Generative AI for Synthetic Data

Esaias Belfrage

August Borna

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6200
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2023

Abstract

Synthetic data generation has emerged as a valuable technique for addressing data scarcity and privacy concerns and improving machine learning algorithms. This thesis focuses on progressing the field of synthetic data generation, which may play a crucial role in AI-heavy industries such as telecommunications. Generative Adversarial Networks successfully generate various types of synthetic data but fall short when modelling the temporal patterns and conditional distributions of time series data. State-of-the-art TimeGAN has shown promise, but there is potential for refinement. We propose *T2GAN*, utilising TimeGAN's novel framework of combining unsupervised and supervised training and extending it using state-of-the-art machine learning techniques, such as Transformers. Through experimental evaluation, we quantify the effectiveness of T2GAN using various benchmark data sets and find that the T2GAN model significantly surpasses the TimeGAN in both discriminative and predictive capacities. Our results demonstrate a 38% enhancement in similarity measures and a 55% reduction in relative prediction error when using synthetic training data. Furthermore, the thesis presents a comprehensive literature study and analysis of generative models, detailing the potential of T2GAN in various domains by enabling privacy-preserving data analysis, facilitating research and development, and enhancing machine learning algorithms.

Acknowledgements

We thank our supervisor, Torgny Holmberg, for his guidance and support throughout this project. We are also grateful to our colleagues Robert Marklund and Johan Ruuskanen for their helpful input and discussions. Additionally, we appreciate the valuable advice from our academic supervisor, Johan Eker, and our examiner, Karl-Erik Årzén. Their collective contributions have been instrumental in the completion of this thesis.

Contents

Acronyms

AI Artificial Intelligence

ICT Information and Communications Technology

SOTA State-of-the-art

GAN Generative Adversarial Network

VAE Variational Autoencoder

MNO Mobile Network Operator

ANN Artificial Neural Network

CNN Convolutional Neural Network

RNN Recurrent Neural Network

LSTM Long Short-Term Memory

GRU Gated Recurrent Unit

DCGAN Deep Convolutional GAN

CGAN Conditional GAN

WGAN Wasserstein GAN

WGAN-GP Wasserstein GAN with Gradient Penalty

ADSGAN Anonymization through Data Synthesis GAN

PCA Principal Component Analysis

t-SNE t-distributed Stochastic Neighbour Embedding

1

Introduction

Data-centric Artificial Intelligence (AI) is a rapidly evolving field transforming how we interact with technology. At the heart of this transformation is the ability to collect, process, and analyse large amounts of data to make predictions, automate tasks, and optimise decision-making in an agile way. The transition is necessary to facilitate emerging technologies and will be the key to unlocking effective large-scale AI systems [61].

As we continue to push the boundaries of what is possible with AI, many challenges are yet to be overcome before generalised and fully autonomous AI systems are achievable. Machine learning is expected to replace traditional software engineering progressively, starting and ending with data. One challenge is the need for high-quality, diverse, and representative training data, essential for reliable AI models [61]. Unfortunately, getting high-quality data is cumbersome, mainly due to extensive data collection and preparation processes. Furthermore, true data-centricity is hampered due to data proprietaries. As a result, progress is inefficient, and growth occurs within silos, with function-specific models rather than generalised innovations [4].

Enter synthetic data. Synthetic data generators are powerful tools that have the potential to generate large amounts of data representative of the natural world, mitigating data-centric adversaries such as data insufficiencies and privacy considerations [22, 48, 65]. This can revolutionise how we develop and deploy AI models by enabling us to train models on large and diverse synthetic data sets otherwise unavailable in the real world.

Consequently, developing practical synthetic data generators that can enhance data quality and enable interchangeable data sets has profound implications for research and businesses. Simply put, it would allow companies unprecedented access to arguably one of the world's most valuable resources. By 2024, 60% of the data used in AI is expected to be synthetic to supplement the limited original sources and feed the large-scale AI model's hunger for data [12].

For Ericsson and the Information and Communications Technology (ICT) industry, synthetic data is an attractive research area [17]. The future of ICT is piloted by the development of 5G and 6G, predestined to rely heavily on autonomous systems and cloud-based operations, requiring a data-centric strategy. Access to data is reportedly a significant barrier to AI and machine learning progress at Ericsson, where synthetic data can mitigate these problems. Another issue, typical when training deep neural networks, is data scarcity resulting in impractical algorithms being unable to converge appropriately.

Previously, data generation has been confined to probabilistic models treating the data as random variables and then sampling based on their modelled underlying joint distribution. However, recent advancements in generative models, such as Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs), show promising results as they offer greater flexibility and performance in representing the data. In addition, they can handle complex data types, including numerical, deterministic, categorical and textual inputs, while capturing temporal dependencies and potentially functioning as an anonymisation technique [64]. However, although deep generative models have shown great potential, several shortcomings must be addressed.

This thesis will explore novel contributions to synthetic data generation, mainly using GANs, and detail their associated challenges. The focus will be moving from tabular data to the more complex streaming setting with time series data. The goal is to explore the current state-of-the-art methods and develop a practical synthetic data generator tool as a proof of concept for future research.

1.1 Related Work

In synthetic data by machine learning, image synthesis using GANs (Nvidia's StyleGAN 3), text-to-image models (OpenAI's DALL-E 2), and plain text generation using natural language processors (OpenAI's GPT-4) have recently received much attention. However, considerable efforts are also exploring the synthetic generation of tabular data using GANs. This thesis will take off from the current state-of-the-art methods for generating tabular data presented by [46, 45, 64] and time series data by [66] and extend their innovations to overcome some of the main challenges, finally exploring the applicability for generating data in a streaming setting [21]. Like much of the previous work, our model will employ a combination of the fundamental advancements made in GAN architecture since it was first introduced in 2014 [22] but also include successful components from the modern Transformer architecture [58].

Despite the efforts in generating synthetic data using GANs, there are still challenges to address. For example, one often desired feature when generating syn-

thetic data that resembles the underlying distribution is maintaining the privacy of the original data. So far, the privacy guarantees from most privacy-oriented GAN frameworks must be more comprehensive for many healthcare, finance and telecommunications applications. For example, some featured models in the literature achieve the most common privacy definitions [29, 36]. On the other hand, less explored are guarantees beyond these standards. Still, the authors of [65] argue for a different technique and privacy metric when generating data such as healthcare records to cater to more rigid and interpretable privacy requirements.

1.2 Problem Formulation

The hypothesis is that it is possible to generate synthetic data suitable for training accurate machine learning models capable of, e.g., detecting anomalies in cloud servers. The project will primarily concern applications within ICT, but the approach is generic and applies to any industry.

Within telecom, a profound issue is that the data is highly proprietary, prohibiting data-centric analysis and modelling across operators. Furthermore, the advancement of 6G and its dependence on autonomous AI motivates cutting-edge tools for synthetic data generation. The project aims to act as a proof of concept of to what extent it is possible, in a general sense, to use GANs to generate synthetic data.

Additionally, the current methods for generating data using GANs often rely on hand-crafted components or extensive pre-processing techniques, limiting the model's generality. To address this issue, we aim to develop a flexible and scalable model to generate synthetic data directly from raw data without relying on hand-crafted features or extensive pre-processing techniques. The goal is to produce a flexible model for time series data capable of dealing with temporal dependencies, high dimensionality and critical outliers.

Successfully developing such a tool may enable ICT providers to access and harness a new domain of Mobile Network Operator (MNO) data – offering further research- and business opportunities.

1.3 Scope and Delimitations

This research comprises two main parts aimed at advancing the field of synthetic data generation, with a particular focus on time series data during experimentation.

The first part consists of an extensive literature study that explores state-of-the-art synthetic data generators, focusing primarily on GANs for both tabular and time series data. This investigation provides a foundation for understanding the current

landscape of synthetic data generation techniques, their limitations, and potential areas for improvement.

The second part of the research involves the experimentation and development of an improved framework for generating synthetic data. This part of the research aims to create a more robust and practical approach to synthetic time series generation by building upon the insights gained from the literature study and addressing the limitations of existing methods.

In terms of scope, this research primarily focuses on developing and evaluating an improved GAN-based framework for synthetic time series data generation. It does not cover all possible synthetic data generation techniques or address other data types, such as text or images. The experiments are also limited to the specific data sets used for evaluation and comparison, and the results may vary when applied to other data sets or domains.

Delimitations of this research include focusing on GANs as the primary method for synthetic data generation instead other generative models, such as VAEs or Transformer architectures. Furthermore, the research is conducted within the context of tabular and time series data. Thus, the developed framework may not directly apply to other data types without modifications or additional considerations. Additionally, although thoroughly described in the literature study, the experimentation does not directly test the privacy aspects of the synthetic data. Instead, it provides some insights into the discussion regarding the privacy of synthetic time series data.

Despite these delimitations, this research makes significant contributions to the field of synthetic time series data generation, offering valuable insights and potential directions for future work. By investigating state-of-the-art methods, developing an improved framework, and evaluating its performance, this research demonstrates the potential of GAN-based techniques for generating realistic and practical synthetic time series data.

1.4 Individual Contributions

Both authors have been deeply involved in all parts of the research, including literature review, model development, experimentation, and testing. Naturally, some division of labour has occurred, such as leading different literature review areas. Furthermore, we have followed an iterative and collaborative approach throughout the model development process, ensuring efficient teamwork.

For instance, Esaias focused on developing the multi-GPU setup during the experiment setup and preparing for implementing the T2GAN in PyTorch. Meanwhile,

August examined various hypotheses and ideas using the existing TimeGAN frameworks to learn the model.

During model development, we worked in parallel, individually exploring different aspects of the framework or training unique models. Esaias focused on experimenting with multi-head attention and transformer concepts, while August compared the performance of different network architectures, including Long Short-Term Memories (LSTMs), Gated Recurrent Units (GRUs), and Convolutional Neural Networks (CNNs). This process identified the most suitable architecture for our synthetic data generation model.

In addition, regular meetings and code reviews helped maintain consistency and a shared understanding of the project, ensuring a seamless integration of our contributions.

2

Background

2.1 Leveraging Synthetic Data in 6G Communications

Telecommunications and information technology are evolving rapidly. Each new generation, from 4G to 5G and now looking towards 6G, brings unique opportunities and challenges [63]. During the 4G era, the advent of cloud-based artificial intelligence facilitated complex data analysis and machine learning tasks on the go. Dedicated hardware components handled specific network tasks, offering a manageable yet inflexible system.

With the shift to 5G, core network functions began migrating from these fixed components to a virtual cloud environment. Although this increased flexibility, it simultaneously introduced management complexities. We foresee an even more significant shift as we anticipate the advent of 6G. The network infrastructure is expected to become entirely virtualised and distributed, providing impressive flexibility and efficiency but vastly increasing system complexity.

This evolution towards more complex, distributed systems presents a significant challenge for telecommunications actors. In such distributed systems, operators host numerous services, making parts of the system or data inaccessible. As a result, the industry must develop ways to autonomously collect, process, and label data from these widespread sources for efficient system maintenance, such as anomaly detection.

Synthetic data generators could be a solution to these challenges. These tools operate between data owners and end users, providing a pipeline to model and conduct system maintenance on proprietary data sources. Synthetic data is artificial data generated by a model designed to have similar statistical properties as the original. It is beneficial when real-world data is either unavailable or confidential.

Synthetic data can augment existing data sets where data is scarce, enabling robust machine learning model training. Alternatively, when data confidentiality due to

privacy concerns poses a problem, synthetic data can be shared and utilised without revealing personal and directly identifiable information [42]. The increasing demand for synthetic data across industries and the growing system complexities in telecommunications underscores the importance of advancing synthetic data generation techniques.

2.2 The Need for Privacy

For the ICT industry, efficient data-centric functions of 5G and 6G networks will be essential to enable the full-scale transition. At Ericsson, privacy-preserving techniques can be utilised in many different ways [16]. One immediate application is a synthetic data generator tool that can be used in-house and with partners to share sensitive data sets or extend insufficient ones. Data volumes are expected to increase immensely over the coming years, requiring a scalable and adaptive data pipeline. For example, consider operator data, which is often sensitive and cannot be shared openly [63]. For instance, it can be location-aware and contain information that can be directly linked to a user and is thus required to be obfuscated or otherwise anonymised.

Anonymisation techniques aim at making it difficult to link the data to a specific individual [38]. However, it is insufficient to anonymise data, such as replacing customer names with customer IDs. In a prize challenge, Netflix manually anonymised 100 million movie ratings using movie-id, customer-id and movie-rating. Nevertheless, a research group [41] later proved it possible to uncover customer records and other sensitive information using very little knowledge of a customer. Therefore, the increasing demand for data-driven decision-making has led to more sophisticated anonymisation techniques, ranging from data masking to advanced machine learning algorithms. However, every privacy-preserving method comes with a trade-off between the risk of privacy violation and data usefulness [15], and a future of democratised data needs more effective ways. The following briefly introduces the most common anonymisation techniques for sensitive data sets.

Consider a private user table D of multiple records, where each record is linked to a user by a unique user ID. Furthermore, each record can have one of four attributes [38]:

- **Direct identifier:** information that can directly identify a user (e.g. name, email address or SSN).
- **Quasi identifiers:** can be linked via auxiliary information to reveal someone's direct identifier or sensitive attribute (e.g. age, gender or geographic location).

- **Sensitive attributes:** personal information that should not be revealed (e.g. salary, disease or political/religious views).
- **Non-sensitive attribute:** if the record is none of the above (e.g. height, weight or eye colour).

An adversary can threaten to disclose information in three different ways [38]: *Identity disclosure, attribute disclosure* and *membership disclosure*. As the name suggests, identity disclosure is when an adversary can associate an individual with a direct identifier. Attribute disclosure is when an individual is linked to a sensitive attribute, and membership disclosure is when it is possible to deduce that an individual is present/absent in the data set.

Table 2.1 Example of a sensitive data table.

Direct Identifier	*Quasi Identifiers*		*Sensitive Attribute*
ID	Zip Code	Age	Net Income
1	23250	20	240,000
2	23750	27	320,000
3	22210	31	240,000
4	22152	48	710,000
5	22160	36	230,000
6	22252	50	370,000

Syntactic models of anonymity

Syntactic models of anonymity are privacy-preserving techniques that focus on data structure and format. They transform data by removing or replacing identifiers, generalising quasi-identifiers, and grouping records to protect privacy and maintain overall correctness. However, these methods can sometimes reduce data granularity and usefulness.

The first and most common anonymisation technique is replacing direct identifiers with IDs. However, as shown by [41] in the Netflix prize challenge, such methods are not enough to protect user privacy even against simple attacks. Most syntactic models, therefore, also generalise the table records by grouping values into categories. This preserves the correctness of the data and protects individuals by making the records indistinguishable, but it can significantly hamper data usefulness.

K-anonymity is a privacy concept used to protect sensitive information in large data sets and utilises the grouping technique mentioned above. The goal of k-anonymity is to anonymise the direct and quasi-identifiers in a data set and divide the records

into blocks, ensuring that there are at least $k-1$ other records in the data set with similar or identical characteristics, such that they are indistinguishable [15]. In other words, k-anonymity measures a data set's "clustering" of matching records.

Table 2.2 Example of a 3-anonymous data table.

Direct Identifier	*Quasi Identifiers*		*Sensitive Attribute*
ID	Zip Code	Age	Net Income
1	2****	20-35	240,000
2	2****	20-35	320,000
3	2****	20-35	240,000
4	22***	36-50	710,000
5	22***	36-50	230,000
6	22***	36-50	370,000

There are several weaknesses with k-anonymity pointed out by [33] [37], primarily that it does not guarantee enough diversity of sensitive attributes within each block. Instead, [33] proposes adding a constraint called *l-diversity*. l-diversity requires that each equivalence class (block of records sharing the same quasi-identifier values) in the anonymised data set contains at least l distinct values for the sensitive attribute. This constraint aims to prevent attribute disclosure by ensuring sufficient diversity in sensitive information within each group, making it harder for an attacker to confidently infer an individual's sensitive attribute based on their quasi-identifier values. Note that Table 2.2 is 3-anonymous but only 2-diverse since there are a minimum of 2 distinct sensitive values for each group of k records (two sensitive attributes are identical, Net income = 240,000 for individual 1 and 3). An example of a 3-anonymous and 3-diverse table is illustrated in Table 2.3, which is 3-diverse because each group of records sharing the same quasi-identifier values contains at least 3 distinct sensitive attribute values. This increased diversity in Table 2.3 makes it more difficult for an attacker to infer an individual's sensitive information based on their quasi-identifiers. In addition, it is possible to further protect the data by adding a *t-closeness* constraint, which ensures that the distribution of sensitive values in a data set is similar to the distribution of sensitive values in the underlying population bounded by a threshold t.

The literature proposes variants of the methods above that do not damage the data utility as $k-$ and l-diversity depending on the application [62, 2, 54]. One example is the Anatomy anonymisation method [62], which similarly divides the data into blocks. However, instead of generalising the quasi-identifiers, it randomly shuffles the sensitive attributes, yielding equivalent privacy guarantees as l-diversity.

Table 2.3 Example of a 3-anonymous, 3-diverse data table.

Direct Identifier	*Quasi Identifiers*		*Sensitive Attribute*
ID	Zip Code	Age	Net Income
1	2****	20-40	240,000
2	2****	20-40	320,000
5	2****	20-40	230,000
3	22***	30-50	240,000
4	22***	30-50	710,000
6	22***	30-50	370,000

Differential privacy

Differential privacy is a mathematical definition used to design algorithms that preserve privacy in data analysis [15]. Differential privacy aims to allow data scientists to extract useful information from sensitive data while hiding individual identities and minimising the risk of re-identification. Another way to put it is it provides individuals with a similar degree of privacy as if their information were deleted from the data set. This is done by ensuring it is impossible to infer whether a single individual was included by running statistical functions on the data set. Note that an individual's contribution to a data set depends on the size of the data set. Thus, differential privacy is a formal mechanism of how much random perturbation must be added to the query result to produce equivalent privacy.

The strength of differential privacy is that it assumes that an attacker knows all records in the data set except one and yet restricts an attacker from violating the privacy of that individual. It is also a formal mechanism for adding noise to the query results. Finally, it is composable, where in the case of differential privacy, multiple queries can be performed on the same data set while still maintaining the confidentiality of the individuals in the data set. This allows for more flexible and efficient use of the data without sacrificing privacy.

Let's consider an example to get a better sense of the applications. Again, consider a private table (see Table 2.4) of multiple records, where each record is linked to an individual by an ID. The sensitive attribute is, in this case, "disease status", which is a categorical value where $0 =$ healthy and $1 =$ diseased. The quasi-identifiers are zip code and age, which are discrete values.

We can use differential privacy to add noise to the data before releasing information about the population (see Table 2.5). For example, we could add Laplace noise to perturb the disease status. Laplace noise samples from the Laplace distribution, which is heavy-tailed and can yield both positive and negative values [19]. The amount of noise is determined by the scale parameter b, where a smaller value

Table 2.4 Raw data containing sensitive information.

Individual	Zip Code	Age	Disease Status
1	90210	25	1
2	12345	32	0
3	56789	40	1
⋮	⋮	⋮	⋮
100	11111	29	0

results in more perturbation and, thus, better privacy protection.

Table 2.5 Differentially private data.

Individual	DP Zip Code	Age	DP Disease Status
1	90210	25	0
2	12345	32	1
3	56789	40	1
⋮	⋮	⋮	⋮
100	11111	29	1

The column disease status is altered by some level of noise determined by the privacy budget, i.e., the level of desired privacy protection. It is worth noting that there are other ways to add noise, and they can differ significantly in effectiveness depending on the situation. Consider an example where we sell lemonade and have three potential buyers willing to pay 1€, 1€, and 3.01€, respectively. As a seller, if we price at 1€, the revenue becomes 3€, pricing at 1.01€ the revenue drops to 1.01€, pricing at 3.01€ the revenue increases to 3.01€ and finally, pricing at 3.02€ the revenue drops to 0€. In this case, choosing how to price is non-private since it will be reasonably easy to deduce whether, e.g., the "rich person" is in the data set. Therefore, adding noise does not seem practical if changes to the sensitive attribute (willingness to pay) have these effects. The lemonade example demonstrates the need for understanding the data and the goal of the analysis when employing differentially private methods.

Furthermore, perturbing the data can obscure or even change the underlying relationships between features, potentially impacting the performance of, e.g., a machine learning model. For example, if there is a positive correlation between age and disease in Table 2.5, depending on the number of samples and the representation of different age groups, adding noise could make the relationship less clear, leading to incorrect decisions and predictions. Therefore, like with the syntactic methods, it is essential to consider the trade-off between privacy and data utility

when using differential privacy.

Anonymisation through generative models

While syntactic methods have apparent drawbacks, quickly damaging data usefulness, differential privacy is often applied when we do not see the raw data and instead query a particular algorithm or analysis. Therefore, it is interesting to investigate other methods of anonymisation. In many cases, the most meaningful aspect of privacy is assessing the identifiability between the original data and the generated synthetic data. For this purpose, the literature suggests various generative models, which will be more thoroughly described in section 2.4, such as the VAE and GAN.

The recent developments in GANs have successfully generated highly realistic images [30] and other types of data, such as tabular data and time series [64, 20] or variations fulfilling differential privacy constraints [56, 29]. However, if assessing identifiability is the objective, these methods do not guarantee privacy. Although there is no general framework for this in the literature, [65] proposes a novel approach by introducing "Identifiability" and mathematically defining it as a metric when assessing the privacy level of synthetic patient data using a unique GAN architecture.

2.3 Data Formats and their Characteristics

Data comes in many shapes and forms, with varying approaches and synthesising techniques. This thesis will focus on the following data types: tabular data and time series data. These data types can also be considered in an offline or online setting.

Tabular data

Tabular data is organised in rows and columns and can contain all the data types above. The columns represent different data features; each row is a single record of multiple features. Tabular is a straightforward way of storing and analysing data, but synthesising tabular data is not trivial. One issue with generating tabular data is the *semantic integrity* within records. While synthesised data might catch each column's underlying distribution and correlation, synthetic values might be impossible. For example, generating a medical record of a patient with a cholesterol level of 60 mg/dL (normal) and labelling them as diabetic would be semantically incorrect [45]. Another issue is that each feature can be continuous, discrete, categorical, textual, and temporally dependent.

Time series data

Time series data refers to data with temporal dependencies and a relationship between data points. The challenge with time series data is capturing temporal pat-

terns such as auto-correlation and periodicity. Examples of time series data are stock prices and weather data. Note that time series data often come in a tabular format. Nevertheless, we distinguish between tabular and time series data since they are processed differently when synthesised using GANs, which is more thoroughly described in Section 2.9.

2.4 The History of Synthetic Data Generators

The advent of synthetic data generation has spurred the development of various techniques, each with its unique approach. Beginning in the 1950s, von Neumann introduced the Markov Chain Monte Carlo (MCMC) methods [50], which generated synthetic data by sampling from an unknown or known distribution. Around the same time, Sklar introduced copula methods to model the relationship between variables, separated from the variables' marginal distributions [10]. A copula is a mathematical function which connects several variables' joint distribution with their marginal distribution and generates synthetic data by sampling from the marginal distributions and combining the samples.

In the 1980s, Pearl pioneered the next breakthrough introducing Bayesian networks [47]. It is a graphical approach where the idea is to find a directed acyclic graph of several variables, where edges between nodes represent dependence between variables. By utilising the directed acyclic graph, it is possible to construct a conditional probability table for each variable. Synthetic data can then be generated by first selecting a starting variable, sampling a value for the variable using its marginal distribution, and finally, using the directed edge to propagate the sample to the other nodes in the graph and sample a new value at each node based on its conditional probability table.

Simultaneously, in the 80s, Boltzmann machines entered the scene and were the first approach utilising artificial neural networks [27]. A Boltzmann machine is a network of binary nodes where each node is either "on" or "off", with learnable, weighted edges connecting the nodes. Each node represents a variable and the edge between their interaction. The goal is to learn the weights for the machine to generate synthetic samples.

In 2013 Kingma et al. introduced the variational autoencoder [31], which uses a deep learning approach. The general idea is to train one neural network to map input data to a lower dimensional latent space and one neural network that aims to reproduce the input data from the latent space. If successful, it is possible to sample from this latent space to generate new synthetic samples. The VAE is detailed in Section 2.6. Goodfellow et al. introduced the GAN in 2014 [22], letting two neural networks compete against each other. The generator receives noise as input and tries to produce realistic images that can fool the discriminator. Simultaneously, the

discriminator receives fake images alternated with real ones to determine whether they are real. See Section 2.7 for more details on the GAN architecture.

This thesis will only consider GANs since it is the most promising generation technique presented in the literature related to our purpose. However, before diving into the machine learning-based synthetic data generators, it is important to understand the relevant Artificial Neural Networks (ANNs) introduced in the next section.

2.5 Artificial Neural Networks

ANNs are computational models inspired by the structure and function of biological neurons [1]. The design simulates how the human brain processes information by forming complex networks of nodes, or artificial neurons, so the network can learn and make predictions based on data inputs. ANNs are composed of multiple layers, each containing multiple artificial neurons. The network processes inputs through the layers, finally outputting from the output layer (see Figure 2.1). The learning process in ANNs occurs by adjusting weights associated with each connection between the artificial neurons. The weights are updated based on the difference between the actual and desired output by minimising a specific loss function. By repeating the process multiple times, using different data inputs, the network learns progressively to produce accurate outputs based on the inputs. There are several types of ANNs, including feed forward networks, convolutional networks, and recurrent networks - each type designed for a specific problem and data structure, such as image recognition, speech recognition, or natural language processing.

A single artificial neuron in an ANN is represented mathematically as:

$$y = f(\sum_{i=1}^{n} w_i x_i + b)$$

where y is the output of the neuron, f is the activation function, w_i is the weight of the i:th input, x_i is the i:th input, and b is the bias. The activation function f determines the neuron's output based on the weighted sum of inputs and biases. The most common choices are sigmoid, hyperbolic tangent (tanh), Rectified Linear Unit (ReLU) and leaky ReLU (see Figure 2.2).

Training an ANN aims to find the optimal weights and biases that minimise a loss function between the actual and predicted outputs. The loss function can be any suitable function, such as the mean squared error (MSE) or the cross-entropy loss, that quantifies the difference between the actual and predicted output. For instance

$$MSE = \frac{1}{N} \sum_{n=1}^{N} (y_n - \hat{y}_n)^2$$

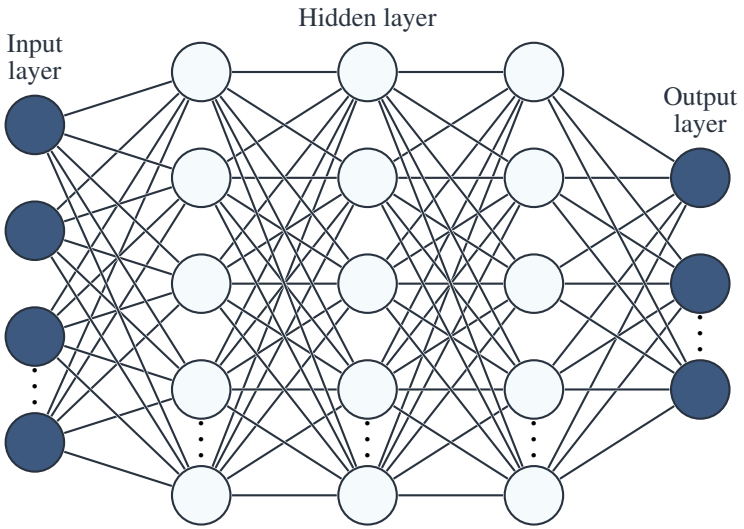


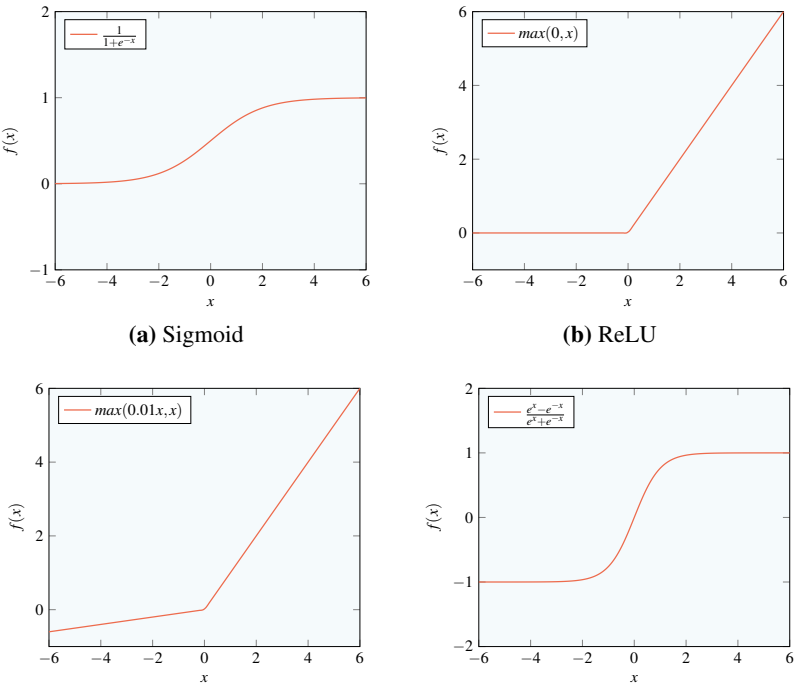
Figure 2.1 A fully connected artificial neural network with three hidden layers.

where y_n is the observed value and $\hat{y}_n$ is the predicted value. Once the loss function is defined, the minimisation problem takes the form:

$$\min_{\mathbf{W}} L(\mathbf{W}; \mathbf{x}, \mathbf{y})$$

where $\mathbf{W}$ is the vector of weights in the network, $\mathbf{x}$ is the input data and $\mathbf{y}$ is the target or label for the data. $L(\mathbf{W}; \mathbf{x}, \mathbf{y})$ is the loss function that measures the difference between the prediction of the network and the true target. The goal is to find the $\mathbf{W}$ values that minimise the loss.

The optimisation process involves techniques such as SGD (stochastic gradient descent) or Adam (Adaptive moment estimation) optimiser. The gradients of the loss function are computed during the training process, and the weights are adjusted iteratively based on the gradients. Backpropagation efficiently computes the gradients by propagating the errors backwards through the network, starting from the output layer towards the input layer. Training is executed in batches, i.e., a subset of the training data. The batch size determines the number of samples used in each iteration that compute gradients and update weights. Using batches gives more efficient computation and resource utilisation, and the choice of batch size depends on factors like available memory and the trade-off between computation time and convergence speed. The optimisation iterates over multiple epochs, each epoch being a complete pass of the training dataset. By employing optimisation techniques, training with batches, iterating over epochs, and utilising backpropagation, the ANN learns to optimise its weights to minimise loss.



(a) Sigmoid **(b)** ReLU

(c) Leaky ReLU **(d)** Tanh

Figure 2.2 Four common activation functions.

In addition, an ANN depends on several hyperparameters that must be defined before training. Hyperparameters are tuneable parameters, such as the number of hidden layers, the number of nodes in a layer, the learning rate and batch size, which can significantly impact the network performance. They are crucial to obtain a model that generalises well. A common problem for neural networks is overfitting, where the networks learn the training data too well. It often occurs when the network is overly complex, e.g., with too many parameters. There are multiple regularisation techniques available to avoid overfitting. Some examples are L1 and L2 regularisation, early stopping or dropout. L1 and L2 regularisation adds a penalty term to the loss function discouraging large weights commonly associated with overfitting. Early stopping sets aside a validation set and involves monitoring the validation error, stopping the training when it no longer improves significantly. Dropout is a regularisation technique where random nodes are ignored during training, reducing the dependence on single neurons.

Finding the proper network structure and fine-tuning hyperparameters can be difficult. The "black-box" nature of ANNs exacerbates the challenge. The inner work-

ings of an ANN are not easily accessible nor interpretable, making it difficult to understand how the network arrived at a particular decision or prediction. This lack of interpretability can limit stakeholders' trust in the predictions made by the network. It can make it challenging to use the network in specific applications, such as healthcare or finance, where interpretability and transparency are crucial. Additionally, the black-box nature of ANNs makes diagnosing and debugging the network challenging when errors occur. Despite these limitations, ANNs have proven to be powerful models for many applications due to their ability to learn complex patterns when large amounts of data are available.

Convolutional neural networks

CNNs are a category of ANNs, capable of processing data with grid-like structures, such as images [24]. The overall architecture is similar to ANNs, but in contrast, each neuron only connects to a small input region. The usual building stones of a CNN are convolutional layers, pooling layers and fully connected layers. Convolutional layers can operate in one dimension, two dimensions, and subsequently in the time dimension. See Figure 2.3 for an example of a 2D convolution. For example, this network could classify images of handwritten digits between $1-9$, where the image size is 28x28.

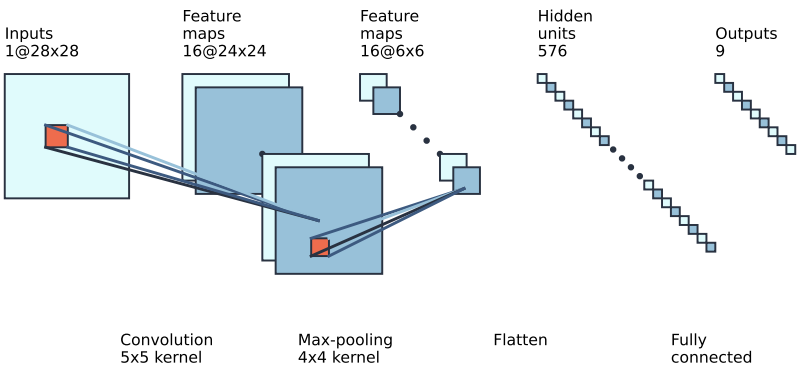


Figure 2.3 Illustration of a 2D convolutional neural network.

There are different versions of pooling layers, such as max pooling and average pooling. Max pooling returns the maximum value of a region, whereas average pooling returns the average of the same; see fig 2.4 for details.

Figure 2.5 details a convolution filter operation where the trainable parameters are the ones in the filter.

The basic idea is to extract local features with the convolutional layers and reduce spatial dimension and feature resolution with pooling layers. Then, the network

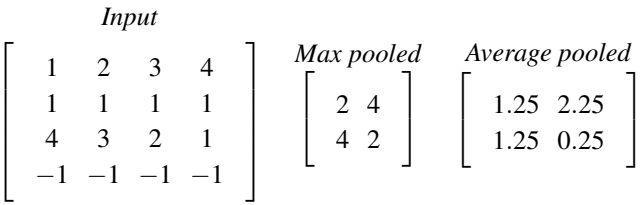


Figure 2.4 Max and average pooling in 2D.

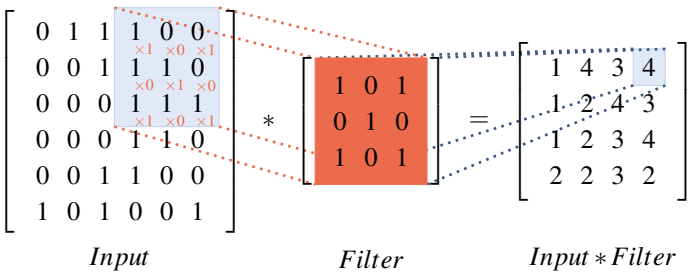


Figure 2.5 One 2D convolutional filter operation.

gradually extracts higher-level feature representation by stacking several convolutional and pooling layers [24]. After reducing spatial dimension sufficiently, a fully connected layer is typically used to perform classification or regression. Finally, the fully connected layer processes the output of the previous layers and produces the final prediction. The training procedure is the same as with an ANN, i.e., minimising a loss function that measures the difference between the predicted and actual values and then updating the weights accordingly. Concluding, the ability of CNNs to automatically learn and extract complex and relevant features from images has made them popular in image classification and object recognition tasks.

Recurrent neural networks

Recurrent Neural Networks (RNNs) are a class of neural networks capable of processing sequential data, such as time series data. Unlike feed forward neural networks, RNNs can take variable length inputs and produce variable length outputs.

The critical feature of RNNs is the ability to utilise a hidden state updated at each time step with new input. The hidden state enables the network to maintain a record of previous inputs and their influence on the current output. Unfortunately, traditional RNNs suffer from the vanishing gradient problem, i.e., gradients become very small as they propagate back through time, resulting in poor performance on long sequences. LSTMs [28] and GRUs [13] were developed to mitigate this, introducing unique gating mechanisms, allowing the network to learn and forget information over time selectively. In addition, the mechanisms prevent the gradi-

ents from vanishing or exploding over long sequences, allowing the network to capture long-term dependencies better.

LSTMs use a memory cell and two states: the current hidden state h_t and the previous hidden state h_{t-1}. There are two cell states: the current c_t and the previous c_{t-1}. The memory cell stores the current state, which regulates the cell's information. The architecture is detailed in Figure 2.6 and the computations in an LSTM can be described by the following equations:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
c_t &= i_t \cdot \tilde{c}_t + f_t \cdot c_{t-1} \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where x_t is the input at time step t, σ is the sigmoid activation function, $\odot$ denotes element-wise multiplication, and W and b are the weight matrices and bias vectors, respectively, to be learned during training.

In addition to the mentioned states, LSTMs can be stacked to form deep networks. When stacking LSTMs, the output hidden state h_t of one LSTM serves as the input hidden state h_{t-1} of the next LSTM. Similarly, the output cell state c_t of one LSTM serves as the input cell state c_{t-1} of the next LSTM. This stacking process allows LSTMs to learn hierarchical representations and capture intricate dependencies in temporal data.

GRUs use a simpler architecture compared to LSTMs, involving three main components: the current input x_t, the current hidden state h_t, and the previous hidden state h_{t-1}. GRUs utilise two gates: the update and reset gates. The update gate determines how much of the previous hidden state to retain, while the reset gate controls the influence of the new input on updating the hidden state. The architecture of GRUs is depicted in Figure 2.7.

The computations in a GRU can be described by the following equations:

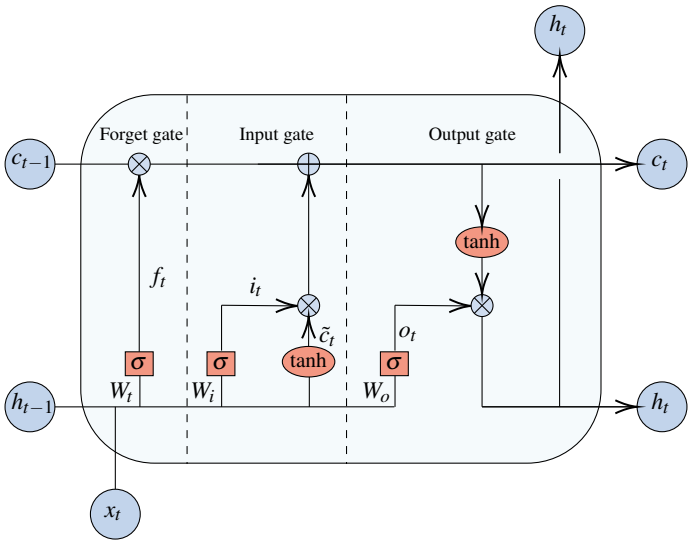


Figure 2.6 Architecture of an LSTM.

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) \\
r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) \\
\tilde{h}_t &= \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t] + b_h) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

where x_t is the input at time step t, σ is the sigmoid activation function, $\odot$ denotes element-wise multiplication, and W and b are the weight matrices and bias vectors, respectively, to be learned during training.

In addition to the mentioned states, GRUs can also be stacked to form deep networks. When stacking GRUs, the output hidden state h_t of one GRU serves as the input hidden state h_{t-1} of the next GRU. The process of stacking GRUs is similar to that of stacking LSTMs, allowing the model to learn hierarchical representations and capture complex dependencies in the data.

LSTMs and GRUs have succeeded in various applications, including natural language processing, speech recognition, and image captioning, and reportedly outperform vanilla RNNs in multiple benchmark tasks [14]. In addition, they are beneficial for processing long sequences and handling dependencies over time, hence their popularity in many deep-learning models.

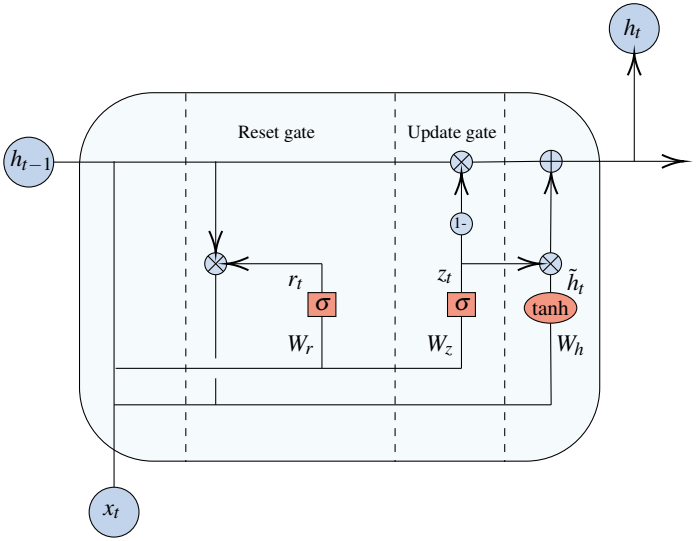


Figure 2.7 Architecture of a GRU.

Transformer neural network

Despite their effectiveness, LSTMs and GRUs still have limitations, especially when processing very long sequences, where they can be computationally expensive. Introduced in 2017 by Vaswani et al. [58], the *Transformer* is a neural network architecture that mitigates these problems by relying exclusively on self-attention mechanisms, eliminating the need for recurrent connections. As a result, the Transformer architecture outperforms RNN-based models in various natural language processing tasks, including language translation, understanding, and generation [49]. The critical element is the Transformer's self-attention mechanism, introduced in 2015 by Bahdanau et al. [6], which allows it to focus solely on the input sequence's relevant parts, thus making it more efficient at processing long sequences and capturing long-term dependencies.

The architecture consists of an encoder-decoder structure, where multiple layers of self-attention and position-wise feed forward neural networks make up the encoder and decoder parts. The self-attention mechanism allows the model to focus on and capture dependencies between relevant input parts. The position-wise feed forward neural networks process each token in the sequence separately and in parallel, in contrast to standard neural networks that operate on the sequence as a whole. Additionally, the architecture uses residual connections and layer normalisation to improve stability and performance. Furthermore, the model incorporates a positional encoding step to help the model differentiate between the position of tokens in the sequence. Since the self-attention itself only considers dependencies

between tokens and not their relative positions, this step is crucial. It assigns a unique vector to each token based on the token's position in the sequence, which, combined with the token's embedding vector, forms the input vector for the model.

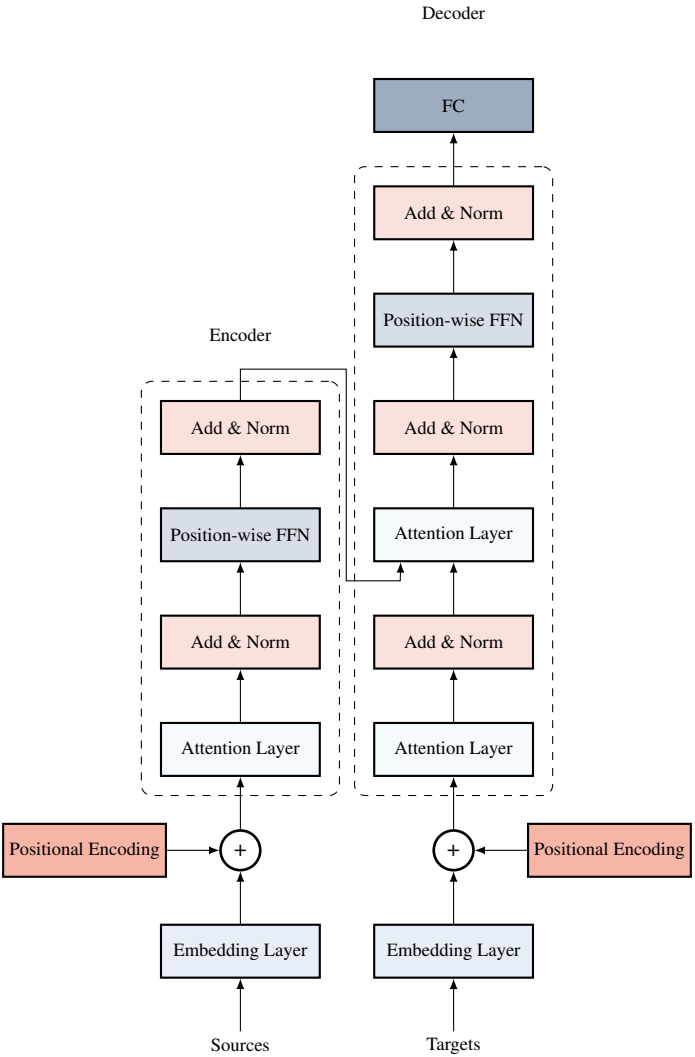


Figure 2.8 Transformer architecture.

The self-attention mechanism computes a weighted sum of the input values, where the weights are given by the dot product of a query vector Q_i and a key vector K_i. Specifically, for a given position i in the input sequence, the self-attention mech-

anism computes the dot product of the query vector, a function of the position i, with the key vectors, functions of all the positions in the sequence. The resulting dot product is then scaled and passed through a softmax function to obtain a set of weights to compute a weighted sum of the value vectors, which are functions of all sequence positions.

Formally, the self-attention mechanism is defined as follows. Given an input sequence of length N, the self-attention mechanism computes the query, key, and value vectors as follows:

$$Q_i = W_Q x_i$$
$$K_i = W_K x_i$$
$$V_i = W_V x_i$$

where x_i is the embedding vector of position i in the sequence, and W_Q, W_K, and W_V are learned weight matrices that transform x_i into the query, key, and value vectors, respectively. The self-attention mechanism then computes the attention weights between position i and j as follows:

$$\alpha_{ij} = \text{softmax}\left(\frac{Q_i K_j^T}{\sqrt{d_k}}\right)$$

where d_k is the dimensionality of the key vectors, whose division improves stability during training. Finally, the self-attention mechanism computes the weighted sum of the value vectors, V_j, using the attention weights, α_{ij}, as follows:

$$y_i = \sum_j \alpha_{ij} V_j.$$

The resulting output vector, y_i, is then fed into a feed forward neural network to obtain the final output of the self-attention layer.

The attention mechanism, however, is not restricted to usage within the Transformer architecture. It has shown success in time series classification due to its ability to focus selectively on the most informative instances within each time series.

An extension of self-attention is multi-headed attention, which enables the network to focus on multiple parts of sequences simultaneously by splitting the sequences and then employing the self-attention mechanism separately for each split part. Concatenating these and passing them through a linear layer yields the output. Consequently, multi-head attention is multiple parallel self-attention mechanisms working together, where the number of heads specifies the number of mechanisms. Concluding, self-attention focuses on a single input sequence, allowing a network

to attend to different parts of that sequence. In contrast, multi-head attention allows the network to attend to multiple parts simultaneously.

Lastly, the Transformer architecture utilises layer normalisation. Initially introduced by [5], layer normalisation is a technique similar to batch normalisation but specially designed for sequential inputs and RNNs. It aims to mitigate the issue of internal covariate shift that occurs within the hidden layers of a deep neural network, i.e., the change in the distribution of the hidden layer activations during training, making it difficult for the layers to adapt to these changes, slowing down the training process and requiring lower learning rates. The operation is defined as:

$$LN(x) = \gamma \frac{x - \mu_x}{\sigma_x} + \beta \tag{2.1}$$

where x is the input vector, μ_x and σ_x are the mean and standard deviation of x computed for each sequential element along the feature dimension, respectively, and γ and β are learnable scale and shift parameters.

2.6 Variational Autoencoder

The VAE is a neural network architecture introduced by Kingma et al. in 2013 [31]. The architecture consists of an encoder and decoder part, where the encoder takes input data and maps it to a lower dimensional latent space, corresponding to parameters of a variational distribution (see Figure 2.9). On the other hand, the decoder part gets a sample from the variational distribution and tries to reconstruct the original input data. The choice of latent distribution is free and may be chosen depending on the circumstances. The reconstruction error is formed with a regularisation term, using backpropagation to learn as a standard neural network. The regularisation term is a KL-divergence term, i.e., a Kullback-Leibler-divergence term, a statistical difference between the latent and target distribution. It enforces continuity and completeness in the model. Continuity means that the model attains a closeness property, i.e., samples close to each other in the latent space should be similar when decoded. If, for instance, the VAE trains on images, two samples close to each other in the latent space should generate closely resembled images. Completeness means that any samples from the distribution should result in meaningful data when decoded, essentially removing any "gaps".

VAEs have been successful in generating synthetic data and are more straightforward to train than, e.g., GANs, but in turn, offer less flexibility. Furthermore, GANs have the favourable property of never seeing original data during training. However, it is difficult to compare the performance of VAEs and GANs when generating synthetic data since it depends on the application [34].

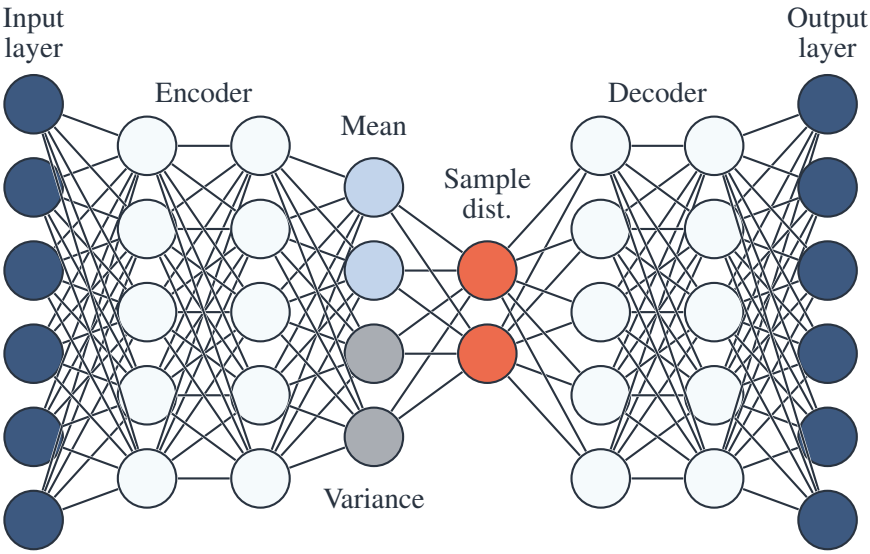


Figure 2.9 A fully connected Variational Autoencoder.

2.7 Generative Adversarial Nets

GANs are a class of neural network architectures that have significantly impacted machine learning since Ian Goodfellow and his colleagues at the University of Montreal introduced them in 2014 [22]. They have proven to be a flexible and powerful tool for generating new, synthetic samples that resemble real-world data. The unique architecture works by training two competing neural networks, a generator and a discriminator, in a two-player game. The generator aims to produce realistic synthetic data that can fool the discriminator, while the discriminator aims to distinguish between natural and synthetic data. During training, the generator updates its parameters to produce better and better synthetic samples. Simultaneously, the discriminator updates its parameters to become more and more effective at detecting synthetic data. This competition between the generator and discriminator continues until the generator reaches a point where it produces synthetic data indistinguishable from actual data, as determined by the discriminator.

GANs can formally be defined as a min-max two-player game. Let G be the generator network and D be the discriminator network. The generator takes a random noise vector z from a prior distribution p_z and produces a synthetic data sample $G(z)$. The discriminator takes either a real data sample x from the actual data distribution p_{data} or a synthetic sample $G(z)$ from the generator and outputs a scalar $D(x)$ or $D(G(z))$ representing the probability that the input is a real sample. The

objective function of the GAN becomes the min-max optimisation problem:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \quad (2.2)$$

where the expectation is taken over the distributions p_{data} and $p_{\mathbf{z}}$. The discriminator D trains to maximize this objective, while the generator G trains to minimize it. The result is a game-theoretic equilibrium where the generator produces indistinguishable samples from real samples, as determined by the discriminator.

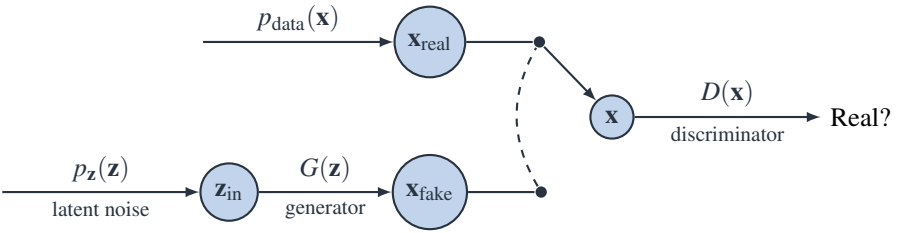


Figure 2.10 Generative adversarial net.

Conditional GAN

Conditional GAN (CGAN) extend the basic GAN framework to a conditional model [40]. In a CGAN, the generator and discriminator receive additional input as a conditional vector y, which can be any auxiliary information such as class labels or data from a different modality. The objective function becomes:

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

The conditioning allows them to generate diverse and class-conditional outputs, such as images of specific objects, faces with specific attributes, and audio with specific properties. In practice, it is done by feeding $\mathbf{y}$ as an additional input layer to the generator and the discriminator. In the generator, $\mathbf{y}$ and the input noise $p_{\mathbf{z}}$ are combined in a joint hidden representation.

Deep convolutional GAN

Deep Convolutional GAN (DCGAN) is a GAN architecture for generating high-resolution images. The architecture employs convolutional layers instead of fully connected layers in both the generator and discriminator (see Figure 2.3 for an illustration of an ANN with convolutional layers). The DCGAN can then exploit the spatial structure in image data, leading to high-quality synthetic images. Motivated by recent findings in convolutional neural nets, the authors suggest eliminating connected layers on top of convolutional features and using Batch Normalisation for some layers to stabilise learning. As a result, DCGANs are better suited for generating complex, high-quality images, including realistic photographs, faces, and even entire scenes.

Wasserstein GAN

The Wasserstein GAN (WGAN) is a modification of the GAN architecture which improves training stability and mitigates other frequently occurring problems with GANs, such as mode collapse [3]. Mode collapse occurs when the generator becomes too confident in its output, generating samples from a minimal set of modes instead of from the complete target distribution, even though samples from missing modes exist in the training data [52].

In addition, it introduces a new distance for the loss function, the *Earth Mover's* distance (EM), i.e., the *Wasserstein-1* distance, which enhances convergence capabilities. The EM distance estimates the distance between the actual and the generated distribution and consequently provides a more meaningful loss metric, replacing the traditional discriminator with a critic. The critic assigns high scores to samples it believes are real and vice versa. This differs from the original discriminator, which outputs a probability of a given sample being real. Another advantage of WGAN is the ability to train the critic until optimality, thanks to the continuity and differentiability of the EM distance. Training the critic to optimality is what cures the mode collapse problem. Note that the Wasserstein distance is central in most state-of-the-art GANs (and will be used later). Thus, we provide the full derivation.

The EM distance is defined as

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x-y\|], \tag{2.3}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of joint distributions $\gamma(x,y)$ such as their marginal distributions are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. An intuitive interpretation is that $\gamma(x,y)$ represents the amount of mass that needs to be moved from x to y to transform distribution $\mathbb{P}_r$ to $\mathbb{P}_g$. Hence, the EM distance represents the cost of obtaining this optimally.

Since the infimum of equation 2.3 is intractable, [3] suggests using the Kantorovich-Rubinstein duality [59]:

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

where the supremum is over all 1-Lipschitz functions $f : X \to \mathbb{R}$. Replacing $\|f\|_L \leq 1$ with $\|f\|_L \leq K$ gives

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

Let $\{f_w\}_{w\in\mathcal{W}}$ be a family of K-Lipschitz functions. Then, instead, consider the problem

$$\max_{w\in\mathcal{W}} \mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] - \mathbb{E}_{z\sim p(z)}[f_w(g_\theta(z)].$$

Utilising this with the standard GAN optimisation problem and standard notation yields the final min-max problem

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

where D is the discriminator and G the generator. Note the similarity with the original min-max problem, equation 2.2. To enforce Lipschitz-continuity, the article suggests using *weight clipping*, i.e., restricting weights to lie within the compact space $[-c,c]$ for some small constant c. However, this is a terrible way of enforcing such continuity as it can lead to further problems. Clipping weights restrict the model and make it harder to train an optimal model. We refer to the original paper [3] for detail and full proof.

Improved W-gan

In 2017, [25] improved the WGAN by forming a WGAN with gradient penalty (Wasserstein GAN with Gradient Penalty (WGAN-GP)). It tended to some issues with the WGAN, particularly enforcing Lipschitz continuity by weight clipping. The article proposes an alternative approach, using a gradient penalty for the critic. This cured many issues, such as improved training stability, showing promising results for many GAN architectures.

A differentiable function f is 1-Lipschitz if and only if $\|\nabla f\|_2 \leq 1$, i.e., the gradients' norm is less or equal to one. Consequently, the paper suggests directly constraining the norm of the gradient of the critic. However, since it is intractable to constrain the gradient norm everywhere, the authors introduce a softer version of the constraint, penalising the gradient norm for a set of random samples, $\tilde{\mathbf{x}} \sim p_{\tilde{\mathbf{x}}}$, instead. Here, $p_{\tilde{\mathbf{x}}}$ is a distribution obtained when sampling uniformly along straight lines between samples from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. This is motivated by the proposition that an optimal critic contains straight lines with gradient norm 1 connecting samples coupled from the data distribution and the generator distribution, proven in the article. Hence, the new objective function takes the form

$$\underbrace{\mathbb{E}_{z\sim p(z)}[D(G(z))] - \mathbb{E}_{x\sim p(x)}[D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda \mathbb{E}_{\tilde{x}\sim p_{\tilde{x}}}[(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2]}_{\text{Gradient penalty}}.$$

The article uses $\lambda = 10$ with good results. We refer to the article [25] for full proof and detail.

Further work in the literature tries to improve the approximation of Wasserstein distance [39]. At the same time, some argue that despite the improvements, WGAN does not achieve a meaningful approximation of the Wasserstein distance [53]. One suggested improvement is c-transform WGAN which replaces the Kantorovich-Rubenstein duality with a weak duality formula to improve mini-batch approximation (please see [39] for details on the c-transform). However, [53] argue mathematically and empirically that WGAN loss is not a meaningful approximation of the Wasserstein distance despite all efforts. Instead, they attribute the success of WGANs precisely to the *failure* of approximating the Wasserstein distance and that the Wasserstein distance is not a desirable loss function for deep generative models and high-dimensional data. Also, a large-scale study [35] comparing GAN loss functions showed that GANs are very sensitive to their hyperparameter setup and that no single loss function consistently outperforms the others. In particular, given the proper hyperparameter configuration, the vanilla GAN could achieve a comparable or better performance than WGAN-GP.

2.8 GANs for Tabular Data

As stated, GANs have evolved from their original application within image and text generation to tabular domain due to their great flexibility in modelling data distributions compared to traditional statistics. Given a table T_{real}, the generator's task in a GAN is learning the underlying distribution of each column in table T_{real} to generate synthetic data samples, eventually resulting in a table T_{fake}. Like most models, the GAN framework requires dividing the data into a training and a test set. The GAN is trained using the training samples and evaluated using the test samples. Performance is measured based on the generator's capability to model the underlying distributions and navigate the challenges with real-world data. If the objective also is privacy protection, a metric for such guarantees and data utility is required [8].

As stated in Section 2.3, tabular data often contains a combination of numerical (discrete or continuous), categorical and textual values. Furthermore, each feature a column represents can be non-Gaussian and multi-modally distributed. On the other hand, the imbalance problem is common for categorical features, i.e., an imbalanced representation of categories, which can lead to mode collapse and poor training for small classes. Furthermore, if categories are sparsely one-hot encoded, the discriminator may learn to recognise real versus synthetic data based on the rarity of certain categories instead of the real data's actual characteristics. In other words, the discriminator might associate rare categories with fake data simply because they appear infrequently in the data set.

The literature proposes several clever GAN architectures to deal with the complexity of real-world data. TableGAN [45] deals with the tabular data format.

The CTGAN [64] enhances the TableGAN by improving the non-Gaussian and multimodal capabilities. Finally, the Private Aggregation of Teacher Ensemble GAN (PATE-GAN) [29] generates differentially private synthetic data, while the Anonymization through Data Synthesis GAN (ADSGAN) [65] introduces an architecture for generating private synthetic data beyond the differential privacy guarantee.

TableGAN

Designed to model the complex statistical properties of tabular data, the TableGAN [45], introduced in 2018, can handle all types of table data while considering privacy and information leakage. The model has controllable parameters to determine the privacy level. The completely synthesised output table does not disclose actual records. However, increasing the privacy level decreases model compatibility, meaning that machine learning algorithms trained on the synthetic table will be less effective. The model adopts the DCGAN architecture but includes an additional neural network called *classifier (C)*. The classifier C tries to predict the synthetic records' labels which proved helpful in maintaining the consistency of values in the generated records. For instance, a record with gender = "Male" and disease = "Uterine Cancer" can be prevented as the classifier learns consistency and semantics from the original table.

CTGAN

Invented in 2019, the CTGAN includes mode-specific normalisation to overcome the non-Gaussian and multimodal distribution, a conditional generator, training-by-sampling to deal with imbalanced discrete columns, and finally, a fully-connected network structure. Previous models, such as TableGAN [45], use min-max normalisation to $[-1, 1]$ for continuous values, while the CTGAN uses the variational Gaussian mixture model (VGM) [7]. To overcome the class-imbalance problem associated with categorical values, the CTGAN builds upon a conditional generator (similar to the conditional architecture described in Section 2.7). The class imbalance occurs when training data is randomly sampled during training, and small classes are insufficiently represented, resulting in a poorly trained generator. Instead, training-by-sampling aims to resample categorical values evenly during training and recover the (not-resampled) real data during the test. The conditional generator inputs random noise and a *condition vector*. It then generates output, forced to mimic the specified condition. Furthermore, the model trains using the WGAN loss with gradient penalty, as described in [3]. Finally, the critic evaluates the output of the conditional generator measured by the difference between the learned and actual conditional distribution.

ADSGAN

The ADSGAN builds upon the Conditional GAN framework [40] (see Section 2.7) where the generator and discriminator are fed a conditional vector as input to condition the generator on some particular information. However, the ADSGAN optimises a conditioning set for each individual in the table and generates components based on these, compared to the Conditional GAN where the condition vector is predetermined, e.g., as data labels. This approach ensures that no combination of features could disclose an individual's identity. Moreover, it permits going beyond the differential privacy metric and instead introduce another metric, identifiability, based on the probability of re-identification given the combination of all data on any individual.

Identifiability as a privacy metric is specifically beneficial when sharing data sets. It poses the question of what is "different enough" from the original data to de-identify the original observations of individuals. The ADSGAN guarantees that two observations are different enough because they are "different" individuals and mathematically defines it as a difference in distance between original and synthetic observations. In particular, the distance is a weighted Euclidean distance. For example, consider the two features "male/female" and "cancer/non-cancer" patients. Now, if there are 50% males and females, respectively, and only 1% cancer patients, the distance between male/female patients should be closer than the distance between cancer/non-cancer patients; hence, the weighted distance.

By this motivation, ε-identifiability implies that there are less than ε ratio observations from the original data set ($\mathcal{D}$) in the generated synthetic data set ($\hat{\mathcal{D}}$) that are "not different enough" from the original observations. First, define a measurement r_i as the minimum weighted distance between an observation x_i and the other original observations in $\mathcal{D}$ and, similarly, $\hat{r}_i$ as the minimum weighted distance between an observation x_i and the generated observations in $\hat{\mathcal{D}}$. Mathematically, using r_i and $\hat{r}_i$, we can define ε-identifiability as follows.

Definition 2.8.1 (ε-identifiability)**.** *$\hat{\mathcal{D}}$ is ε-identifiable from $\mathcal{D}$ if*

$$\mathcal{I}(\mathcal{D}, \hat{\mathcal{D}}) = \frac{1}{N} \left[\mathbb{I} \left(\hat{r}_i < r_i \right) \right] < \varepsilon$$

where $\mathbb{I}$ represents the identity function and

$$r_i = \min_{\mathbf{x}_j \in \mathcal{D}/\mathbf{x}_i} \left\| \mathbf{w} \cdot (\mathbf{x}_i - \mathbf{x}_j) \right\| \tag{2.4}$$

$$\hat{r}_i = \min_{\hat{\mathbf{x}}_j \in \hat{\mathcal{D}}} \left\| \mathbf{w} \cdot (\mathbf{x}_i - \hat{\mathbf{x}}_j) \right\|. \tag{2.5}$$

Now, $1 - \varepsilon$ is the proportion of de-identified individuals in the synthetic data where 0-identifiability implies a perfectly anonymous data set and 1-identifiability as a perfectly identifiable data set.

2.9 GANs for Time Series Data

Time series data is prevalent in various fields, such as finance, medicine, and engineering. Its analysis aims to capture the temporal dependencies by finding the conditional distribution $p(\mathbf{x}_t|\mathbf{x}_{1:t-1})$ given a sequence $\mathbf{x}_{1:t} = (\mathbf{x}_1 \dots \mathbf{x}_t)$. Depending on the application, time series modelling is commonly associated with auto-regressive techniques through statistical, state-space or state-of-the-art deep learning models. The aim is to represent the distribution of sequences as a product of conditional probabilities, where each term corresponds to the probability of observing the next data point given previous observations. Mathematically, this is expressed as $\prod_{t=1}^{T} p(x_t|x_{1:t-1})$ and is essentially a deterministic approach in contrast to generative models which aims to sample new sequences without external conditioning randomly. Early attempts of employing the GAN architecture for generating time series data try to model the distribution of the entire sequence $p(\mathbf{x}_{1:t})$ directly without relying on the auto-regressive structure. However, [66] argues that this does not sufficiently capture the step-wise temporal dependencies and proposes a mechanism to combine the GAN research field with auto-regressive-based machine learning advancements resulting in the *TimeGAN*.

Due to the extra dimension in time, time series data pose unique challenges for generative models. While tabular data is modelled as a two-dimensional table, time series data is modelled in three dimensions with a 2D table for each time step. Usually, time series data sets come in a two-dimensional format, where each row corresponds to a sample in time, like the S&P 500 prices in Table 2.6 from Yahoo Finance.

Table 2.6 Daily S&P 500 prices and trade volume. The dashed area represents the sliding windows of the data.

Date	Open	High	Low	Close	Volume ('000)
09 Mar 2023	3,998.66	4,017.81	3,908.70	3,918.32	4,445,260
08 Mar 2023	3,987.55	4,000.41	3,969.76	3,992.01	3,535,570
07 Mar 2023	4,048.26	4,050.00	3,980.31	3,986.37	3,922,500
06 Mar 2023	4,055.15	4,078.49	4,044.61	4,048.42	4,000,870
03 Mar 2023	3,998.02	4,048.29	3,995.17	4,045.64	4,084,730
02 Mar 2023	3,938.68	3,990.84	3,928.16	3,981.35	4,244,900
01 Mar 2023	3,963.34	3,971.73	3,939.05	3,951.39	4,249,480
28 Feb 2023	3,977.19	3,997.50	3,968.98	3,970.15	5,043,400
27 Feb 2023	3,992.36	4,018.05	3,973.55	3,982.24	3,836,950

It is possible to achieve the 3D structure by sampling the data using a sliding window. This way, it is possible to create k sequences of time series data of length

l and dimension n. More specifically, if the S&P 500 data set contains 9 rows with 5 features of data (time-column is excluded), using a window length of, e.g., 7 and step size 1, the resulting data cube has shape $(3,(7,5))$. If, on the other hand, the data set consists of, e.g., multiple patient records of time series, the data can instead be reshaped into three dimensions directly.

TimeGAN

TimeGAN is a framework capable of generating synthetic time series data, achieving state-of-the-art performance when introduced in 2019 [66]. The approach proposed by the authors involves a unique combination of supervised and unsupervised training compressed in a latent space representation to reduce the dimensionality of the data simplifying the problem for adversarial training. The authors motivate the latent representation by the characteristics of time series data, where lower-dimensional variation factors significantly influence the temporal dynamics of complex systems. See Figure 2.11 for the architecture.

Therefore, the TimeGAN performs dimensionality reduction through an embedding function, mapping the high-dimensional input data to a lower-dimensional space, similar to the encoder in a VAE. The adversarial training utilises the reduced dimensional representation allowing the generator and discriminator network to operate and produce output in the "simpler" latent space. Furthermore, it is argued that a generator relying solely on the binary feedback from the discriminator in adversarial training may not be sufficient to capture the step-wise conditional distribution in time series data, despite being auto-regressive. Therefore, the authors introduce a supervised loss to specialise the generator to learn the conditional distributions. Finally, a recovery function transforms the embedded representation to its original form, much like the decoder in a VAE.

TimeGAN's training process unfolds in three key stages: pre-training the embedding and recovery functions, pre-training the supervisor function, and joint training of all networks (including the generator and discriminator).

1. The first stage focuses on learning the embedding and recovery functions. This step establishes connections between feature and latent spaces, enhancing the adversarial component's ability to grasp the data's temporal dynamics via lower-dimensional representations.

   More formally, let $\mathcal{X}$ be the real feature space, with $\mathbf{x}_t \in \mathcal{X}$ representing the real features at time t. Let $\mathcal{H}$ be the latent feature space, with $\mathbf{h}_t \in \mathcal{H}$ representing the latent features at time t. The embedding function, denoted as E, maps real features to latent features:

$$E : \mathcal{X} \to \mathcal{H}, \quad \mathbf{h}_t = E(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

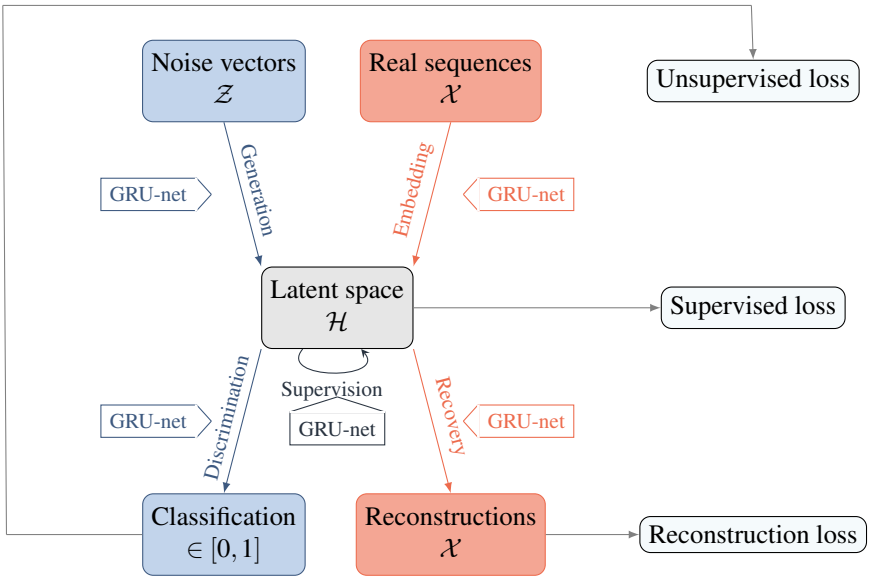


Figure 2.11 Block scheme of TimeGAN architecture.

implemented using a recurrent neural network (RNN) such as an LSTM or GRU. The recovery function, denoted as R, maps latent features back to real features:

$$R : \mathcal{H} \to \mathcal{X}, \quad \hat{\mathbf{x}}_t = R(\mathbf{h}_t)$$

implemented using another RNN. The goal in the first stage is to learn the embedding function E and the recovery function R by minimising the *reconstruction loss*:

$$\mathcal{L}_R = \mathbb{E}_{\mathbf{x}_{1:T} \sim p} \left[\sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2 \right]$$

2. In the second stage, the supervisor function is trained to generate subsequent sequences using the latent representation of real data for guidance.

   Again, let $\mathcal{H}$ be the latent space, with $\mathbf{h}_t \in \mathcal{H}$ representing the latent features up to time t and $\mathbf{h}_{t-1}$ representing the latent features up to time $t-1$. The Supervisor network, denoted as S, aims to learn the mapping to the subsequent features:

   $$S : \mathcal{H} \to \mathcal{H}, \quad \hat{\mathbf{h}}_t = S(\mathbf{h}_{t-1})$$

   The goal of the second stage is to train the supervisor function S using the latent representation of real data $\mathbf{h}_t$ as supervision and prepare it for its role

in the generative process, minimising the loss:

$$\min_{S} \sum_{t} \|\mathbf{h}_t - \hat{\mathbf{h}}_t\|_2$$

also implemented using an RNN.

3. The third and final stage involves joint training of all networks, culminating in adversarial training, concurrently minimising the *Unsupervised*, *Supervised*, and *Reconstruction losses* (Figure 2.11).

   Using the pre-trained embedding and recovery functions, it is now possible to reconstruct $\hat{\mathbf{x}}_{1:T}$ from the latent representation $\mathbf{h}_{1:T}$. Moreover, the supervisor function S is pre-trained on the mapping from the subsequent features to the next latent features. Let $\mathbf{z}_t \in \mathcal{Z}$ be the noise vector at time t. The generator G is responsible for generating the fake embedded features as follows:

$$G : \mathcal{Z} \times \mathcal{H} \to \mathcal{H}, \quad \hat{\mathbf{h}}_t = G(\mathbf{h}_{t-1}, \mathbf{z}_t)$$

   The generation process is two-fold. First, the generator produces unsupervised samples. Additionally, the generator is provided with the previous embedded features and produces supervised samples, allowing for a gradient to be computed on a loss capturing the discrepancy of the actual and estimated conditional distributions $p(\mathbf{H}|\mathbf{H_{t-1}})$ and $\hat{p}(\mathbf{H}|\mathbf{H}_{t-1})$ in a supervised fashion.

   The discriminator network D is tasked with distinguishing between real and fake samples for both the supervised and unsupervised generation process:

$$D : \mathcal{H} \to [0,1], \quad D(\mathbf{h}_t) = y_t, \quad D(\hat{\mathbf{h}}_t) = \hat{y}_t$$

   where y_t denotes the predicted labels for real features and $\hat{y}_t$ for fake features. Additionally, the embedded, recovery and supervisor functions are continually trained. Consequently, joint training involves minimizing the reconstruction, supervised and unsupervised loss:

$$\mathcal{L}_R = \mathbb{E}_{\mathbf{x}_{1:T} \sim p} \left[\sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2 \right]$$

$$\mathcal{L}_S = \mathbb{E}_{\mathbf{h}_{1:T} \sim p} \left[\sum_t \|\mathbf{h}_t - G(\mathbf{h}_{t-1}, \mathbf{z}_t)\|_2 \right]$$

$$\mathcal{L}_U = \mathbb{E}_{\mathbf{x}_{1:T} \sim p} \left[\sum_t \log y_t \right] + \mathbb{E}_{\hat{\mathbf{x}}_{1:T} \sim \hat{p}} \left[\sum_t \log(1 - \hat{y}_t) \right]$$

Now, let θ_e, θ_r, θ_g, θ_d denote the parameters of the embedding, recovery, generator, and discriminator networks respectively. The training procedure can be divided

into two main optimisation problems. The first problem involves training the components θ_e, and θ_r on the supervised and reconstruction loss

$$\min_{\theta_e,\theta_r}(\lambda \mathcal{L}_S + \mathcal{L}_R)$$

where $\lambda \geq 0$ is a balancing hyperparameter. Including $\mathcal{L}_S$ is important since it conditions the embedding process to facilitate learning of temporal relationships. Furthermore, the adversarial training is set up as:

$$\min_{\theta_g}(\eta \mathcal{L}_S + \max_{\theta_d} \mathcal{L}_U)$$

where $\eta \geq 0$ is a another balancing hyperparameter. In addition to the unsupervised min-max game of the adversarial training, the generator is trained to iterate across time via $\mathcal{L}_S$.

2.10 Evaluation Metrics for Synthetic Data Generation

With numerous methods to generate synthetic data, evaluation metrics naturally play a vital part in comparing methods. For example, they allow quantifying the quality of the data or assessing how well it represents the actual distribution and underlying process. Hence, evaluating metrics may significantly impact the result and model selection. However, several evaluation metrics exist, each with advantages and limitations. Therefore, it is common to divide evaluation metrics into three classes; visual, statistical and machine learning-based [8].

Note that there are other evaluation methods perhaps more commonly associated with GAN evaluation than those listed below, such as the Inception Score (IS) [51], the Fréchet Inception Distance (FID) [26], and the Perceptual Length (PL) [30]. However, these use pre-trained image classification networks, such as the *inception network*, to extract features from the generated images and compare them with the features of the real data. As a result, these tests are most appropriate for evaluating synthetic image data and may not be as applicable for evaluating other data types.

Visual evaluation

Visual inspection is necessary to assess the performance of a GAN, as it provides information that quantitative metrics can't cover by visually comparing distributions, cumulative sums, and column correlation [8]. For example, distribution plots provide a sanity check on whether the model generates statistical properties but does not reveal hidden relationships. Likewise, the cumulative sum of each column can visually indicate similarities between distributions and provide comprehension for categorical values but does not reveal any relationships between columns. Finally, comparing the synthetic and real data correlation tables can show if the generator correctly models the relationships between columns.

Statistical metrics

Statistical metrics compare the distribution of each numerical column in the synthetic data set to the original. One of the most commonly used tests for continuous columns is the Kolmogorov-Smirnov (KS) test. It is a non-parametric method for comparing two continuous distributions. It assesses the similarity between the original and synthetic distribution using a two-sample KS test and the empirical CDF. The metric measures the maximum distance between the observed and expected CDF values. The Chi-squared (CS) test for discrete columns compares real and synthetic values distributions, resulting in the *p-value*, i.e., the probability that the two samples come from the same distribution.

Additional statistical methods to assess the quality of synthetic samples include Principal Component Analysis (PCA) [9] and t-distributed Stochastic Neighbour Embedding (t-SNE) [57], dimensionality reduction techniques that visualise high-dimensional data in a lower-dimensional space. PCA identifies the principal components of the data, i.e., linear combinations of the original variables that explain the most variation in the data. In synthetic data quality assessment, PCA can be used to compare the distributions of the synthetic data to the original data to assess whether the synthetic data captures the same patterns and relationships as in the original data.

The t-SNE identifies clusters and subclusters in high-dimensional data. It maps each data point to a lower dimensional space, such that close points in the original data are close in the lower dimensional space and vice versa for faraway points. Contrary to PCA, t-SNE compares the clustering structure of the synthetic data and that of the original data. Hence, t-SNE depicts local relationships better, while PCA captures global patterns. Note that the dimensionality reduction following the PCA and t-SNE analysis allows for visual evaluation of the synthetic representation in 2D or 3D, depending on the number of components.

Machine learning-based metrics

Machine learning-based metrics are standard when evaluating the performance of a synthetic data generator. The concept revolves around generating data with equal efficacy as the original data. There are two common ways to evaluate the usefulness of synthetic data. One of the most popular metrics when evaluating GANs is the *detection metrics*. They attempt to measure the quality of the synthetic data by training a binary classifier to differentiate between them. For example, the *area under the ROC curve* (AUC) is a typical detection metric, which measures the classifier's ability to distinguish real from synthetic samples. In the context of time series data, in [66], they train an additional discriminator, akin to the one used in adversarial training, to distinguish between the real and synthetic samples. This discriminator employs an LSTM to capture the temporal dependency.

Another popular metric is the *machine learning efficacy*, which measures synthetic data's impact on a downstream machine learning model. In other words, the machine learning model should achieve similar performance when trained on synthetic data compared to original data and improve when trained on an extended set by combining the two. For example, the accuracy of a supervised learning model trained on synthetic data can be one measure of the machine learning efficacy [8]. For time series data, in [66], they train an "off the shelf" predictive model using an LSTM to measure the downstream effect of the synthetic data.

Although machine learning-based metrics are practical for assessing the usability of the data, it is important to challenge these metrics when evaluating the model. For instance, the detection metric depends on the choice of the classifier and is itself a model selection process. For example, high-dimensional data can pose similar challenges to the classifier [44] and the data generator. Thus, combining these metrics with visual and statistical evaluation is recommended [55].

3

Methodology

3.1 Literature Study

The literature study investigates the potential of GANs for generating synthetic data while preserving privacy. The research area, explicitly using machine learning models, is relatively immature; thus, the project's scope mandates an extensive literature review to achieve this objective. In particular, it covers the background of synthetic data and its applications in the Information and Communication Technology (ICT) sector, as well as various GAN architectures and synthetic data generators. The focus lies on understanding different synthetic data generators and GAN architectures, such as VAEs, Deep Convolutional GAN (DCGAN), Wasserstein GAN (WGAN), WGAN-GP, and Conditional GAN. Following are GANs designed explicitly for tabular data generation, including TableGAN, CTGAN, CopulaGAN, and ADSGAN, as well as time series data, including TimeGAN and evaluation metrics for all the above methods.

Several extensive surveys have been published comparing GAN frameworks for generating tabular data [8, 45, 64], and there are contributions investigating the privacy aspect of synthetic data from GANs and ways to control privacy levels. However, it is clear from the research that there is a significant difference between the State-of-the-art (SOTA) models for tabular data and time series data. The research on synthetic time series data is more immature and dispersed. Therefore, our contributions and experiments will focus on the time series methods to further shed light on the advantages and drawbacks of TimeGAN and possible improvements.

3.2 Experiment Setup

After thoroughly understanding the literature, the research proceeds with several experiments to implement different GAN models using the PyTorch framework from the Torch library. The choice of PyTorch for this research stems from its

inherent advantages in terms of interpretability and ease of use and has recently evolved into the preferred framework for research. Furthermore, it offers all the features this project aims to utilise, such as distributed data parallelism to scale the training process to multiple GPUs for faster training. For training, we use a virtual machine with 40 cores, 9 NVIDIA GeForce GTX 1080 GPUs and 251 GB of disk memory, allowing for large data volumes and cloud-based training, with Jupyter Lab as the IDE.

Data sets

The data sets for model development are selected to represent various data characteristics and complexities, ensuring a comprehensive examination of the models' performance. These data sets encompass various temporal and non-temporal data, periodic and aperiodic trends, and correlated and high-dimensional features. The diverse selection is crucial in assessing the GAN models' adaptability and performance across distinct data types, providing insights into their strengths and limitations. Furthermore, including smaller and larger data sets allows us to analyse the scalability of the models and evaluate their efficiency in handling different data volumes. Finally, the data sets are well suited for benchmarking against existing models, facilitating model development. Please see Table 3.1 for an overview.

In the context of this research, we have opted not to split the data into separate training and testing sets when training the GANs. The primary goal of implementing GANs in this study is to generate synthetic data that closely replicates the original data rather than predicting or classifying unseen instances. Consequently, concerns about overfitting are less relevant in this case, as our objective is to capture the data distribution as accurately as possible.

To demonstrate the generalisability of the final model, we use the same model setup across all data sets, encompassing diverse characteristics and complexities. This approach allows us to showcase the models' adaptability and performance in generating synthetic data from different sources without compromising the fidelity of the generated data to the original distributions. However, as a final test set, no model development is conducted on a data set from Ericsson to demonstrate model generalisability further. The Ericsson data consists of six features from a server at Ericsson: CPU utilisation, interrupts per second, load average over 15 minutes, load average over 5 minutes, load average over 1 minute and memory utilisation. To further evaluate the model, we also incorporate a data set with labelled outliers to assess to which extent it is possible to use the synthetic data to train an outlier detection model. The data used for this is the number of taxi passengers in New York, presented hourly.

Table 3.1 Overview of data sets.

Data set	# Features	# Observations	Characteristics
Tabular data			
MVN	10	1000	Multivariate Gaussian
Abalone [18]	8	4177	Mixed feature types and correlation
Time series data			
Sines	5	10000	Mixed periodicity
Stocks [23]	6	3685	Aperiodic, and highly correlated features
Energy [11]	26	19735	High dimension, correlation and periodicity
Ericsson [60]	6	1440	Correlation and periodicity
Taxi [32]	2	5160	Labelled outliers

Implementing existing models for tabular data

The initial experiments aim to better understand synthetic tabular data generation using existing GANs and some models' unique techniques. The experiments are primarily for learning and involve various GAN models, including WGAN for numerical and mixed numerical-categorical data, DCGAN with Wasserstein distance, Conditional GAN, and CTGAN on both simulated and Abalone data. All models are implemented from scratch using the PyTorch library and our framework for the multi-GPU set-up to get hands-on experience with coding the different GAN architectures. To understand and compare performance, the model evaluation involves comparing real and synthetic data based on:

- **Distribution across feature:** Histogram plots for real and synthetic data for each feature to assess the model's ability to replicate the data distribution.
- **Conditional distribution across categories:** Histogram plots of real and synthetic data for categories conditioned on a specific feature to accurately evaluate the model's capability to link categories with the associated feature.
- **Feature correlation:** Correlation matrices of real and synthetic data to evaluate the model's effectiveness in capturing the relationships between features.

Implementing existing models for time series data

Following this, we examine the TimeGAN for time series data. The TimeGAN requires an extensive translation from TensorFlow to PyTorch. Therefore, the initial experiments aim to replicate the result from the original TimeGAN paper with our

PyTorch model rather than the original TensorFlow version. Early experiments with the TimeGAN architecture reveal limitations when trained using simulated data. Important observations include:

- **Complex architecture:** The model involves many "moving parts" with multiple losses and minimisation problems to solve. In addition, the combination of embedded, supervised and unsupervised training makes the model difficult to interpret and debug.
- **Unstable training:** When tested on different simulated datasets, the model often exhibits mode collapse, making it impractical to use without modification on new data sets.
- **Replicate results:** It is difficult to replicate some of the results presented in the paper. For instance, some deviations between the suggested setup in the paper and the published code make comparing results tricky.

3.3 T2GAN

This section presents the T2GAN architecture. It builds upon the TimeGAN framework presented in Section 2.9 but also incorporates fundamental techniques from successful GANs and other SOTA machine learning architectures, such as the Transformer. We will detail the enhancements to the TimeGAN architecture, including the Wasserstein distance with gradient penalty, temporal convolutional layers, and the Transformer concepts of positional encoding, multi-head attention and layer normalisation. A full model overview is available in Appendix 6.1.

Key Concepts

Recall the fundamental principles of the TimeGAN architecture:

- **Embedding:** TimeGAN utilises an embedding function for dimensionality reduction, mapping high-dimensional input data to a lower-dimensional space, similar to a VAE's encoder.
- **Adversarial Training:** The generator and discriminator networks operate in this lower-dimensional latent space, allowing for simplified output production.
- **Supervised Loss:** To better capture step-wise conditional distributions in time series data, the TimeGAN introduces a supervised loss, specialising the generator in learning conditional distributions.

- **Recovery:** A recovery function transforms the embedded representation back to its original form, much like a VAE's decoder.

The T2GAN operates via the same fundamental principles but with a different architecture configuration, illustrated in Figure 3.1.

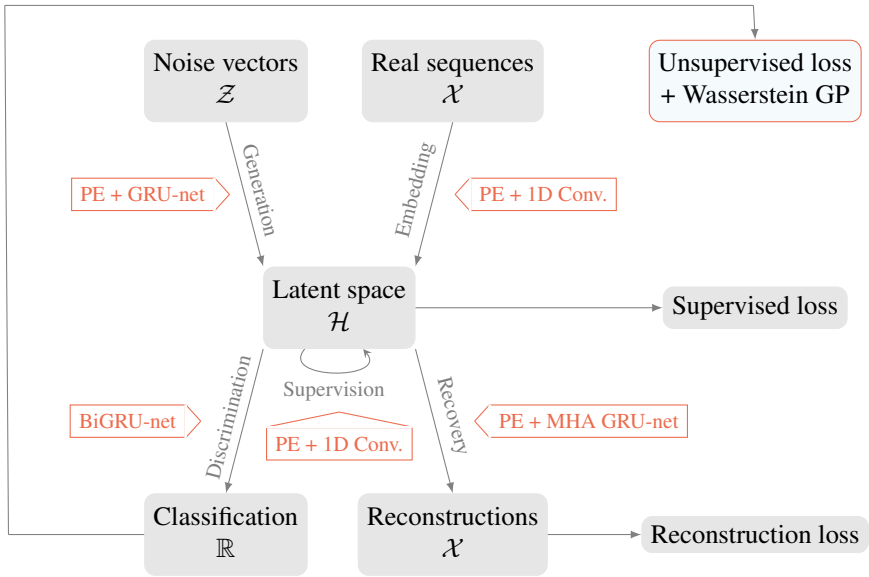


Figure 3.1 Block scheme of T2GAN architecture. PE stands for positional encoding, and MHA for multi-head attention. Grayed-out modules are unchanged compared to TimeGAN, and red modules are updated.

The aim is to develop improvements based on the most promising techniques revealed by the literature study in successful GANs and other SOTA generative machine learning architectures. The following list summarises the key improvements:

- **WGAN-GP:** Wasserstein distance with gradient penalty as loss function demonstrates improved model stability, preventing mode collapse.
- **Temporal Convolutions:** SOTA GANs for both tabular and time series data successfully employ convolutional elements to represent the data in a higher dimensional space to capture complex dynamics. In the time series setting, this is done via temporal 1D convolutions.
- **Positional Encoding:** Positional encoding can provide neural networks with additional information about the relative positions of data points in a se-

quence. By injecting this as input, the model is better equipped to recognise and preserve temporal relationships, leading to a more coherent representation of the generated time series data.

- **Multi-head attention:** Attention mechanisms have successfully modelled long-range dependencies and captured global context in sequence-to-sequence tasks, playing an instrumental role in the Transformer architecture. Utilising multi-head attention in decoder networks (such as the recovery function) allows them to learn diverse aspects of the input representation.

- **Layer normalisation:** Layer normalisation can stabilise training by operating independently on each instance in a batch by normalising along the feature dimension, in contrast to batch normalisation, which computes the mean and variance along the batch dimension. Adding layer normalisation aims to mitigate the issues of internal covariate shift, which can lead to slow convergence and improve the overall training stability.

A note on Transformers. The Transformer architecture has recently gained immense popularity and is the basis of the most complex sequential generative models, such as the GPT models. Thus, it is natural to consider the Transformer architecture when generating time series data. As introduced in [58], the typical Transformer consists of two parts: one encoder and one decoder, where the latter is trained in a supervised manner. For instance, replacing the embedding and recovery functions with a Transformer encoder-decoder architecture would be possible. However, due to the supervised and unsupervised training components in the TimeGAN, without a fairly extensive redesign, this approach would not be customised to be directly incorporated into the TimeGAN model.

Instead, we choose to utilise the critical properties of the Transformer directly, which are its positional encoding, multi-head attention mechanism and layer normalisation, into the existing architecture, keeping the overall TimeGAN framework intact.

Wasserstein distance with gradient penalty

In the original TimeGAN paper, the unsupervised loss is computed using Binary cross-entropy with logits (BCE) loss, i.e., combining a Sigmoid layer and the BCE Loss. The first improvement is incorporating the Wasserstein distance in the unsupervised loss and enforcing the necessary Lipschitz constraint with gradient penalty to ensure acceptable approximation. The Wasserstein distance with gradient penalty addresses some of the critical issues faced by traditional GANs, particularly regarding training stability, mode collapse and vanishing gradients. Based on the success of the WGAN-GP for tabular data, the TimeGAN model could benefit from incorporating the Wasserstein distance with gradient penalty

in its architecture, leading to more stable and robust training. This is done by replacing the existing BCE loss with the Wasserstein distance. However, we found a summation of the two losses added further stability, combining the best of both worlds. Recall the Wasserstein distance with gradient penalty:

$$\mathcal{L}_W = \underbrace{\mathbb{E}_{z\sim p(z)}[D(G(z))] - \mathbb{E}_{x\sim p(x)}[D(x)]}_{\text{Critic loss}} + \underbrace{\lambda \mathbb{E}_{\tilde{x}\sim p_{\tilde{x}}}[(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2]}_{\text{Gradient penalty } \mathcal{L}_{gp}}.$$

Now, to combine the BCE loss with the Wasserstein distance for the critic and the generator, we add the Wasserstein distance to the unsupervised loss:

$$\mathcal{L}_U = \mathbb{E}_{\mathbf{x}_{1:T}\sim p}\left[\sum_t \log y_t\right] + \mathbb{E}_{\hat{\mathbf{x}}_{1:T}\sim \hat{p}}\left[\sum_t \log(1-\hat{y}_t)\right] + \mathcal{L}_W$$

The gradient penalty term operates as follows:

1. A set of random samples $\tilde{\mathbf{x}}$ is generated by interpolating between real data samples $x \sim p(x)$ and generated data samples $G(z)$, where $z \sim p(z)$. This interpolation is done by drawing a random number α uniformly between 0 and 1 and then computing $\tilde{x} = \alpha x + (1-\alpha)G(z)$.
2. The gradient of the critic function $D(\tilde{x})$ with respect to the random samples $\tilde{x}$ is computed: $\nabla_{\tilde{x}} D(\tilde{x})$.
3. The gradient norm, denoted by $||\nabla_{\tilde{x}} D(\tilde{x})||_2$, is computed, measuring how large the gradients are for the random samples.
4. The gradient penalty term is calculated as the squared difference between the gradient norm and 1, $(||\nabla_{\tilde{x}} D(\tilde{x})||_2 - 1)^2$. This term penalises the critic function when the gradient norm differs from 1, which is the desired Lipschitz constraint.
5. The gradient penalty term is then incorporated into the critic's loss function with a penalty coefficient, denoted by λ (e.g., $\lambda = 10$): $\lambda \mathbb{E}_{\tilde{x}\sim p_{\tilde{x}}(\tilde{x})}[(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2]$, where $p_{\tilde{x}}(\tilde{x}) = \alpha p(x) + (1-\alpha)p(z)$.

These enhancements aim to better model the underlying data distribution and improve training stability.

Temporal convolution

As a next improvement, we opt for a 1D convolution architecture in the embedding function, replacing the GRU layers with 1D convolution. This idea originates from [43] and is further discussed in [66] as a potential embedding function. The choice is motivated by the ability of 1D convolutions to capture temporal dependencies

in time series data and resonates well with the success of convolutional layers in Tabular GANs. Additionally, we find replacing the GRU layers in the supervisor function with 1D convolutions to yield significantly better results. However, the recovery, generator and discriminator networks still operate best with GRU layers as the autoregressive element.

1D convolution operates by sliding the kernel along the time axis of the input data, computing the dot product between the filter weights and the input at each position. The output of a layer with input size (N, C_{in}, L_{in}) is defined as:

$$\text{out}(N, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N, k)$$

where $\star$ is the valid cross-correlation operator, N is the batch size, C denotes the number of channels, L_{in} is the length of the signal sequence. We use a kernel of size 5, stride of 1 and dilation of 1. We then choose padding such that $L_{out} = L_{in}$ to keep a consistent sequence length:

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel size} - 1)}{\text{stride}} \right\rfloor + 1.$$

The 1D convolutions aim to better detect various features and short-term patterns in the data by learning appropriate filter weights, producing a well-suited latent space for adversarial training. Moreover, aligned with the TimeGAN approach, we use a deep architecture, stacking multiple 1D convolution layers to enable the model to learn hierarchical features, where higher layers capture more abstract and complex temporal structures. This hierarchical feature learning ability is crucial in modelling the intricate dependencies in time series data. Additionally, adding batch normalisation and ReLU activation functions between the layers contributes to increased stability during training and encourages the model to learn richer feature representations.

Positional encoding

In the T2GAN architecture, we incorporate a fixed positional encoding as an additional input to enhance the model's awareness of sequential information. In the Transformer architecture, positional encoding was necessary due to the absence of sequential information in the self-attention mechanism. However, in the case of T2GAN, the addition of positional encoding serves a different purpose.

Time series data in the TimeGAN architecture is processed using a sliding window approach, which generates sequences of a fixed length at each step. However, this approach results in significant overlap between successive sequences. To illustrate, if we slide a window of size 24 ahead by a single unit, there will be 23 identical

data points out of 24 between two adjacent sequences. On the other hand, using a larger step size drastically reduces the number of observations, which is not ideal.

This inherent overlap introduces a potential pitfall in model training. The recurrent similarity in the data could prompt the model to overemphasise recurring features since the same patterns appear many times across different sequences, leading to learning redundant information and potentially missing unique patterns.

In T2GAN, we address this by giving each sequence a positional encoding. The idea is that the positional encoding act as a regulariser, mitigating the potential problem with overlapping sequences and allowing the model to learn a more diverse set of features.

Multi-head attention-based

Introduced in [6] and further discussed in [66], a multi-head attention mechanism in the recovery function could improve the recovery of complex temporal dependencies. Analogous to the Transformer, the multi-head attention in the recovery function operates by projecting the input sequence into query, key, and value matrices using the different linear transformations:

$$\begin{aligned} Q &= XW_Q, \\ K &= XW_K, \\ V &= XW_V, \end{aligned}$$

where X is the input data, and W_Q, W_K, and W_V are learnable weight matrices. Next, the compatibility scores between the query and key pairs are computed via the scaled dot-product for each head to weight the value vectors, effectively capturing the relevant context from the input sequence:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the key vectors. The multi-head attention mechanism computes the attention function multiple times in parallel (with different learned linear projections) to allow the model to learn diverse aspects of the input representation. The individual attention outputs are then concatenated and projected using a linear transformation:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O$$

where h is the number of attention heads, and W^O is a learnable weight matrix.

Layer normalisation

As a successful normalisation technique in RNNs [5], and other sequence-based methods such as the Transformer [58], normalising the input across the feature

dimension to have zero mean and unit variance speeds up and stabilises training. In T2GAN, we apply layer normalisation operation in all networks, see equation 2.1.

3.4 Model Training

Training involves parameterising the combination of embedding, recovery, generator, and discriminator networks. First, we create N sequences of time series data of length T and dimension d, resulting in data set $\mathcal{D}$, using a sliding window of length $T = 24$ and step size 1.

During training, the embedding network first learns the mapping from feature space $\mathcal{X}$ and latent space $\mathcal{H}$. Consequently, the recovery network learns to reverse this mapping. Next, the generator creates synthetic latent codes using random noise sampled from a Wiener process $\mathcal{Z}$. The discriminator then tries to distinguish between real latent codes obtained from the embedding function and synthetic latent codes generated by the generator.

A gradient penalty is computed and added to the Wasserstein distance to ensure Lipschitz continuity via a combination of real and synthetic latent codes. Again, the resulting losses are the reconstruction loss for data recovery, the unsupervised loss for discrimination between real and synthetic latent codes, the supervised loss for realistic latent code generation and the gradient penalty loss for discriminator smoothness.

We employ the same optimisation process as described in Section 2.9, [66]. Let θ_e, θ_r, θ_g, θ_d denote the parameters of the embedding, recovery, generator, and discriminator networks respectively. The training procedure is divided into two main optimisation problems. The first problem involves training the components θ_e, and θ_r on the supervised and reconstruction loss

$$\min_{\theta_e,\theta_r}(\mathcal{L}_S + \mathcal{L}_R).$$

Furthermore, the adversarial training is set up as follows:

$$\min_{\theta_g}(\mathcal{L}_S + \max_{\theta_d}[\mathcal{L}_U + \mathcal{L}_{gp}])$$

In addition to the unsupervised min-max game of the adversarial training, the generator is trained to iterate across time via $\mathcal{L}_S$. The training procedure alternates between these two optimisation problems, allowing the generator to learn temporal dynamics from the supervised loss while refining its output based on the discriminator's feedback through adversarial training. The pseudo-code for training is outlined in Algorithm 1.

Algorithm 1 Pseudo code for the T2GAN algorithm

Input: $\lambda = 10, \mathcal{D}$, batch size n_{mb}, learning rate γ=0.001
Initialize: $\theta_e, \theta_r, \theta_g, \theta_d$
while Not converged **do**
    (1) Map between Feature Space and Latent Space
    Sample $\mathbf{x}_{1,1:T_n}, \ldots, \mathbf{x}_{n_{mb},1:T_{n_{mb}}} \overset{\text{i.i.d}}{\sim} \mathcal{D}$
    for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
        $\mathbf{h}_{n,t} = e(\mathbf{h}_{n,t-1}, \mathbf{x}_{n,t})$
        $\tilde{\mathbf{x}}_{n,t} = r(\mathbf{h}_{n,t})$

    (2) Generate Synthetic Latent Codes
    Sample $\mathbf{z}_{1,1:T_n}, \ldots, \mathbf{z}_{n_{mb},1:T_{n_{mb}}} \overset{\text{i.i.d.}}{\sim} \mathcal{Z}$
    for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
        $\hat{\mathbf{h}}_{n,t} = g(\hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t})$

    (3) Distinguish between Real and Synthetic Codes
    for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
        $y_{n,t} = d(\overleftarrow{\mathbf{h}}_{n,t}, \overrightarrow{\mathbf{h}}_{n,t})$
        $\hat{y}_{n,t} = d(\overleftarrow{\hat{\mathbf{h}}}_{n,t}, \overrightarrow{\hat{\mathbf{h}}}_{n,t})$

    (4) Compute Gradient Penalty
    for $n = 1, \ldots, n_{mb}, t = 1, \ldots, T_n$ **do**
        Sample $\alpha \sim U[0,1]$
        $\tilde{\mathbf{h}}_{n,t} = \alpha \hat{\mathbf{h}}_{n,t} + (1-\alpha)\mathbf{h}_{n,t}$
        $\mathbf{gp}_{n,t} = (\|\nabla_{\tilde{\mathbf{h}}_{n,t}} d(\tilde{\mathbf{h}})\|_2 - 1)^2$

    (5) Compute Reconstruction, Unsupervised, Supervised, and Gradient Penalty Losses
    $\mathcal{L}_R = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \|\mathbf{x}_{n,t} - \tilde{\mathbf{x}}_{n,t}\|_2 \right]$
    $\mathcal{L}_U = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[[\sum_t \log y_{n,t}] + [\sum_t \log(1 - \hat{y}_{n,t})] + [\frac{1}{t} \sum_t y_{n,t}] + [\frac{1}{t} \sum_t \hat{y}_{n,t}] \right]$
    $\mathcal{L}_S = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\sum_t \left\| \mathbf{h}_{n,t} - g(\mathbf{h}_{n,t-1}, \mathbf{z}_{n,t}) \right\|_2 \right]$
    $\mathcal{L}_{gp} = \frac{1}{n_{mb}} \sum_{n=1}^{n_{mb}} \left[\frac{1}{t} \sum_t \left\| \mathbf{gp}_{n,t} \right\|_2 \right]$

    (6) Update $\theta_e, \theta_r, \theta_g, \theta_d$ via Stochastic Gradient Descent (SGD)
    $\theta_e = \theta_e - \gamma \nabla_{\theta_e} - [\mathcal{L}_S + \mathcal{L}_R]$
    $\theta_r = \theta_r - \gamma \nabla_{\theta_r} - [\mathcal{L}_S + \mathcal{L}_R]$
    $\theta_g = \theta_g - \gamma \nabla_{\theta_g} - [\mathcal{L}_S + \mathcal{L}_U]$
    $\theta_d = \theta_d + \gamma \nabla_{\theta_d} - [\mathcal{L}_U + \lambda \mathcal{L}_{gp}]$
end while

(7) Synthetic Data Generation
(7.1) *Sample* $\mathbf{z}_{1,1:T_n}, \ldots . \mathbf{z}_{N,1:T_N} \overset{\text{i.i.d.}}{\sim} \mathcal{Z}$

(7.2) *Generate synthetic latent codes*
for $n = 1, \ldots, N, t = 1, \ldots, T_n$ **do**
    $\hat{\mathbf{h}}_{n,t} = g(\hat{\mathbf{h}}_{n,t-1}, \mathbf{z}_{n,t})$

(7.3) *Mapping to the feature space*
for $n = 1, \ldots, N, t = 1, \ldots, T_n$ **do**
    $\hat{\mathbf{x}}_{1:T_n} = r(\mathbf{h}_{n,t})$

Output: $\hat{\mathcal{D}} = \{\hat{\mathbf{x}}_{1:T_n}\}_{n=1}^{N}$

After training, the generator creates synthetic latent codes by sampling from the latent space distribution. These synthetic latent codes are then mapped to the feature space using the recovery network, producing the final synthetic time series data. The output of the T2GAN algorithm is a set of realistic synthetic time series data.

3.5 Model Evaluation

To evaluate the performance of T2GAN against the original version, we establish a comparable model evaluation process with nuanced metrics that offer deeper insights into the model's proficiency.

- **PCA and t-SNE:** In the first visual test, we employ PCA and t-SNE as dimensionality reduction techniques to visualise and contrast the model's ability to capture local and global data patterns effectively. This approach helps visually assess the generated data's quality and compare it with TimeGAN.
- **Feature correlation:** The second visual test involves presenting the correlation matrices of real and synthetic data to gauge the model's capability to accurately capture intricate relationships between features. This test highlights any discrepancies in the correlations between the original and generated data sets.
- **Discriminative score:** The third test is a detection metric to benchmark T2GAN against TimeGAN. The metric measure how well an off-the-shelf LSTM classifier can distinguish between real and synthetic samples. This is measured using the classification error and is presented as a score $|accuracy - 0.5|$, where a score of 0 implies the samples are indistinguishable.
- **Predictive score:** The fourth test is a machine-learning efficacy metric to test the synthetic data in a downstream prediction task and benchmark T2GAN against TimeGAN. This metric measures the synthetic data's usefulness for a prediction task using an off-the-shelf LSTM. This is measured using the mean absolute error (MAE), where a score of 0 implies perfect prediction.
- **Outlier detection:** The final test assesses the synthetic data's usefulness in training an outlier detection method, Isolation Forest. Using the anomaly labelled Taxi data, T2GAN generates a synthetic version of a clean segment without outliers. Then, two distinct Isolation Forest models fit the data using the original and synthetic renditions of the clean data. Finally, the two Isolation Forest models predict outliers in a labelled test set of original data

containing multiple outliers. In the Taxi data, examples of anomaly events are New York City Marathon, snow storms or other special occasions affecting the taxi travel pattern. The difference in performance indicates T2GAN's effectiveness for a downstream anomaly detection task.

4

Results

4.1 Experiments on Time Series Data

Tab. 4.1 presents the discriminative and predictive scores for the T2GAN and TimeGAN models across various data sets (Sines, Stocks, Energy, and Ericsson). The T2GAN consistently outperforms TimeGAN regarding discriminative scores. Furthermore, the predictive scores for both models are similar on Stocks, while the T2GAN achieves a lower predictive score on the Energy data set. On average, the T2GAN demonstrates a 38% performance increase in discriminative scores and a reduction of relative prediction error by a factor of 2.21 when using synthetic training data. The predictive score when using original data for training is also displayed, in particular, note that for three of the four data sets, the T2GAN achieves $93-98\%$ efficacy.

Table 4.1 Discriminative and Predictive score on different data sets (Bold indicates best performance).

Metric	Model	Sines	Stocks	Energy	Ericsson
Discriminative	T2GAN	$\mathbf{.290 \pm .103}$	$\mathbf{.076 \pm .034}$	$\mathbf{.327 \pm .023}$	$\mathbf{.241 \pm .014}$
Score	TimeGAN	$.492 \pm .005$	$.147 \pm .025$	$.451 \pm .011$	$.368 \pm .039$
(Lower the better)					
Predictive	T2GAN	$\mathbf{.326 \pm .002}$	$\mathbf{.039 \pm .000}$	$\mathbf{.024 \pm .002}$	$\mathbf{.110 \pm .002}$
Score	TimeGAN	$.387 \pm .004$	$\mathbf{.039 \pm .000}$	$.040 \pm .002$	$.135 \pm .004$
(Lower the better)	Original Data	$.318 \pm .000$	$.037 \pm .000$	$.009 \pm .000$	$.102 \pm .001$

Table 4.2 summarises the results of the outlier detection analysis on the Taxi data using Isolation Forest. The metrics considered in this evaluation include precision, recall, and F1-score. Comparing the original data to the synthetic data generated by T2GAN, we observe that the synthetic data achieve slightly lower scores than the original data but still reach a ~85% efficacy compared to the original for outlier prediction tasks.

Table 4.2 Outlier detection metrics using Isolation Forest. The values are presented as mean ± 95% confidence intervals.

Metric	Original Data	T2GAN
Precision	**.595 ± .034**	.474 ± .049
Recall	**.483 ± .026**	.429 ± .033
F1-score	**.532 ± .004**	.448 ± .023

Figure 4.1 showcases the PCA visualisations of original and synthetic data generated by T2GAN and TimeGAN models for the Sines, Stocks, Energy, and Ericsson data sets. The first and second columns depict T2GAN, whereas the third and fourth columns represent TimeGAN. Blue markers denote original data, and orange markers indicate synthetic data. In general, T2GAN captures the underlying structure of the original data more closely, especially for the Stocks, Energy and Ericsson data sets. In particular, T2GAN has a more consistent and precise overlap between the original and synthetic data sets using PCA and t-SNE.

Figure 4.2 showcases the t-SNE visualisations of original and synthetic data generated by the T2GAN and TimeGAN models for the Sines, Stocks, Energy, and Ericsson data sets. The first column depicts T2GAN, whereas the second column represents TimeGAN. Blue markers denote original data, and orange markers indicate synthetic data. Note that T2GAN generates data with notably better overlap with the original data in the t-SNE visualisation, particularly for the Sines and Stocks data sets. Therefore, T2GAN captures the local structure of the original data more closely across all data sets.

Figure 4.3 displays the correlation matrices for the original and synthetic data across the Sines, Stocks, Energy, and Ericsson data sets. The first column is the correlation matrix of the actual data. The next two columns represent the correlation for synthetic data from T2GAN and TimeGAN. T2GAN produces almost identical correlation matrices to the original data. While both models synthesise good correlation structures, T2GAN consistently outperforms TimeGAN, especially for the Sines data, with almost identical correlation matrices.

We present a source-of-gain analysis to understand the contributions of each improvement introduced in the T2GAN model. This analysis evaluates the model's performance with and without specific components: WGAN-GP, temporal convolution, multi-head attention and positional encoding using the discriminative and predictive scores for evaluation. The results of the analysis are presented in Table 4.3.

Table 4.3 Source-of-Gain Analysis (via Discriminative and Predictive scores). MHA stands for multi-head attention, and PE for positional encoding.

Metric	Method	Sines	Stocks	Energy	Ericsson
Discriminative Score (Lower the better)	T2GAN	**.290 ± .103**	**.076 ± .034**	.327 ± .023	**.241 ± .014**
	w/o WGAN-GP	.295 ± .081	.146 ± .030	**.314 ± .037**	.366 ± .013
	w/o Temp. Conv.	.408 ± .041	.227 ± .025	.393 ± .004	.319 ± .013
	w/o MHA + PE	.494 ± .010	.298 ± .038	.486 ± .007	.309 ± .054
Predictive Score (Lower the better)	T2GAN	**.326 ± .002**	**.039 ± .000**	**.024 ± .002**	**.110 ± .002**
	w/o WGAN-GP	.349 ± .001	**.039 ± .000**	.025 ± .001	.115 ± .006
	w/o Temp. Conv.	.370 ± .001	.069 ± .003	.029 ± .001	.167 ± .004
	w/o MHA + PE	.348 ± .005	.044 ± .002	.074 ± .006	.131 ± .008

4.2 Sources of Gain

Table 4.3 shows that T2GAN performs best across all data sets. It is worth noting that the inclusion of the different components all contributes significantly to increased performance. Removing the WGAN-GP component results in slightly worse performance in the Sines, Stocks, and Ericsson data sets. Furthermore, the absence of temporal convolution significantly decreases performance across all data sets, indicating the importance of this component in capturing the conditional distribution. Finally, the model without the Transformer components shows the most substantial decline in performance for all data sets, emphasising the critical role of the multi-head attention and positional encoding mechanism in generating realistic synthetic data. Furthermore, we observe that the additions in T2GAN predominantly influence the discriminative scores, with smaller changes to the predictive scores.

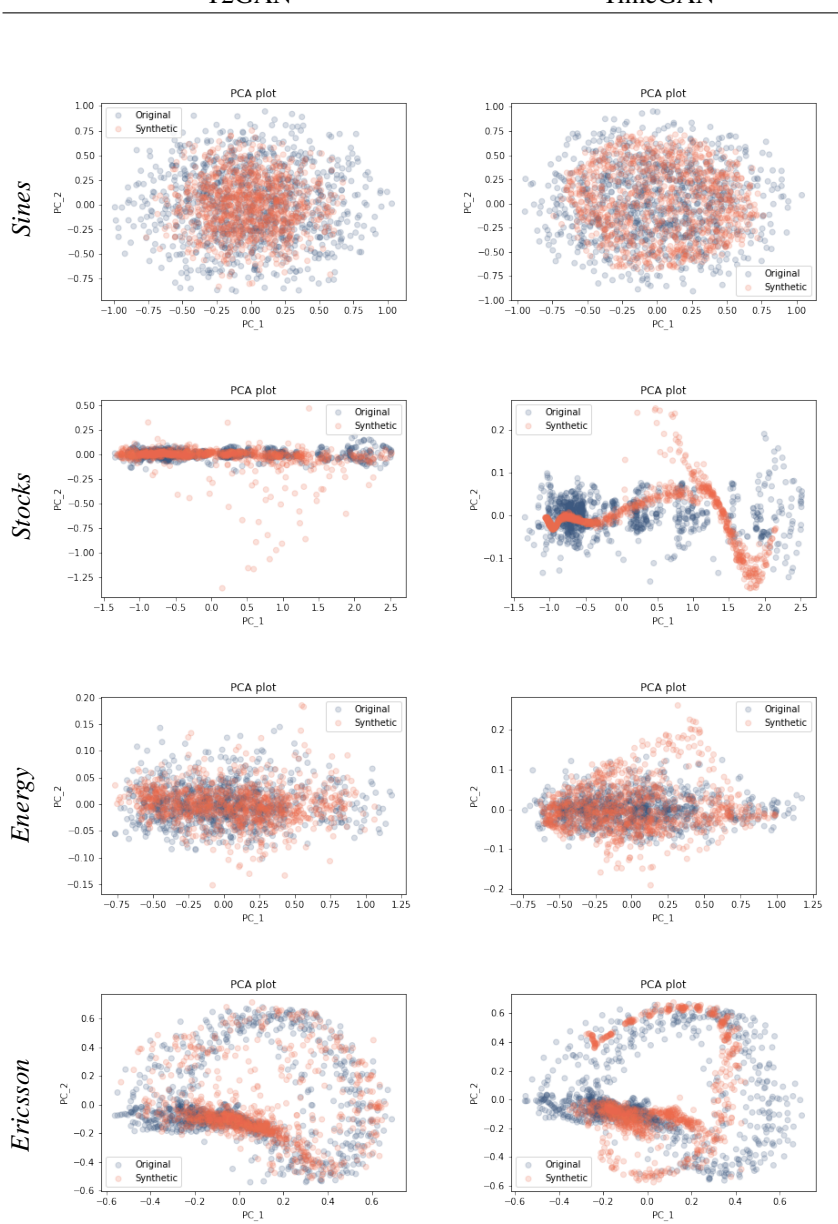


Figure 4.1 PCA visualisations for Sines, Stocks, Energy, and Ericsson data sets in rows. Columns represent models: 1st for T2GAN, 2nd for TimeGAN. Blue represents original data, and red represents synthetic data. For ideal synthetic data, a high degree of overlap between the blue and red markers is desired.

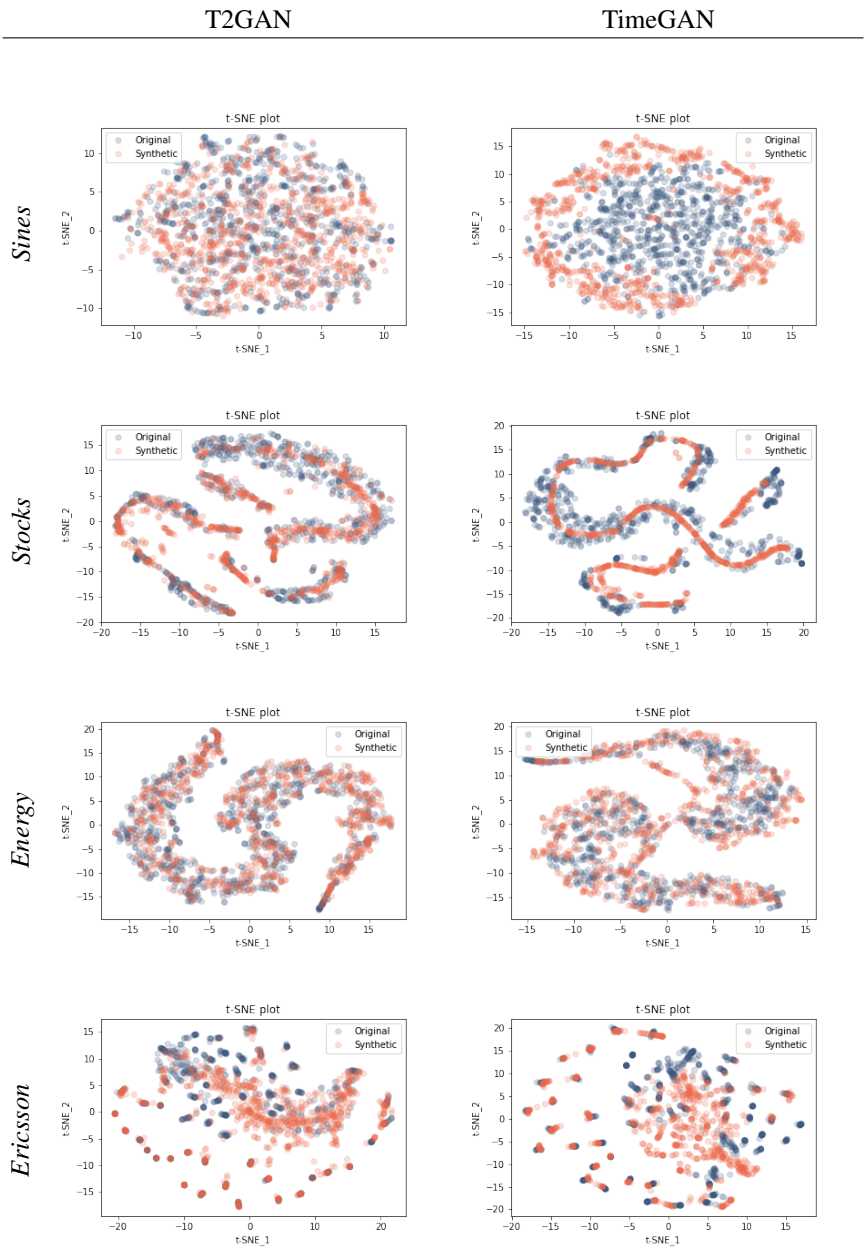


Figure 4.2 t-SNE visualisations for Sines, Stocks, Energy, and Ericsson data sets in rows. Columns represent models: 1st for T2GAN and 2nd for TimeGAN. Blue represents original data, and red represents synthetic data. For ideal synthetic data, a high degree of overlap between the blue and red markers is desired.

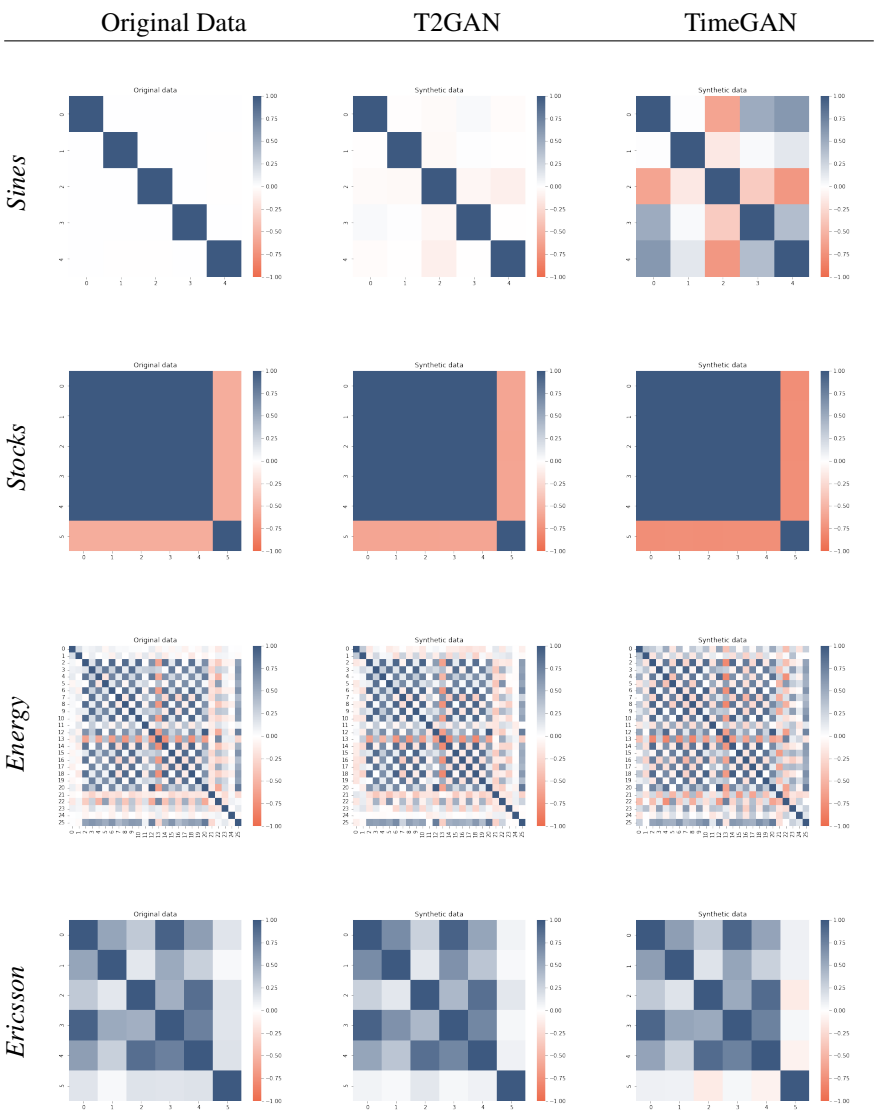


Figure 4.3 Correlation matrix with rows representing Sines, Stocks, Energy, and Ericsson datasets. Columns show the computed correlation between original and synthetic data from T2GAN and TimeGAN.

5

Conclusion

In the context of synthetic tabular data, the literature study reveals that the existing methods, presented in Section 2.8, perform remarkably well for many types of tabular data sets. In addition, these methods also address common issues, such as mode collapse, class imbalance and training instability. However, while these methods are successful, it is also clear that moving on to time series data requires additional tools. The nature of the time series data poses unique problems with the added temporal dimension.

There are several successful techniques from tabular GANs that this thesis has shown successful for time series generation, such as the Wasserstein distance, convolution and Transformer architectures. Furthermore, there are several promising techniques this thesis does not explore further in the time series domain, such as

- **Mode-specific normalisation**: to combat multimodal distributions.
- **Incorporating a classifier**: enhancing semantics within samples.
- **Conditional generator**: allowing the generation of well-dispersed samples even from imbalanced data sets.

It is interesting how well these translate to time series data, especially considering data with categorical, textual or static features. T2GAN is already well-equipped to handle categorical inputs via one-hot encoding. However, the above concepts could become essential if applied to high dimensional or unbalanced data, such as log files where textual data must be converted to classes. Moreover, the model framework translates to static features; however, for a full breakdown of the mathematical definitions with static features, we refer to the original TimeGAN paper [66].

5.1 Performance Evaluation of T2GAN

The following discussion highlights the implications of the improvements made to the TimeGAN architecture and their impact on the model's performance.

Wasserstein distance with gradient penalty

The first objective is to enhance training stability by incorporating the Wasserstein distance with gradient penalty into the T2GAN model. Replacing the Binary cross-entropy loss in the adversarial training with the Wasserstein distance should mitigate mode collapse and increase training stability.

The experiments reveal increasing performance when adding the Wasserstein distance to the loss. The model delivers the best results when employing the Wasserstein distance in combination with the BCE loss. The takeaway is that the Wasserstein distance with gradient penalty helps mitigate vanishing gradients compared to only using the BCE loss, thus stabilising training and avoiding mode collapse.

Temporal convolutions

The second improvement introduced to the TimeGAN model is incorporating temporal convolutional layers. The rationale behind this addition lies in the complementary strengths of GRUs and temporal convolutions. While GRUs excel at capturing long-range dependencies due to their ability to update hidden states and remember or forget information over extended periods, temporal convolutions may be better suited for encoding tasks focusing on short-term patterns through its sliding kernels. By integrating temporal convolutions in the embedding function, the TimeGAN model can more effectively capture complex dynamics in time series data and translate them into the latent space. In addition, this approach aligns with state-of-the-art neural networks in tabular and time series domains, which have successfully employed convolutional elements to enhance their performance.

Upon analysing the results of the T2GAN model with the temporal convolutional layers in the embedding function, there is a significant performance increase, suggesting the 1D convolution is suitable to capture the underlying data structure. Furthermore, this motivates testing the 1D convolution in the other "encoder-like" functions, such as in the generative process. Employing the 1D convolution in the supervisor function, i.e., in the supervised generation process, the performance increases further. However, the original GRU network yields the best performance of the unsupervised generator network.

The takeaway is that 1D convolutions capture and encode the latent space's temporal dependencies and cross-sectional dynamics. However, they are not suited to the unsupervised generation process, where we must generate the entire sequence simultaneously, meaning that some longer-term representation is necessary. On the

other hand, the recurrent nets excel at capturing long-range dependencies in time series data, making them particularly suitable for unsupervised generative tasks, where the ambition is to generate the entire sequence at once. Then, the 1D convolutional supervisor function adds further short-term structure by operating on the unsupervised samples. This synergy between the two components allows for more accurate and diverse synthetic time series generation.

Transformer components

Finally, incorporating Transformer components, such as positional encoding, multi-head attention, and layer normalisation, constitute the final improvements in the T2GAN model. Layer normalisation is known to enhance the performance of recurrent networks, and we show that positional encoding improves performance for all networks (excluding the discriminator). The multi-head attention with positional encoding mechanism is the most significant enhancement among these three components. It is one of the primary bottlenecks of the previous model and significantly increases performance when included in the "decoding" recovery function. More specifically, the multi-head attention mechanism contributes to increased realism by accurately reconstructing the generated samples from the latent space, resulting in superior performance across all evaluation metrics. Furthermore, positional encoding demonstrates its effectiveness in regularising overlapping sequences. Even though all networks already include auto-regressive components and should not have to rely strictly on these additions, we demonstrate that by adding the additional information of positional encoding and attention layers, T2GAN can better navigate the complex time series domain.

Evaluation metrics

Interpreting the evaluation metrics for the T2GAN model requires understanding that metrics, such as the Discriminative and Predictive scores, are highly dependent on the data sets. Therefore, it is not feasible to directly compare the values across data sets. Instead, in this section, we discuss model evaluation's general challenges and limitations and provide specific examples and findings related to each data set to help interpret the evaluation metrics better.

The model is evaluated with respect to the given data set. For instance, in the case of sinusoids, the model achieves a rather disappointing discriminative score of 0.290, meaning the classifier achieves 79% accuracy (50% would imply it is guessing). Intuitively, generating sinusoids should be a "simple" problem. However, this also implies that the classification task is relatively easy, as minor discrepancies between the real and synthetic data are very noticeable. The challenge when evaluating each data set lies in understanding this balance.

Similarly, the model struggles to generate perfect values for data sets containing constant or partially constant features, such as in the Energy data. Instead, it tends

to add some noise (although very small), which the classifier can potentially detect fairly easily. This behaviour is not necessarily a bug but rather a feature, as the TimeGAN is not designed to reconstruct real signals from noise perfectly. For example, Stock data is notoriously difficult to model, and the results of the Stock data resonate well with these observations. In this case, the TimeGAN achieves a low discriminative score, and the predictive performance for the LSTM trained on real and synthetic data is very similar.

Furthermore, the outlier detection test using the Isolation Forest algorithm provides insight into the performance of the T2GAN model, particularly in generating synthetic data that captures the original data's outlier behaviour while maintaining overall data structure and patterns. Interestingly, the model trained on synthetic data has higher recall but lower precision than the original data. These results indicate that the synthetic data generated by the T2GAN model produces a smoother representation of class boundaries between normal and anomalous instances. Consequently, the model trained on synthetic data is better equipped to detect outliers near the decision boundary, yielding a higher recall, but simultaneously becomes oversensitive, resulting in too many predicted outliers and a lower precision.

One potential concern with the Taxi outlier data set is its distinct periodicity, which may have implications for the synthetic data and outlier test. For example, as stated, T2GAN is trained using sequences of length $T = 24$ and may struggle to reconstruct longer-term periodicity in the taxi data, which has a clear period of 24 hours. Therefore, if the synthetic data does not accurately reflect the periodicity or other temporal patterns, the outlier patterns may be diluted, leading to suboptimal performance in the outlier detection task. Specifically, the Isolation Forest algorithm relies on the underlying data distribution to distinguish between normal and anomalous points. If the periodicity is poorly reflected, it could explain the increased sensitivity to outliers, leading to false-positives detections and reduced precision.

Despite these challenges and limitations, T2GAN can generate highly realistic time-series samples. This is evidenced by the highly similar PCA and t-SNE visualisations in Figure 4.1 of real and synthetic data across all data sets, indicating that the model effectively captures the underlying patterns and dynamics of the data.

Privacy preservation in T2GAN

In the context of time series data, privacy preservation may be crucial for ensuring the confidentiality of sensitive information, making them inaccessible or difficult to access during model development. In such cases, anonymisation through a generative model could offer unique opportunities.

In time series data, privacy can be viewed from two perspectives: protecting the

underlying signal and preserving the individual sample's privacy. This section discusses the privacy-preserving ability of T2GAN and explores potential approaches to ensure privacy in time series data generation.

When considering time series data, the privacy of the underlying signal refers to preserving the overall patterns, trends, or dynamics within the data without revealing sensitive information about the specific data source. In contrast, individual sample privacy focuses on protecting the privacy of each sample sequence within the time series (e.g., a patient record) or the points within the sequences, ensuring that the generated synthetic data does not directly expose any sensitive information about the individual or entities that the data represents.

For the underlying signal, the TimeGAN model generates synthetic data that effectively captures the patterns and dynamics without directly copying the input data, providing some level of privacy. However, this method may not be well suited if the signals are especially sensitive. Although additional measures can be taken to protect privacy while maintaining data utility, such as distorting the signal in a controlled manner similar to the noise distortion in the ADSGAN [65], this scenario prohibits most modelling tasks by design since modelling a signal ultimately reveals its properties.

In the case of individual sample privacy, the TimeGAN model generates synthetic data that is not a direct copy of the original data points. However, to ensure and define private samples, it is essential to establish a privacy metric or utilise a privacy-preserving framework such as differential privacy or ε-identifiability defined in Definition 2.8.1 [15, 65]. By incorporating differential privacy constraints into the generative model, it should be possible to control data utility and privacy preservation, even for time series. Another approach is to modify T2GAN to hedge against specific adversarial attacks. For instance, if certain outliers are sensitive, the model can be tuned to avoid reconstructing such sensitive elements.

Ultimately, depending on the application, there may be simpler methods of protecting the private elements of time series data. We acknowledge that fully understanding the privacy implications of synthetic data generated by T2GAN requires further research and discussions regarding specific use cases. Instead, this thesis demonstrates that T2GAN can generate synthetic data completely from noise without seeing the original data and is a first step towards private synthetic time series.

5.2 Findings

This study explores various GAN frameworks for synthetic data generation, enhances the TimeGAN model and discusses the implications for synthetic time series generation. The following points summarise the key findings and contributions

of our research:

1. Existing methods for synthetic tabular data perform well for many data types and address common issues such as mode collapse, class imbalance, and training instability. However, these methods require additional tools when applied to time series data due to the temporal dimension. Techniques from tabular GANs, such as mode-specific normalisation, incorporating a classifier, and a conditional generator, have not been explored further in the time series domain. However, these techniques have the potential for temporal data, especially when dealing with categories or static features, and will likely play an essential role in extending the TimeGAN to other sequential data, such as log files.

2. TimeGAN's design intentionally divides the raw time series into more minor, i.i.d, subsequences, to facilitate better learning of underlying patterns and dependencies in the data while modelling time dependencies. However, this approach limits the output to the 3D format where sequence samples lack order. As a result, the current model does not support the complete modelling and synthesis of time series with very long sequences (e.g., inputting a 1000x6 Stock data set, training with sequence length $T = 1000$, and outputting a synthetic 1000x6 series since it would imply that the model only trains on 1 sample). Nevertheless, even though trained on shorter sequences, the generator is technically capable of outputting longer sequences simply by adjusting the shape of the input noise. Our observations are that the output sequence only preserves temporal dynamics up to the sequence length used for training. If the data contains long-term dynamics such as periodicity or trends, they may not be present in the synthetic samples.

3. Incorporating the Wasserstein distance with gradient penalty into the T2GAN model enhances training stability. It mitigates mode collapse but does not significantly increase performance for the evaluation metrics.

4. Temporal convolutional layers in the T2GAN model effectively capture and encode temporal dependencies and cross-sectional dynamics in the latent space. The unsupervised generator, however, requires recurrent nets, which excel at capturing long-range dependencies in time series data.

5. Transformer components, such as positional encoding, multi-head attention, and layer normalisation, significantly improve the performance of the T2GAN model. The multi-head attention mechanism, in particular, is crucial in accurately reconstructing generated samples.

6. The model evaluation metrics are highly dependent on the data sets used, influencing the interpretation of a model's performance. While T2GAN

demonstrates its ability to generate highly realistic time series samples, as evidenced by the similarity between PCA and t-SNE visualisations of real and synthetic data across all data sets, it is essential to consider the nuances of each data set when assessing the overall performance and applicability of the model in various contexts. Additionally, we show that the synthetic data performs reasonably well in its current state but is not on par with the original data in a downstream anomaly detection task.

7. Further research is necessary to fully understand the privacy implications of using synthetic time series data as a private source. We conclude that the data is synthesised completely from noise without seeing the original data. The individual samples may behave similarly to the original source but never as a direct copy.

While the improvements to the TimeGAN model have led to enhanced performance and more accurate representations of the time series data, it is essential to acknowledge that these enhancements also introduce increased model complexity to an already complex model. For example, incorporating the Wasserstein distance, temporal convolutional layers, and Transformer components adds to the computational cost, resulting in a longer training time. The main contributor to this increase is the gradient penalty computation, which requires temporarily disabling a GPU acceleration library utilised by PyTorch when computing the discriminator gradients. Furthermore, the TimeGAN framework training process involves training multiple networks individually and jointly, minimising three loss functions with a partly entangled parameter space. Consequently, the hyperparameter space is also complex. Although it certainly can be optimised further, it has to generalise well to different data sets.

A comprehensive ablation study is needed to assess these additions' contributions and significance. This will allow for a better balance between model complexity and performance and facilitate a deeper understanding of the essential components required for optimal results. Such analysis could also uncover potential redundancies or inefficiencies within the T2GAN model to address in future architecture iterations.

5.3 Future Work

- **Adaptation to textual data:** Investigate applying the T2GAN model to log files where textual data needs converting into classes. This process can lead to high-dimensional and imbalanced data sets, making it an exciting area for exploration and further model improvements.

- **Semantics enhancement:** Building upon the success of incorporating classifiers in tabular GANs, future work could incorporate a classifier within the T2GAN model to enhance the semantics within generated samples, especially when working with high dimensional data, such as log files, as maintaining consistency across the feature dim become increasingly complex.
- **Online learning:** Extend the T2GAN model to handle streaming sources, allowing it to adapt to changing patterns and trends in real time. This would require the model to continuously update its knowledge, leading to better performance and relevance for various applications. In this context, T2GAN's design of generating shorter sequences may be especially appropriate. However, it would necessitate the additional functionality of remembering and integrating previous sequences to ensure the continuity and coherence of the generated time series data.
- **Learning from longer sequence lengths:** Further exploration into developing the TimeGAN model to support longer sequences, allowing it to better handle and synthesise time series with very long dependencies or periodicities. This could involve refining the model's architecture or training approach to capture long-term dynamics more effectively.
- **Scalability:** Assess the scalability of the T2GAN model for large-scale, real-time streaming data applications. This would involve optimising the model's architecture and training process to ensure efficient processing and generation of synthetic time series data under real-world conditions.

These areas for future work can help advance the TimeGAN model's capabilities, making it more versatile and applicable to a broader range of time series data, such as log files and streaming data challenges.

6

Appendix

6.1 Appendix 1. Model Overview

Algorithm 2 Embedder Network

Input: $x \overset{\text{i.i.d}}{\sim} \mathcal{D}$, **Output:** $h \in \mathcal{H}$
Compute Positional Encoding Matrix pe:
    $pos \leftarrow [0, \ldots, N-1]^T$,
    $pe_{(pos,2i)} \leftarrow \sin(pos/10000^{\frac{2i}{d}})$,
    $pe_{(pos,2i+1)} \leftarrow \cos(pos/10000^{\frac{2i}{d}})$

Infuse input with positional encoding
$x \leftarrow x + pe$

Temporal ConvLayer:
    for $i = 1, \ldots n_{layers}$ **do**
        $x \leftarrow$ Conv1D(x)
        $x \leftarrow$ batch_norm(x)
        $x \leftarrow$ ReLU(x)

$x \leftarrow$ layer_norm(x)
$x \leftarrow$ fc_out(x)
return sigmoid(x)

Algorithm 3 Supervisor Network

Input: $h \sim \mathcal{H}$, **Output:** $h \in \mathcal{H}$
Compute Positional Encoding Matrix pe:
    $pos \leftarrow [0, \ldots, N-1]^T$,
    $pe_{(pos,2i)} \leftarrow \sin(pos/10000^{\frac{2i}{d}})$,
    $pe_{(pos,2i+1)} \leftarrow \cos(pos/10000^{\frac{2i}{d}})$

Infuse input with positional encoding
$h \leftarrow h + pe$

Temporal ConvLayer:
    for $i = 1, \ldots n_{layers}$ **do**
        $h \leftarrow$ Conv1D(h)
        $h \leftarrow$ batch_norm(h)
        $h \leftarrow$ ReLU(h)

$h \leftarrow$ layer_norm(h)
$h \leftarrow$ fc_out(h)
return h

Algorithm 4 Recovery Network

Input: $h \sim \mathcal{H}$, **Output:** $x \in \mathcal{X}$
Compute Positional Encoding Matrix pe:
    $pos \leftarrow [0, \ldots, N-1]^T$,
    $pe_{(pos,2i)} \leftarrow \sin(pos/10000^{\frac{2i}{d}})$,
    $pe_{(pos,2i+1)} \leftarrow \cos(pos/10000^{\frac{2i}{d}})$

Infuse input with positional encoding
$h\_pe \leftarrow h + pe$

Compute Multi-head attention:
    $Q =$ fc_q(h)
    $K =$ fc_k(h)
    $V =$ fc_v(h)
    $\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$
    $y = \alpha V$
    $h =$ fc(y)
$h \leftarrow$ layer_norm($h + h\_pe$)
for $i = 1, \ldots n_{layers}$ **do**
    $h \leftarrow$ GRU(h)
$h \leftarrow$ fc_out(h)
return sigmoid(h)

Algorithm 5 Generator Network

Input: $z \overset{\text{i.i.d}}{\sim} \mathcal{Z}$, **Output:** $h \in \mathcal{H}$
for $i = 1, \ldots n_{layers}$ **do**
    $z \leftarrow \text{GRU}(z)$
$z \leftarrow \text{layer\_norm}(z)$
$z \leftarrow \text{fc\_out}(z)$
return $\text{sigmoid}(z)$

Algorithm 6 Discriminator Network

Input: $h \sim \mathcal{H}$, **Output:** $y \in [0, 1]$
$h \leftarrow \text{GRU\_bidirectional}(h)$
$h \leftarrow \text{layer\_norm}(h)$
$h \leftarrow \text{fc\_out}(h)$
return h

Bibliography

[1] C. C. Aggarwal et al. “Neural networks and deep learning”. *Springer* **10**:978 (2018), p. 3.

[2] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu. “Achieving anonymity via clustering”. *ACM Transactions on Algorithms (TALG)* **6**:3 (2010), pp. 1–19.

[3] M. Arjovsky, S. Chintala, and L. Bottou. “Wasserstein generative adversarial networks”. In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[4] S. A. Assefa, D. Dervovic, M. Mahfouz, R. E. Tillman, P. Reddy, and M. Veloso. “Generating synthetic data in finance: opportunities, challenges and pitfalls”. In: *Proceedings of the First ACM International Conference on AI in Finance*. 2020, pp. 1–8.

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton. “Layer normalization”. *arXiv preprint arXiv:1607.06450* (2016).

[6] D. Bahdanau, K. Cho, and Y. Bengio. “Neural machine translation by jointly learning to align and translate”. *arXiv preprint arXiv:1409.0473* (2014).

[7] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[8] S. Bourou, A. El Saer, T.-H. Velivassaki, A. Voulkidis, and T. Zahariadis. “A review of tabular data synthesis using gans on an ids dataset”. *Information* **12**:09 (2021), p. 375.

[9] F. B. Bryant and P. R. Yarnold. “Principal-components analysis and exploratory and confirmatory factor analysis.” (1995).

[10] S. Burney and O. Bin Ajaz. “Copulas: a historical literature review and major developments abstract”. In: 2020.

[11] L. M. Candanedo, V. Feldheim, and D. Deramaix. *Appliances energy prediction data set*. `https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction`. Accessed: March 15, 2023. 2017.

[12] S. Castellanos. “Fake it to make it: companies beef up ai models with synthetic data”. *The Wall Street Journal* (2021). URL: https://www.wsj.com/articles/fake-it-to-make-it-companies-beef-up-ai-models-with-synthetic-data-11627032601.

[13] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. “Learning phrase representations using rnn encoder-decoder for statistical machine translation”. *arXiv preprint arXiv:1406.1078* (2014).

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

[15] C. Clifton and T. Tassa. “On syntactic anonymity and differential privacy”. In: *2013 IEEE 29th International Conference on Data Engineering Workshops (ICDEW)*. IEEE. 2013, pp. 88–93.

[16] M. D’Angelo. “Priorities and revenue of zero-touch automation”. In: Ericsson. 2023.

[17] G. Dévai. “Synthetic data generation tlg report”. In: Ericsson. 2022.

[18] D. Dheeru and E. Karra Taniskidou. *Uci machine learning repository*. https://archive.ics.uci.edu/ml/datasets/Abalone. Accessed: February 15, 2023. 2019.

[19] C. Dwork, A. Roth, et al. “The algorithmic foundations of differential privacy”. *Foundations and Trends® in Theoretical Computer Science* **9**:3–4 (2014), pp. 211–407.

[20] C. Esteban, S. L. Hyland, and G. Rätsch. “Real-valued (medical) time series generation with recurrent conditional gans”. *arXiv preprint arXiv:1706.02633* (2017).

[21] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama. “Machine learning for streaming data: state of the art, challenges, and opportunities”. *ACM SIGKDD Explorations Newsletter* **21**:2 (2019), pp. 6–22.

[22] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. “Generative adversarial networks”. *Communications of the ACM* **63**:11 (2020), pp. 139–144.

[23] *Google stock data*. Yahoo Finance. Accessed: March 15, 2023. 2023.

[24] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. “Recent advances in convolutional neural networks”. *Pattern recognition* **77** (2018), pp. 354–377.

[25] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. “Improved training of wasserstein gans”. *Advances in neural information processing systems* **30** (2017).

[26] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "Gans trained by a two time-scale update rule converge to a local nash equilibrium". *Advances in neural information processing systems* **30** (2017).

[27] G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

[28] S. Hochreiter and J. Schmidhuber. "Long short-term memory". *Neural computation* **9**:8 (1997), pp. 1735–1780.

[29] J. Jordon, J. Yoon, and M. Van Der Schaar. "Pate-gan: generating synthetic data with differential privacy guarantees". In: *International conference on learning representations*. 2019.

[30] T. Karras, S. Laine, and T. Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4401–4410.

[31] D. P. Kingma and M. Welling. "Auto-encoding variational bayes". *arXiv preprint arXiv:1312.6114* (2013).

[32] G. Lan. *Numenta anomaly benchmark (nab) dataset: nyc$_t$axi*. `https://github.com/numenta/NAB/tree/master/data"`. Accessed: March 15, 2023. 2012.

[33] N. Li, T. Li, and S. Venkatasubramanian. "T-closeness: privacy beyond k-anonymity and l-diversity". In: *2007 IEEE 23rd international conference on data engineering*. IEEE. 2006, pp. 106–115.

[34] S.-C. Li, B.-C. Tai, and Y. Huang. "Evaluating variational autoencoder as a private data release mechanism for tabular data". In: *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE. 2019, pp. 198–1988.

[35] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet. "Are gans created equal? a large-scale study". *Advances in neural information processing systems* **31** (2018).

[36] C. Ma, J. Li, M. Ding, B. Liu, K. Wei, J. Weng, and H. V. Poor. "Rdp-gan: ar\'enyi-differential privacy based generative adversarial network". *arXiv preprint arXiv:2007.02056* (2020).

[37] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. "L-diversity: privacy beyond k-anonymity". *ACM Transactions on Knowledge Discovery from Data (TKDD)* **1**:1 (2007), 3–es.

[38] A. Majeed and S. Lee. "Anonymization techniques for privacy preserving data publishing: a comprehensive survey". *IEEE access* **9** (2020), pp. 8512–8545.

[39] A. Mallasto, G. Montúfar, and A. Gerolin. "How well do wgans estimate the wasserstein metric?" *arXiv preprint arXiv:1910.03875* (2019).

[40] M. Mirza and S. Osindero. "Conditional generative adversarial nets". *arXiv preprint arXiv:1411.1784* (2014).

[41] A. Narayanan and V. Shmatikov. "Robust de-anonymization of large sparse datasets". In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE. 2008, pp. 111–125.

[42] S. I. Nikolenko. "Synthetic data for deep learning". *arXiv preprint arXiv:1909.11512* (2019).

[43] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. "Wavenet: a generative model for raw audio". *arXiv preprint arXiv:1609.03499* (2016).

[44] V. Pappu and P. M. Pardalos. "High-dimensional data classification". *Clusters, Orders, and Trees: Methods and Applications: In Honor of Boris Mirkin's 70th Birthday* (2014), pp. 119–150.

[45] N. Park, M. Mohammadi, K. Gorde, S. Jajodia, H. Park, and Y. Kim. "Data synthesis based on generative adversarial networks". *arXiv preprint arXiv:1806.03384* (2018).

[46] N. Patki, R. Wedge, and K. Veeramachaneni. "The synthetic data vault". In: *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2016, pp. 399–410.

[47] J. Pearl. "Fusion, propagation, and structuring in belief networks". *Artificial intelligence* **29**:3 (1986), pp. 241–288.

[48] A. Radford, L. Metz, and S. Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks". *arXiv preprint arXiv:1511.06434* (2015).

[49] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training" (2018).

[50] C. Robert and G. Casella. "A short history of markov chain monte carlo: subjective recollections from incomplete data" (2011).

[51] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved techniques for training gans". *Advances in neural information processing systems* **29** (2016).

[52] A. Srivastava, L. Valkov, C. Russell, M. U. Gutmann, and C. Sutton. "Veegan: reducing mode collapse in gans using implicit variational learning". *Advances in neural information processing systems* **30** (2017).

[53] J. Stanczuk, C. Etmann, L. M. Kreusser, and C.-B. Schönlieb. "Wasserstein gans work because they fail (to approximate the wasserstein distance)". *arXiv preprint arXiv:2103.01678* (2021).

[54] T. Tassa, A. Mazza, and A. Gionis. "K-concealment: an alternative model of k-type anonymity." *Trans. Data Priv.* **5**:1 (2012), pp. 189–222.

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

[56] A. Torfi, E. A. Fox, and C. K. Reddy. "Differentially private synthetic medical data generation using convolutional gans". *Information Sciences* **586** (2022), pp. 485–500.

[57] L. Van der Maaten and G. Hinton. "Visualizing data using t-sne." *Journal of machine learning research* **9**:11 (2008).

[58] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". *Advances in neural information processing systems* **30** (2017).

[59] C. Villani et al. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[60] *Wara ops*. wara-ops.org. Accessed: April 27. 2023.

[61] S. E. Whang, Y. Roh, H. Song, and J.-G. Lee. "Data collection and quality challenges in deep learning: a data-centric ai perspective". *The VLDB Journal* (2023), pp. 1–23.

[62] X. Xiao and Y. Tao. "Anatomy: simple and effective privacy preservation". In: *Proceedings of the 32nd international conference on Very large data bases*. 2006, pp. 139–150.

[63] Y. Xiao, G. Shi, Y. Li, W. Saad, and H. V. Poor. "Toward self-learning edge intelligence in 6g". *IEEE Communications Magazine* **58**:12 (2020), pp. 34–40.

[64] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni. "Modeling tabular data using conditional gan". *Advances in Neural Information Processing Systems* **32** (2019).

[65] J. Yoon, L. N. Drumright, and M. Van Der Schaar. "Anonymization through data synthesis using generative adversarial networks (ads-gan)". *IEEE journal of biomedical and health informatics* **24**:8 (2020), pp. 2378–2388.

[66] J. Yoon, D. Jarrett, and M. Van der Schaar. "Time-series generative adversarial networks". *Advances in neural information processing systems* **32** (2019).

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
May 2023

Document Number
TFRT-6200

Author(s)
Esaias Belfrage
August Borna

Supervisor
Torgny Holmberg, Ericsson, Sweden
Johan Eker, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Generative AI for Synthetic Data

Abstract

Synthetic data generation has emerged as a valuable technique for addressing data scarcity and privacy concerns and improving machine learning algorithms. This thesis focuses on progressing the field of synthetic data generation, which may play a crucial role in AI-heavy industries such as telecommunications. Generative Adversarial Networks successfully generate various types of synthetic data but fall short when modelling the temporal patterns and conditional distributions of time series data. State-of-the-art TimeGAN has shown promise, but there is potential for refinement. We propose T2GAN, utilising TimeGAN's novel framework of combining unsupervised and supervised training and extending it using state-of-the-art machine learning techniques, such as Transformers. Through experimental evaluation, we quantify the effectiveness of T2GAN using various benchmark data sets and find that the T2GAN model significantly surpasses the TimeGAN in both discriminative and predictive capacities. Our results demonstrate a 38% enhancement in similarity measures and a 55% reduction in relative prediction error when using synthetic training data. Furthermore, the thesis presents a comprehensive literature study and analysis of generative models, detailing the potential of T2GAN in various domains by enabling privacy-preserving data analysis, facilitating research and development, and enhancing machine learning algorithms.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-82

Recipient's notes

Security classification