions, but no indication is given on which months are covered. With more samples available, we could split the dataset on always the same period, easing the learning process for the generative model.

Regarding the generative models used, we can indicate that:
Variational Autoencoders, and more generally, generative models that optimize a similarity metric defined a priori, are not able to learn complex distribution like the one where this dataset originates from if no extensive tuning is added.

Convolutional methods can be applied to time series data, even if it's not their main application domain. With a simple visual inspection, we have seen how generative models employing convolutional approaches are able to produce realistic-looking data, even if not perfectly (flat regions).

Our Variational Autoencoder with learned similarity metric is able to avoid mode collapse. Although similar approaches have been already proposed in the literature, this simple application of Variational Autoencoders and Improved Wasserstein GANs was able to obtain results comparable to the other proposed models, supported by a simple and intuitive formulation.

The proposed generative models, that use the Improved Wasserstein GAN formulation, are able to obtain good results, although post-processing of the data is needed to eliminate bad flat regions: without post-processing, a classifier can exploit features that the models have trouble to represent.

# Chapter 7

# Conclusions

In this chapter, we give some conclusions based on the results obtained by our experimentations. In Section 7.1 we give a summary of our work, whereas in Section 7.2 we answer the initial research questions defined in Section 1.2.

## 7.1 Summary

In this work we analyzed the feasibility of approaching the task of generating spiking time series found in a common dataset used for research in the banking domain, using Generative Adversarial Networks.

We proposed the use of an improved theorization of the standard GAN, namely the Improved Wasserstein GAN, used as a base model and as an alternative formulation in which inputs for the critic are augmented in order to mitigate mode collapse. We also proposed an improved formulation of the traditional Variational Autoencoder, that allows the model to automatically learn an appropriate similarity metric thanks to an Improved Wasserstein GAN critic.

Given the novelty of our application, we use a baseline generative model to make comparisons, namely a Variational Autoencoder. Also, to simulate the task of an expert being required to generate this type of data, we add another model that uses statistics of the input dataset in order to generate new time series.

Given the absence of classes in the dataset used, combined with the peculiarity of the time series, we presented an evaluation framework that allows to obtain both quantitative and qualitative insights on the performance of our proposed models, resulting in an end-to-end approach that enables comparisons. We approach the task of quantitative evaluation using classifiers trained to distinguish real from generated data, and the task of qualitative evaluation by analyzing bitmaps generated from datasets created after each generative model was trained.

The results obtained suggest that the proposed models, namely the Improved

WGAN with and without packing, achieve better results than the comparison models employed: contrarily to the Variational Autoencoder used, this model didn't collapse during training. The new approach proposed, namely the Variational Autoencoder with learned similarity metric, is able to achieve similar results and is able to also avoid mode collapse like the others. Our qualitative evaluation shows how the visualization proposed is, in fact, able to allow quick comparisons even by non-experts, without the need of supporting quantitative results.

## 7.2 Research questions

In this section, we use the experimental results obtained in Chapter 6 to answer each research question defined in our work.

### 7.2.1 RQ1: How can we generate real-valued spiking time series coming from the banking domain?

Our work has shown how Generative Adversarial Network models are able to learn a latent space structure that allows producing real-valued time series patterns, similar to the input dataset used. One main problem is that the generated samples lack quality on flat regions, meaning that the critic model is still unable to correctly understand all the semantics of the data: this phenomenon could come from the low amount of information available to the models during training, meaning that increasing it for each time step could benefit the training process. Other architecture could also be tested, but we think that it shouldn't be the main focus.

### 7.2.2 RQ2: How can we evaluate the performance of models generating real-valued spiking data?

The proposed evaluation framework is a step in the right direction as it allows to tackle the problem of evaluating different generative models when specific evaluation metrics are absent. The fact that this data is so peculiar translates in difficulty in defining other quantitative evaluation metrics, needed to give more robust results. Overcoming the lack of typical qualitative evaluation due to the peculiar data at hand with the introduction of time series bitmaps allowed quick and easy comparisons. On the other hand, this approach suffers from the fact that bitmaps discard the locations of the sub-patterns found, resulting in less reliable results. Overall, if a trend is seen in both evaluation procedures, we can give sound conclusions. Finally, the fact that real samples are flat in most time steps, resulted in having to artificially process the generated datasets in order to obtain comparable performances. To avoid introducing expert knowledge in the problem, a better comparison task would be preferred.

### 7.2.3 RQ3: How can we eliminate the need for defining a similarity metric for generative models that require it?

Our experiments show how our proposed Variational Autoencoder with learned similarity metric is able to remove the collapsing problem from the comparison traditional Variational Autoencoder. The model also is able to achieve results similar to our Improved Wasserstein GAN models, indicating a good training process. While our evaluation framework is still not able to give definitive results on the ability of our model to effectively learn any input distribution, we are positive that the results are good enough to define it a viable alternative to generative models using pre-defined similarity metrics.

# Chapter 8

# Future work

In this chapter, future work following ours is proposed.

Given the results obtained by our work, we identify four future main research paths: the first is in developing a better evaluation framework, in order to improve the quality of the future applications on this task. The second is directed towards studying the effective performance of our proposed VAE with learned similarity metric. Third, the application of real-world models to our data could give interesting insights. Finally, working with labeled data could translate in the ability to generate samples conditioned on some inputs.

As indicated in Chapter 7, the proposed evaluation framework is still not able to produce strong results, and for this reason, more research could be driven towards mitigating this issue. By developing better quantitative evaluation tasks, such as forecasting or sample classification, more scores could be available for consideration. On the topic of qualitative evaluation, bitmap generation seemed a suitable choice, although better representations that also keep the locality of the features would be preferred.

More work can be done towards assessing the effective capabilities of our proposed VAE with l.s.m. model: as we haven't defined a specific architecture but a general model, it will be possible to apply it in every field in which a Variational Autoencoder can be applied. This then allows using traditional evaluation methods to critically compare our model to other generative alternatives and effectively understanding how valid is this approach in other scenarios. The fact that is very easy to convert a traditional Variational Autoencoder into our formulation means that fields in which such models didn't work before could be tested, reducing the need of developing yet another model. As this model positions itself between traditional Variational Autoencoders and Generative Adversarial Networks, more studies could be conducted towards understanding how viable and comparable is this method compared to one or the other.

The quantitative evaluation proposed in our framework is a simple binary classification task, but more involved methods could be applied: for example, predictive and forecasting models coming from real business application could be transferred to our dataset, allowing much stronger conclusions to be drawn. Also, models tasked with detecting anomalies or inconsistencies in the data could be used, in order to approach the comparison problem from another perspective.

An important task in the banking domain is fraud detection: by analyzing the patterns in the input data, a model is required to understand if the user is actively trying to steal money from the system or other users. By using a dataset annotated with this type of data for each sample, the generative models proposed could be transformed into conditional generative models, allowing the user to generate data with particular characteristics. In this case, being able to generate fraudulent and non-fraudulent samples, new comparison techniques could also be added to the evaluation framework.

# Bibliography

A. Antoniou, A. Storkey, and H. Edwards. Data Augmentation Generative Adversarial Networks. *ArXiv e-prints*, November 2017.

Martín Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 2017.

Petr Berka and Marta Sochorova. Berka, 1999 czech financial dataset, 1999. URL `https://sorry.vse.cz/~berka/challenge/pkdd1999/berka.htm`.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL `https://doi.org/10.1007/BF00994018`.

Ugur Demir and Gözde B. Ünal. Patch-based image inpainting with generative adversarial networks. *CoRR*, abs/1803.07422, 2018.

Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *CoRR*, abs/1802.04208, 2018.

C. Esteban, S. L. Hyland, and G. Rätsch. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. *ArXiv e-prints*, June 2017.

William Fedus, Ian Goodfellow, and Andrew Dai. Maskgan: Better text generation via filling in the ____. 2018. URL `https://openreview.net/pdf?id=ByOExmWAb`.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017.

Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the automatic anime characters creation with generative adversarial networks. *CoRR*, abs/1708.05509, 2017.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

Nitin Kumar, Venkata Nishanth Lolla, Eamonn J. Keogh, Stefano Lonardi, and Chotirat Ratanamahatana. Time-series bitmaps: a practical visualization tool for working with large time series databases. In *SDM*, 2005.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. *CoRR*, abs/1512.09300, 2015.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15(2):107–144, Oct 2007. ISSN 1573-756X. doi: 10.1007/s10618-007-0064-z. URL `https://doi.org/10.1007/s10618-007-0064-z`.

Zinan Lin, Ashish Khetan, Giulia C. Fanti, and Sewoong Oh. Pacgan: The power of two samples in generative adversarial networks. *CoRR*, abs/1712.04086, 2017. URL `http://arxiv.org/abs/1712.04086`.

Olof Mogren. C-RNN-GAN: continuous recurrent neural networks with adversarial training. *CoRR*, abs/1611.09904, 2016.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. How good is my gan? In *The European Conference on Computer Vision (ECCV)*, September 2018.

Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *CoRR*, abs/1609.02612, 2016.

Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *CoRR*, abs/1803.01541, 2018.