Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Eivind Kohmann

Investigating the Capability of Generative Adversarial Networks in Capturing Implicit Laws in Physical Systems

Master's thesis in Master of Science in Informatics
Supervisor: Inga Strümke
Co-supervisor: Signe Riemer-Sørensen and Pål Vegard Johnsen
June 2023

NTNU
Norwegian University of
Science and Technology

Eivind Kohmann

Investigating the Capability of Generative Adversarial Networks in Capturing Implicit Laws in Physical Systems

Master's thesis in Master of Science in Informatics
Supervisor: Inga Strümke
Co-supervisor: Signe Riemer-Sørensen and Pål Vegard Johnsen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

The challenge of interpreting generative models, particularly in the context of Generative Adversarial Networks (GANs), poses a significant issue in understanding what they truly learn. Despite their remarkable ability to generate realistic-looking data, an important question arises whether the data also adhere to the same laws and constraints implicitly followed by the real data. Laws governing the underlying process or system embody crucial aspects of the real world and must be followed for the generated data to be considered useful and valid, especially when used for learning or analysis purposes.

This thesis investigates the capability of GANs to capture the underlying laws through data from a simulated physical system with known underlying laws. The system under investigation centers around the collision of shallow-water waves, which are solutions to the Korteweg-de Vries equation and obey the conservation principles. Experiments are conducted using a dataset of colliding waves of various heights, and samples are evaluated based on their adherence to conservation laws. We evaluate the performance difference between two separate GANs, which have the same architecture but are trained with and without information about the underlying conservation laws.

Comparisons of conservation error statistics show clear performance differences between the models. In the best-case scenario, the regular GAN exhibits, on average, more than three times more conservation errors compared to the informed version. Considering the average GAN performance, the conservation errors are much more significant. The findings suggest that GANs have limitations in accurately capturing the underlying constraints.

We find various barriers that hinder GANs' ability to capture the underlying constraints. It is primarily believed to be due to the statistical nature of GANs, as they are not able to catch the implicit laws only through sample statistics. Their performance is also greatly affected by the unpredictable and sensitive nature of GANs in response to hyperparameter combinations and architectural design, making it difficult to find the proper parameters to reach desired results. Future studies to explore alternative GAN architectures with various recurrent modules and with larger network capacities should be conducted to confirm whether similar results and challenges also persist across a broader range of GANs.

Sammendrag

Utfordringen med tolkning av generative modeller, spesielt blant Generative Adversarial Networks (GANs), utgjør et betydelig problem når det handler om å forstå hva de faktisk lærer. Til tross for deres bemerkelsesverdige evne til å generere data som ser realistisk ut, oppstår et viktig spørsmål om hvorvidt dataene også følger de samme lovene og begrensningene som implisitt følges av virkelige data. Lover som styrer den underliggende prosessen eller systemet, inneholder avgjørende aspekter av den virkelige verden og må følges for at den genererte dataene skal være gyldige, spesielt når de brukes til statistiske analyser.

Dette prosjektet undersøker GANs evne til å fange opp de underliggende lovene gjennom data fra et simulert fysisk system med kjente underliggende lover. Systemet som undersøkes, omhandler kollisjonen mellom havbølger, som er løsninger av Korteweg-de Vries-likningen og følger prinsippene om massebevaring, bevegelsesbevaring og energibevaring. Eksperimenter utføres ved bruk av et datasett med kolliderende bølger av ulike høyder, og modellene evalueres basert på deres samsvar med bevaringslovene. Vi evaluerer ytelsesforskjellene mellom to separate GAN modeller, som har samme arkitektur, men er trent med og uten kunnskap om bevaringslovene.

Sammenligninger av statistikk for bevaringsfeil viser tydelige forskjeller i ytelse mellom de to modellene. I det beste tilfellet viser en vanlige GAN over tre ganger mer bevaringsfeil sammenlignet med en informerte versjon. Men med tanke på en gjennomsnittlig GAN er bevaringsfeilene langt mer betydningsfull. Resultatene antyder at GANs har begrensninger når det gjelder å følge de underliggende begrensningene med liten feil.

Vi finner forskjellige årsaker som hindrer GAN modeller i å fange opp de underliggende begrensningene. Det antas primært å skyldes GANs statistiske natur, som ikke er i stand til å fange opp de implisitte lovene gjennom kun statistikk fra dataene. GANs evne påvirkes også i stor grad av GANs uforutsigbare og sensitive natur i respons på kombinasjoner av hyperparametere og nettverkdesign, noe som gjør det vanskelig å finne riktige kombinasjoner for å nå sitt potensiale. Fremtidige studier for å utforske alternative GAN-arkitekturer med ulike tilbakevendende moduler og høyere nettverkskapasiteter bør utføres for å bekrefte om lignende resultater og utfordringer også vises på et bredere spekter av GANs.

Preface

This thesis is presented as partial fulfillment for the degree of Master of Science in Informatics at the Norwegian University of Science and Technology (NTNU). It is the outcome of a collaborative project involving the Department of Computer Science (IDI) and SINTEF Digital and was supervised by Assoc. Prof. Inga Strümke at NTNU, along with Signe Riemer-Sørensen and Pål Vegard Johnsen at SINTEF. The thesis builds upon a preparatory specialization project conducted in the fall of 2022. Certain portions of this thesis's background, related work, and experiment sections have been adapted from the preparatory project. This is also mentioned in the respective sections.

I am grateful to all my supervisors for their continuous support throughout the year, providing valuable guidance and engaging in fruitful discussions that helped refine my ideas. A special appreciation goes to Signe and Pål for actively staying updated on my progress and reviewing my work.

Furthermore, I would like to thank Sølve Eidnes at SINTEF for providing the necessary notebooks for numerical solutions of the Korteweg–De Vries equation. Without this assistance, completing this thesis would have been considerably more challenging.

Trondheim, 9th June 2023
Eivind Kohmann

Contents

Figures

Tables

Acronyms

BCE Binary Cross Entropy. 24, 29, 31, 39, 54

C-RNN-GAN Continuous-RNN-GAN. 32, 33, 36, 41

COTGAN Causal Optimal Transport GAN. 37, 39, 51, 58, 62, 65–69, 80, 81, 83, 85, 102, 103, 107

GAN Generative Adveserial Network. 1, 3–5, 21–23, 27, 29–31, 34, 37, 40, 42–44, 49–54, 85, 103, 107–111

GRU Gated Recurrent Unit. 34, 40, 41, 111

KdV Korteweg-de Vries. 52, 71, 72, 77, 78, 107

KL Kullbeck-Leibler. 10, 11

LSTM Long Short-Term Memory. 31, 39, 111

PCA Principal Component Analysis. 36, 42, 46, 55, 63, 65, 66, 88, 89, 91, 96, 99, 108, 109

PDE Partial Differential Equation. 43, 71

RGAN Recurrent GAN. 32–34, 36, 39, 41, 51, 58, 59, 62, 65–69, 107

RTSGAN Real-World Time Series GAN. 39–42, 51, 58, 61, 65–69, 107

SW Fast approximation of the Sliced-Wasserstein distance. 3, 14, 17, 18, 45, 63, 65–67, 69, 77, 83, 84, 89, 91, 95, 99, 103–105, 107, 108, 110, 130, 133

t-SNE t-Distributed Stochastic Neighbor Embedding. 36, 42, 46, 47, 55, 63, 66, 85, 88, 89, 96, 99, 108, 109

TimeGAN Time Series GAN. 33–36, 39, 41, 51, 58–60, 65–69, 107

UMAP Uniform Manifold Approximation and Projection. 46, 47, 55, 63, 65, 66, 85, 89, 96, 108, 109

Chapter 1

Introduction

In the field of artificial intelligence, there has been lots of innovation the recent years and rapid implementations to utilize the latest techniques for various applications. Among these techniques are generative models, which have become increasingly popular due to their ability to create unrestricted amounts of realistic data. They can do so in an unsupervised manner without the need for labels, making them applicable to a wide range of tasks. Generative models have gained significant attention and adoption across diverse fields, including computer vision, natural language processing, and audio, and their utility continues to advance with its fast-moving research frontier. The success of generative models, especially in the form of Generative Adveserial Networks (GANs) [1], flow-based models [2], and variational autoencoders [3], has revolutionized how researchers and industries approach problems related to small datasets, privacy protection [4–8], anomaly detection [9–12], and uncertainty quantification [13–16] to mention a few.

However, despite the success of generative models, they have also presented new challenges, particularly in terms of trust and interpretation of generated data. Generative models can generate instances that do not match a particular case in the available dataset but resemble multiple known ones, which could be considered a valid sample by a human observer. Evaluating the sample validity can not be achieved solely by comparing them to existing data, and existing statistical methods may be considered insufficient without the adoption of domain knowledge. This issue has gained significance as GANs, for example, are being studied for critical applications such as medical diagnosis [17], MR image reconstruction [18, 19], and upsampling brain signal data [20], where interpretability is crucial for ensuring the safety and effectiveness of the models.

In many applications, it is necessary for the generated data to not only look realistic but also adhere to the same constraints as the real data. Both explicit constraints concerning specific data limitations, such as value ranges, and implicit constraints, which the data obeys as a consequence of being a measurement of a system governed by laws and underlying principles, for instance, conservation laws in physical systems. In situations when the generated data does not adhere

to the correct constraints, the data convey incorrect information or impossible cases that may be utilized without being aware of it.

In domains where underlying constraints in the system or process being modeled are known and defined, they can be incorporated into the model to force the generation of valid data. However, in most cases, the underlying constraints are either unknown, impossible to define, or inaccessible. In such cases, the GAN must learn the underlying constraints solely through the statistics of the data and without knowledge about the underlying system. While it may be tempting to assume that the generated data will also emulate the inherent rules of the original system, there is no guarantee of this. This thesis delves further into exploring the level at which a GAN can successfully learn and follow such underlying constraints.

During the work of this thesis, OpenAI released the highly popular ChatGPT. While the architecture and purpose of ChatGPT are different from GANs, the question of what it truly learns remains common. Understanding the limitations and interpretability of generative models like ChatGPT is crucial for ensuring their reliability and trustworthiness.

1.1 Motivation

Understanding the limitations of Generative Adversarial Networks in capturing the implicit laws in systems is crucial for utilizing them correctly and in appropriate applications. Implicit constraints are present in all types of data and must be strictly followed for the generated data to be considered valid. If generated samples do not adhere to the same underlying laws, they may induce different statistics and potentially influence decision-making processes. Enhancing our knowledge concerning the capabilities of GANs would not only highlight possible model flaws but also drive research toward developing more reliable GAN frameworks.

Despite the notable progress of GANs and their widespread claim for generating realistic-looking data, demonstrated by models like StyleGAN [21] and DragGAN [22], there remains a significant gap in comprehending their performance in domains that go beyond common images. Examples of such data types include medical images, time series data, and system modeling. Although GANs have demonstrated remarkable success in image synthesis tasks, evaluating their performance and reliability in these specialized domains is of great importance for practical applications.

Existing research has explored the capabilities of GANs in adhering to geometrical constraints in image data [23], to generate divergence-free fields [23], and their ability to adhere to the Euler scheme when predicting values for one-dimensional Lorenz systems [24]. These studies demonstrated that GANs often struggle to adhere to underlying constraints. Additionally, Stinis *et al.* [24] acknowledged the need for further investigations into more complex and higher-dimensional systems. Further examination of other data types and settings would contribute to a better understanding of the capabilities and effectiveness of GANs

in various domains.

One specific area of interest lies in the domain of time series with complex high-dimensional data, which pose unique challenges for generative models. In the case of time series data, accurately capturing temporal dependencies and dynamics is crucial for generating meaningful samples, especially in the presence of nonlinearities within complex high-dimensional data.

This thesis aims to explore this gap by investigating the performance of GANs in simulating physical systems. Specifically, evaluate the extent to which generated simulations adhere to conservation laws within a closed system involving colliding waves described by the Korteweg-de Vries (KdV) equation [25].

1.2 Contribution

This thesis makes contributions to the field of Generative Adversarial Networks by presenting findings related to their fidelity and methods for evaluating them.

First and foremost, it extends previous research on GAN's capability to adhere to underlying constraints, specifically in the domain of high-dimensional multivariate time series data. While prior research has predominantly focused on image data and one-dimensional sequences, this thesis delves into the challenges and capabilities of GANs when applied to complex and higher-dimensional time series data.

To address the evaluation problem associated with GANs, a novel evaluation metric called the Fast approximation of the Sliced-Wasserstein distance (SW), proposed by Nadjahi [26], is employed and tested. The SW metric estimates the distance between two distributions when only samples from each distribution are accessible. By utilizing this metric, the thesis tackles the evaluation challenge and assesses the performance of GAN models in generating time series data.

Furthermore, this thesis provides an overview of various GAN frameworks for generating multivariate time series data. It explores different GAN architectures and techniques, comparing their performances and shedding light on the strengths and weaknesses associated with each approach.

1.3 Research questions

This thesis aims to explore the capabilities and limitations of Generative Adversarial Networks in adhering to underlying physical laws and examine the effectiveness of a novel evaluation metric. The following research questions serve as the primary inquiries driving this thesis:

- **RQ1**: *How well do regular GANs conform to the underlying physical laws governing the data in comparison to physics-informed GANs?*
- **RQ2**: *What are the limitations and challenges GANs face in adhering to complex physical laws in multivariate time series data?*

- **RQ3**: *How effective is the approximated Sliced-Wasserstein distance in evaluating the performance of GANs on time series?*

However, before delving into the primary research questions, it is crucial to address an additional and important secondary research question to ensure the adoption of an appropriate model for the main experiments:

- **RQ1.1**: *What is the comparative performance of different time series GAN models when generating long sequence lengths?*

Answering RQ1.1 will provide valuable insights into the performance and suitability of different time series GAN models, establishing a solid ground for the subsequent investigation of the primary research questions.

1.4 Research method in brief

To comprehensively analyze different GAN frameworks, an initial experiment is carried out using a straightforward dataset to compare and evaluate their performance. Based on the outcomes of this experiment, the most promising framework is selected for the main experiment, which also answers RQ1.1.

The main experiments involve assessing the performance difference between a regular and physics-informed GAN. The physics-informed GAN is employed as a benchmark for comparison, enabling the analysis of performance metrics, sample statistics, and adherence to underlying constraints.

1.5 Structure of the report

The thesis is structured as follows: **Chapter 2** introduces necessary and essential theoretical ideas to understand GANs, how they operate, and ideas for evaluating them. This includes a minimal introduction to probability theory and optimal transport theory. Using the theory and similar mathematical notation, we formally introduce GANs in **Chapter 3**. **Chapter 4** presents an overview of different GANs for time series generation, including important milestones and varieties, along with previous work on investigating GANs on physical systems. **Chapter 5** presents an outline for the experiments with considerations and design choices. It includes an explorative experiment in addition to the main experiments. Every implementation detail, along with results and discussion, is presented about the exploratory experiment in **Chapter 6**, while **Chapter 7** includes the implementation details concerning the main experiment. The results of the main experiment are then presented and discussed in **Chapter 8**. The final chapter, **Chapter 9**, concludes our findings along with future work.

Chapter 2

Theoretical background

This chapter introduces a theoretical foundation to understand key concepts about GANs and evaluation metrics. The covered topics encompass a brief introduction to concepts in probability theory, statistical distance, and the utilization of optimal transport theory for transforming probability distributions.

2.1 Probability theory

Probability theory is one of the cornerstones in machine learning and deep learning methodologies, and a poor understanding of probability theory makes it difficult to grasp important ideas and methods. Therefore, we introduce some essential and fundamental ideas to refresh the underlying theory and provide the minimum knowledge needed to understand upcoming concepts.

This introduction is meant to be short and only refreshes parts of the field. We will therefore leave out several noteworthy corollaries and edge cases. Only the discrete cases, situations where the set of all possible outcomes is finite, are introduced here as we believe extending them beyond does not provide a significantly better understanding than what is captured in the discrete cases. The reader is assumed to be familiar with basic set theory and mathematical notation.

Probability space

In probability theory, a construct to formally model a random process or experiment is called a *probability space*. Probability spaces are the foundation of probability theory and provide a framework for analyzing and understanding randomness and uncertainty. They consist of three key elements:

1. A sample space, Ω, is the set of all possible elementary outcomes from a random process or experiment.
2. An event space $\mathcal{F}$, which is a selected number of subsets of Ω describing events we would like to consider. An event $E \in \mathcal{F}$ represents a set of outcomes of the experiment.

3. A probability function $P : \mathcal{F} \to [0,1]$, which assigns a probability to each event in the event space, represented by a number between 0 and 1.

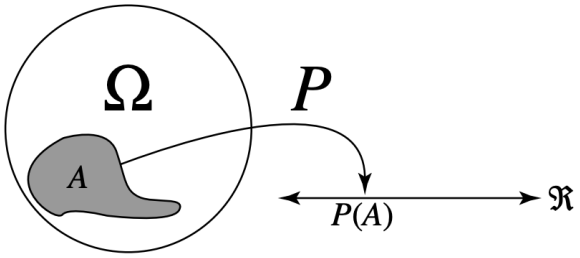


Figure 2.1: An illustrative figure of the three elements that together define a probability space. A sample space Ω, an event A, and a probability measure P. (Illustration from Paskin [27].)

In order to create a sensible model of probability, the elements must be defined in such a way they satisfy three particular axioms, namely *The axioms of probability*:

1. The probability of an event is always non-negative, $P(E) \geq 0$ for all $E \in \mathcal{F}$
2. The probability of the sample space is 1, $P(\Omega) = 1$
3. For two disjoint events $A, B \in \mathcal{F}$, the probability of both events happening is equal to the sum of each respective event happening, $P(A \cup B) = P(A) + P(B)$

It is important to note that all $\omega \in \Omega$ are referred to as outcomes or elementary events. Moreover, the probability of an event $E \in \mathcal{F}$ is computed as $P(E) = \sum_{\omega \in E} P(\omega)$, the sum of the probability of all outcomes in the event. Since we define each event as a subset of the discrete sample space, an event is considered countable, meaning it contains only finitely many elements.

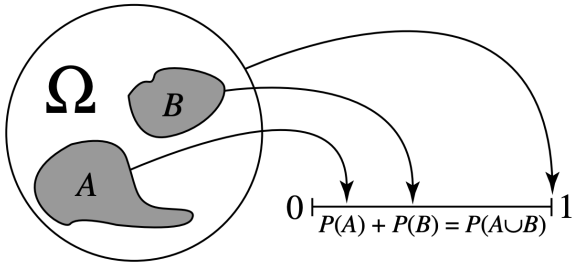


Figure 2.2: Illustration of the axioms of probability theory. (Illustration from Paskin [27].)

An example of a probability space is rolling a fair 6-sided die. Here, the sample space would equal $\Omega = \{1,2,3,4,5,6\}$ all the individual sides and let the event space $\mathcal{F}$ be the set of all subsets of the sample space. The event space would then contain events such as $\omega = \{6\}$ ("the die rolled a 6") or $\omega = \{1,3,5\}$ ("the die rolled an odd number"). By the use of a fair die, the probability function

can be defined for each outcome to be equal to $1/6$. In formal terms, $P(\omega) = 1/6$, for all $\omega \in \Omega$. The probability function could then be used to give the probability of a particular event happening, e.g., $P(\{6\}) = 1/6$ and $P(\{1,3,5\}) = P(\{1\}) + P(\{3\}) + P(\{5\}) = 3/6 = 1/2$.

Random variables

To quantify the probability of certain events, *random variables* serve as a formalization to "pick out" aspects of the experiment's outcomes. A random variable is defined as function $X : \Omega \to \Xi$ from a sample space Ω to a measure space Ξ, which is often the real numbers for continuous X. The purpose of X is to provide a means of measuring events in a way that can be analyzed and understood. The mapping of events to outcomes in Ξ allows exactly that.

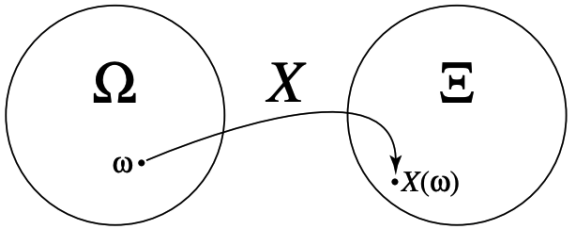


Figure 2.3: A random variable X mapping outcomes to measure space. (Illustration from Paskin [27].)

Random variables can be used to define events, e.g., $\{\omega \in \Omega : X(\omega) = true\}$. Such a set contains all of the outcomes ω that result in the function X being true. This allows us to define specific events and determine the probability of that event happening by using the probability function P, e.g., $P(\{\omega \in \Omega : X(\omega) = 1\})$, which is often reduced to just $P(X = 1)$.

The choice of X depends on the nature of the experiment and the quantity of interest. Its purpose is to reflect on some interesting outcomes of the experiments and provide insight.

An example of a random variable is the outcome of rolling a fair 6-sided die. We define the random variable X as the function that maps each possible outcome of rolling the die to that number itself, $X(\omega) = \omega$. The sample space Ω is the set $\{1,2,3,4,5,6\}$, which the co-domain Ξ also equals since we map the outcomes to themselves. Using this random variable, we can define events such as $X \geq 4$, which is the event that the number rolled is greater than or equal to 4. We can also calculate the probability of certain events, such as $P(X = 3)$, which is the probability of rolling a 3.

Probability mass

The probability of a certain event in the sample space is calculated via the *probability mass*. Considering the random variable X takes on a specific value, it is defined as measuring the probability mass of X where $X = x$. In general, the probability mass function $p_X : \Xi \to [0,1]$ is defined as $p_X(x) := P(\{\omega : X(\omega) = x\}) = P(X = x)$ for all $x \in \Xi$. For continuous random variables, this is instead referred to as the *probability density*.

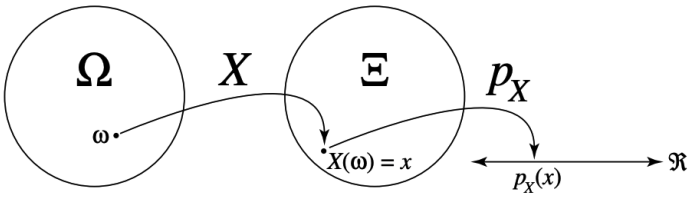


Figure 2.4: A probability mass function maps an outcome of a random variable to a number. (Illustration from Paskin [27].)

For the probability mass function to be valid, every $x \in \Xi$ needs to have a non-zero probability mass, and the sum of the probability of each event must equal 1. Formally summarized as

$$\sum_{x \in \Xi} p_X(x) = 1 \quad \text{and} \quad p_X(x) \geq 0, \text{for all } x \in \Xi.$$

As the probability mass function can provide the probability of any event described by X, fetching the probability for all the possible values X can take provides us with a *probability distribution*.

Probability distribution

A probability distribution provides a comprehensive understanding of the behavior of a random variable and enables the calculation of fundamental statistics such as the *expected value* and *variance*. Statistics offer insights into the random variable's behavior, facilitating predictions for future experiments and comparisons with other probability distributions.

The expected value represents the weighted average of all possible outcomes, offering insights into the random variable's most frequently occurring average value. It is defined as:

$$\mathbb{E}[X] = \sum_{x \in \Xi} x p_X(x). \tag{2.1}$$

The variance measures how spread out the distribution is from the expected value. It is defined as the expected value of the squared difference between the random variable and its expected value:

$$\text{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \sum_{x \in \Xi} (x - \mathbb{E}[X])^2 p_X(x). \tag{2.2}$$

While the expected value and variance are essential descriptors for a useful distribution representation, higher statistical calculations known as *moments* can be computed to provide more comprehensive insights. However, out of these various moments, the expected value and variance are the most frequently utilized and commonly reported statistics in academic research.

Multiple random variables

It is possible to define multiple random variables and extend the concepts discussed earlier to calculate the probability of multiple outcomes. Suppose we have two random variables, X and Y, denoted by $X : \Omega \to \Xi$ and $Y : \Omega \to \Upsilon$, respectively. The joint probability mass function $p_{XY}(x, y) = P(\omega : X(\omega) = x, Y(\omega) = y)$ describes the probability of both X and Y taking certain values x and y, respectively, where $x \in \Xi$ and $y \in \Upsilon$.

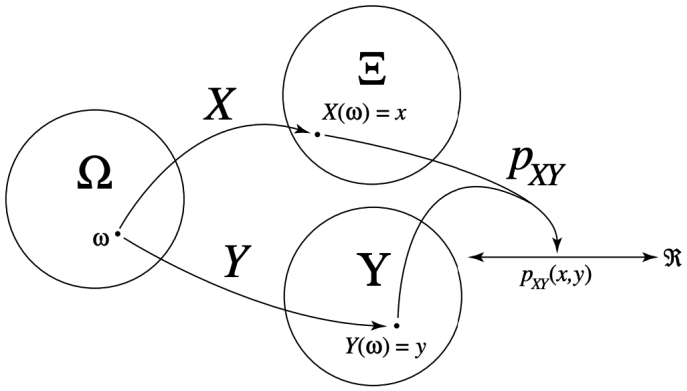


Figure 2.5: A probability mass function maps the outcome of two random variables to a number. (Illustration from Paskin [27].)

To compute the probability of an event involving both X and Y, we can use the joint probability mass function, such as the probability that X takes a value less than or equal to x and Y takes a value less than or equal to y, given by $P(X \leq x, Y \leq y) = \sum_{i=1}^{x} \sum_{j=1}^{y} p_{XY}(i, j)$.

When several random variables $(X_1, X_2, ...X_n)$ are considered in the same example, grouping them into a single vector called a *random vector* is common. The joint distribution of a random vector $\mathbf{X} = (X_1, X_2, ...X_n)$ is then denoted as

$$P(X_1 = x_1, X_2 = x_2, ..., X_n = x_n),$$

where $x_1, x_2, ..., x_n$ are the respective outcomes of the random variables. This compact notation reduces redundancy and enables one to discuss many related random variables at the same time.

It is worth noting that when dealing with random variables and densities, the probability space is always implicitly present behind the calculations, although it is often overlooked. In most experiments, you do not have access to the probability

shape, but you can infer it through samples (and sometimes assumptions about its shape, e.g., Gaussian).

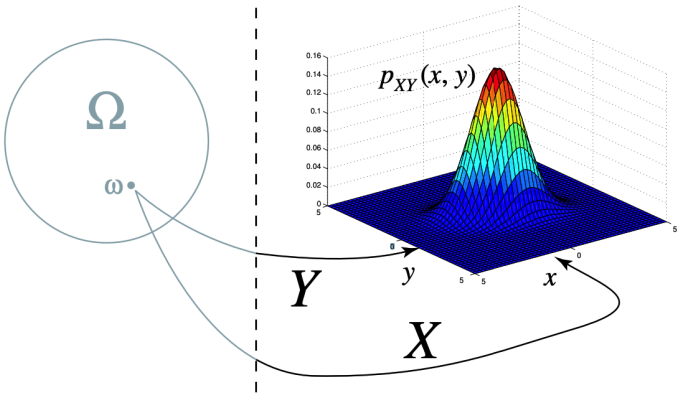


Figure 2.6: The probability space is often implicit when working directly with random variables and their joint densities. (Illustration from Paskin [27].)

2.2 Metrics and divergence

In the context of evaluating machine learning models and comparing probability distributions, *metrics* and *divergences* play a crucial role. A metric, formally defined as a function, quantifies the similarity or dissimilarity between objects or points in space. In particular, a metric on a set $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that satisfies the following properties for all $x, y, z \in \mathcal{X}$:

1. Non-negativity: $d(x, y) \geq 0$, and $d(x, y) = 0$ if and only if $x = y$.
2. Symmetry: $d(x, y) = d(y, x)$.
3. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$.

Described with words: 1) the distance metric must always be greater and only zero when considering the distance from itself, 2) the distance is symmetric, and 3) the distance is always shortest when not considering an intermediate point unless it lies on the same line between x and z.

These properties ensure that a metric provides a valid measure of distance or dissimilarity between points in the set $\mathcal{X}$. The Euclidean distance on a plane is a famous example: $d_2((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

On the other hand, the notion of divergences is a different type of measure that does not satisfy all the axioms of a metric but is still considered crucial in the field of statistics. A divergence measures the dissimilarity between two probability distributions [28]. One commonly used divergence is the Kullbeck-Leibler (KL) divergence, denoted as $D_{\mathrm{KL}}(P \parallel Q)$, where P and Q are two probability distributions. The KL divergence quantifies the difference between the two distributions by computing the expected value of the logarithmic difference between their probabilities. Mathematically, the KL divergence is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{2.3}$$

Despite being asymmetric, with $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$, and violating the triangle inequality, the KL divergence remains highly valuable. It is widely used to measure dissimilarity between probability distributions in various applications, such as information theory and machine learning.

2.3 Optimal transport theory

Optimal transport is a problem concerning transforming one probability mass distribution to another as efficiently as possible. Thinking of one dirt pile and a hole with the same volume, one can imagine filling in the hole by shoveling the dirt into the hole with the least travel distance and dirt being moved around.

To introduce the idea of the theory, we will consider moving a distribution of boxes from a stack defined by the probability measure p_X to a stack q_X in discrete positions. The left-hand side in Figure 2.7 illustrates the defined stack for the two distributions containing 6 boxes and 5 discrete positions. Here each box has a mass equal to $1/6$, and the probability mass of a particular position corresponds to the sum of the mass of the boxes.

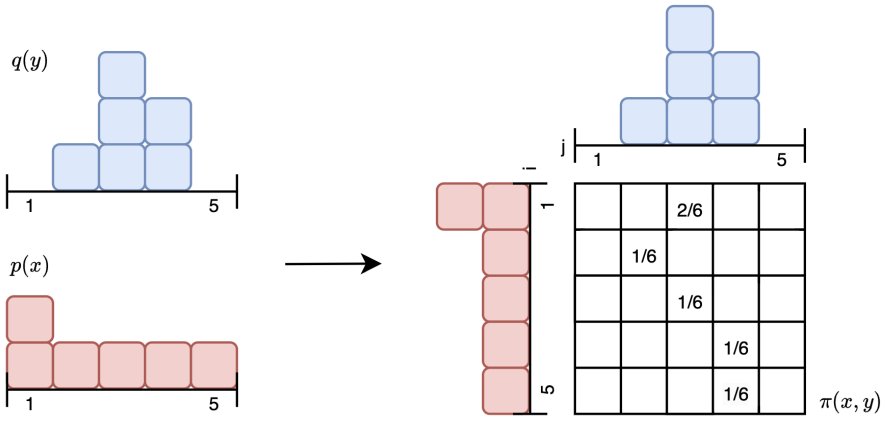


Figure 2.7: Left: Two probability distributions p (red boxes) and q (blue boxes), illustrated of boxes with destiny equal $1/6$. Right: A valid and optimal transportation plan π (matrix) to move probability mass from the distribution p to look like q. Blank matrix entries are zeros.

By using optimal transport, we would like to transform p_X into q_Y using a minimal amount of total work. Work is defined as the distance moved multiplied by the mass transported. To determine the cost of moving a box at position x_i to y_j, a **transportation cost** c is used and defined as:

$$c(x_i, y_j)^p = ||x_i - y_j||^p, \tag{2.4}$$

where $||\cdot||$ denotes the norm, and the exponent p defines the costliness of moving long distances. We set p to equal 1 for this example. This makes the cost function a measure of the distance between two points, which is always positive.

In the discrete case, it is easy to construct a transportation cost matrix $\mathbf{C}_{ij}$ to describe the distances from every x_i to y_j where the indices i and j specify the index in the cost matrix. In our example, $\mathbf{C}_{ij}$ would be a 5×5 matrix.

To determine how many units of mass to transport from one distribution to another, a **transportation plan** $\pi(x, y)$ is considered[1]. This is a map that explains how much mass and where to transport it to transform the initial distribution into the other and serves as the function we are interested in solving. The transportation plan can also be registered as a matrix, $\boldsymbol{\pi}_{ij}$, with the same size as the cost matrix, but where each entry describes how much mass to move from x_i to y_i. For π to be meaningful and concise with our probability mass distributions, it needs to satisfy three conditions:

1. $\sum_j \pi(x_i, y_j) = p(x_i)$, for all rows $i \in \{1, 2, ..., 5\}$
2. $\sum_i \pi(x_i, y_j) = q(y_j)$, for all columns $j \in \{1, 2, ..., 5\}$
3. $\pi(x_i, y_j) \geq 0$, for all combinations $i, j \in \{1, 2, ..., 5\}$

Here condition 1) states that the total mass transported at a starting position x_i must equal the amount of mass available to be transported, which is $p(x_i)$. In the example of a transportation plan in Figure 2.7, this can be thought of as summing row-wise and obtaining the correct total mass for the red distribution. Similarly, for condition 2), which states that where the mass is transported to, y_j must equal the amount available to be collected, which is $q(y_j)$. Using the same figure, this can be understood as summing column-wise and obtaining the blue distribution. Both conditions 1 and 2 make sure that all of the mass is considered when transforming between the distributions. The third condition states that all transportation plans must have non-negative values: Moving negative mass is prohibited.

By combining the transportation costs and a transportation plan, we can formulate an equation that computes the total work required to transform one distribution into another. Multiplying each piece of mass by the distance it has moved, the total work can be mathematically formulated as follows:

$$\text{Total work} = \sum_i \sum_j \boldsymbol{\pi}_{ij} \mathbf{C}_{ij}, \tag{2.5}$$

where we multiply the matrices component-wise and sum over all the indices. The main objective considered in optimal transport theory is then to minimize this quantity (equation 2.5), which can be achieved using linear programming. However, when the number of observations, for instance, the cost and plan matrix, becomes large, this is no longer feasible. Luckily, approximations and alternative

[1]Allowing the masses to be split corresponds to the Kanotorovich formulation of the optimal transport problem. This is a relaxation of the Monge formulation of the problem, which is more strict, thus, harder to solve. Kanotorovich's formulation is more computationally feasible and commonly used in modern applications. For more details, see Villani [29] and Peyré and Cuturi [30]

optimization problems exist to combat this issue.

The idea behind this formulation of the theory was to introduce it in a setting that was intuitive and easy to grasp. Therefore, only discrete probability mass distributions with the smallest movable mass being $1/6$ were considered. In reality, this mass can be split into whatever quantity, and the distributions can be continuous. This requires, however, a more detailed notation and familiarity with several other ideas, which can be found in much more sophisticated works such as "Optimal Transport" by Villani [29] and "Computational Optimal Transport" by Peyré and Cuturi [30].

2.3.1 Wasserstein distance

The *Wasserstein distance*, also known as the Earth Mover's distance and first defined by Kantorovich [31], is a commonly used method for quantifying the dissimilarity between two probability distributions. It is exactly the same optimal transport problem introduced in the previous section but formulated in terms of probability theory. As a result, the Wasserstein distance has a seemingly complex representation that is expressed as follows:

$$\mathcal{W}_p(p_X, q_Y) = \left(\inf_{\pi \in \Pi(p_X, q_Y)} \mathbb{E}_{(x,y)\sim\pi}[c(x,y)^p] \right)^{1/p}, \tag{2.6}$$

where $\mathcal{W}_p(p_X, q_Y)$ represents the Wasserstein distance between two probability measures, p_X, and q_Y, and the cost of transporting mass from x to y is given by $c(x, y)$. The "inf" denotes the selection of the least costly element in the set of all possible transport plans $\pi \in \Pi(p_X, q_Y)$, which represent all possible transport plans that move mass from p_X to q_Y. The power p in the cost function acts as a sensitivity to large distances, and taking the p-th root at the end ensures that the resulting distance measure is a true metric (defined in section 2.2). The computation of the work can be rephrased as the integration of the cost function over the joint probability space defined by the transport plan π. This integration can be equivalently expressed as the expected value of the cost function. Here shown for continuous variables X and Y:

$$\begin{aligned} \text{Total work} &= \int\int c(x,y)^p \, \pi(x,y) dx \, dy \\ &= \int c(x,y)^p d\pi(x,y) \\ &= \mathbb{E}_{(x,y)\sim\pi}[c(x,y)^p] \end{aligned} \tag{2.7}$$

The Wasserstein equation satisfies all three axioms of a metric (defined in section 2.2), which means it is a proper distance function defined to measure the distance between two probability distributions. This can be verified as $\mathcal{W}(P,Q) = 0$ only when $P = Q$. It is symmetric $\mathcal{W}(P,Q) = \mathcal{W}(Q,P)$ as this would correspond

to just a transposed transport plan and satisfies the triangle inequality. It is more difficult to prove that it holds the triangle inequality but is shown by Clement and Desch [32].

As stated in section 2.3, calculating the Wasserstein distance between two probability distributions is not feasible if the number of observations is large. However, some special cases exist where the Wasserstein distance can be computed easily, which will be important for later sections. One of those occurrences is when the distributions involved follow a Normal distribution. Calculating the Wasserstein-2 distance (with $p = 2$) can then be obtained using this formula:

$$\mathcal{W}_2\left(N(\mathbf{m}_1, \Sigma_1), N(\mathbf{m}_2, \Sigma_2)\right) = ||\mathbf{m}_1 - \mathbf{m}_2||^2 + \mathrm{Tr}\Big[\Sigma_1 + \Sigma_2 - 2(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2}\Big], \tag{2.8}$$

for two normal distributions $\mathcal{N}(\mathbf{m}_1, \Sigma_1)$, $\mathcal{N}(\mathbf{m}_2, \Sigma_2)$ with means $\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{R}^d$, and covariance matrices $\Sigma_1, \Sigma_2 \in \mathbb{R}^{d\times d}$ over dimensions d, which often represent the number of variables in a dataset. Here Tr denotes the Trace operator, which sums over all elements on the main diagonal. In the case of symmetry where $\Sigma_1\Sigma_2 = \Sigma_2\Sigma_1$, the formula can be written in the simpler form

$$\mathcal{W}_2\left(N(\mathbf{m}_1, \Sigma_1), N(\mathbf{m}_2, \Sigma_2)\right) = ||\mathbf{m}_1 - \mathbf{m}_2||^2 + \left|\left|\Sigma_1^{1/2} - \Sigma_2^{1/2}\right|\right|_F^2, \tag{2.9}$$

where $||\cdot||_F$ denotes the Frobenius norm, essentially summing the squares off all elements and taking the square root as such for a matrix A of size $n \times m$, $||A||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m}|a_{ij}|^2}$. Proof of why this is true is provided by Givens and Shortt [33].

The second case where the calculation of the Wasserstein distance happens to be straightforward is for univariate distributions. In the case of two sets of n observations $\{x_i\}_{i=1}^n, \{y_i\}_{i=1}^n$ from two univariate distributions p_X, q_Y, respectively, the distance can be calculated using

$$\mathcal{W}_p(p_X, q_Y) = \frac{1}{n}\sum_{i=1}^{n}||x_i - y_i||^p, \tag{2.10}$$

where the sets $\{x_i\}_{i=1}^n, \{y_i\}_{i=1}^n$ are sorted, such that $x_1 \leq ... \leq x_n$ and $y_1 \leq ... \leq y_n$. The formula can be expressed analytically and more precisely defined but is avoided here to reduce confusion. A complete formulation is provided by Nadjahi [26] on page 37.

In cases when the distributions are not Gaussian or univariate, the Wasserstein is intractable to calculate for high-dimensional problems. However, recent advances in optimal transport have provided estimation methods of the Wasserstein distance, which have been popularized and allow the application of optimal transport to high-dimensional data. We will introduce and employ two of these methods: the *Sinkhorn divergence* and the *Fast approximation of the Sliced-Wasserstein distance*.

2.3.2 Sinkhorn divergence

Sinkhorn divergence [34] estimates the Wasserstein distance by introducing a notion of entropy in the optimization objective, referred to as *entropic regularization*. This has a blurring effect on the optimal transport plan and allows for different algorithms that can calculate the new optimum more efficiently. Here we will not introduce the algorithm behind the calculation but rather the idea behind this formulation of the problem. See Cuturi [34] for details concerning the algorithm.

The main difference between the Wasserstein distance and the Sinkhorn divergence is that the latter introduces an entropy term H in the original Wasserstein distance. It measures the entropy of a transport plan, defined as:

$$H(\pi) = -\sum_i \sum_j \pi_{ij} \big(\log(\pi_{ij}) - 1\big), \tag{2.11}$$

where the transport plan π is a matrix, and H is the entropy function. The entropy function acts as a regularizer that punishes the transport plan for having entries equal or very close to 0, e.g., if one of the entries in π is 0, $H(\pi) = -\infty$. Adding the regularization function to the original Wasserstein-distance objective, we get the following:

$$\mathcal{W}_{p,\epsilon}(p_X, q_Y) = \left(\inf_{\pi \in \Pi(p_X, q_Y)} \left\{ \mathbb{E}_{(x,y)\sim\pi}[c(x,y)^p] - \epsilon H(\pi) \right\} \right)^{1/p}. \tag{2.12}$$

Here the $\mathcal{W}_{p,\epsilon}(p_X, q_Y)$ denotes the estimated Wasserstein distance with a new parameter ϵ, which controls the significance of the entropy term. Notice that the infimum considers both terms. This results in the optimal transport plan being less strict and encouraging mass transportation more evenly, not just from the high peaks. Figure 2.8 shows the effects of reducing the regularization strength for a simple 1-dimensional optimal transport problem. The top heatmaps and bottom surface plots visualize the optimal transport plans for different ϵ values, where the marginal densities are displayed as blue and red lines on the bottom plots. It should be noted that the Sinkhorn divergence does not satisfy all of the axioms to be defined as a metric, which the original Wasserstein distance does, as $\mathcal{W}_{p,\epsilon}(p_X, p_X) \neq 0$. Hence it is called a divergence.

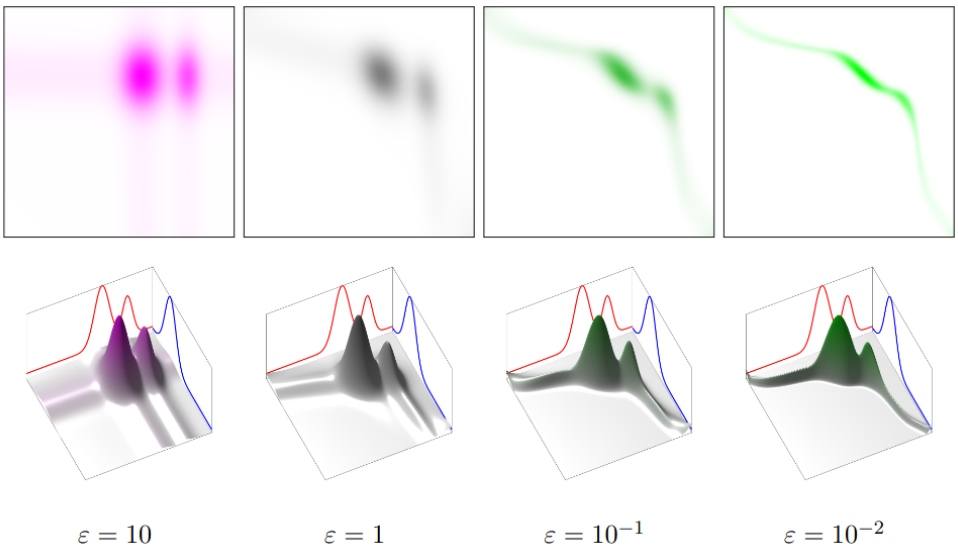


Figure 2.8: The impact of ϵ between the optimal transport plan between two 1-dimensional densities. (Illustration from Peyré and Cuturi [30]).

2.3.3 Sliced-Wasserstein distance

The Sliced-Wasserstein distance [35] is an alternative approach for estimating the Wasserstein distance by simplifying the computation by mapping high-dimensional data onto a 1-dimensional space. This allows the Wasserstein distance to be estimated using 1-dimensional distributions instead of high-dimensional distributions, which is much more feasible.

The technique involves slicing the distributions along various random directions defined by random unit vectors u_l. The slicing is defined as taking the inner product of the unit vector and all observations $\{x\}_{i=1}^n$, denoted $\langle u_l, x_i \rangle$, which maps the observations to $\mathbb{R}$. This defines our projected distribution slice, which we write $u_l^* p_X$ over the measure p_X, and is used to calculate the Wasserstein distance. By doing this multiple times with different random directions, we obtain a set of 1-dimensional Wasserstein distances that can be averaged to efficiently approximate the high-dimensional Wasserstein distance.

More formally, let p_X and q_Y be two probability measures defined on $\mathbb{R}^d$. The Sliced-Wasserstein distance is defined as follows:

$$\mathcal{SW}_p(p_X, q_Y) = \left(\frac{1}{L} \sum_{l=1}^{L} \left(\mathcal{W}_p(u_l^* p_X, u_l^* q_Y) \right)^p \right)^{1/p}, \tag{2.13}$$

where u_l is a random unit vector of size $1 \times d$ sampled uniformly from a unit sphere $S = \{\theta \in \mathbb{R}^d : ||\theta|| = 1\}$ and $u_l^* p_X$, $u_l^* q_Y$ denote the projections of p_X and q_Y onto the space defined by u_l. The Sliced-Wasserstein distance computes the Wasserstein distance between the 1-dimensional projections of p_X and q_Y along

L random directions and takes the p-th root of the average of the L distances. The number of random directions L determines the accuracy of estimation, where more is better. Letting $L \to \infty$ has been shown to make the $\mathcal{SW}$ estimate converge to the true Wasserstein distance.

Necessary details and facts to understand exactly why this is true are out of the scope of this thesis but provided by Nadjahi [26] on page 40.

The important trick that makes this method work is that projecting the samples onto a 1-dimensional space allows us to utilize the closed-form formula for univariate distributions (equation 2.10). The Wasserstein distance can then be efficiently calculated for each projected distribution and averaged. Algorithm 1 presents a relatively straightforward approach to computing the Sliced-Wasserstein and gives a clear overview of how the projections are obtained and used.

Algorithm 1 Algorithm to calculate the Sliced-Wasserstein.

$\mathcal{SW} = 0$
for $l = 1, \cdots, L$ **do**
    Sample u_l uniformly from the unit sphere S
    for $i = 1$ to n **do**
        Project: $x_i' = \langle u_l, x_i \rangle$, $y_i' = \langle u_l, y_i \rangle$
    end for
    Sort: $x'(1) \leq x'(2) \leq \cdots \leq x'(n)$, $y'(1) \leq y'(2) \leq \cdots \leq y'(n)$
    $\mathcal{SW} \leftarrow \mathcal{SW} + \frac{1}{n} \sum_{i=1}^{n} |x'(i) - y'(i)|^p$
    $\mathcal{SW} = (\mathcal{SW}/L)^{1/p}$
    return $\mathcal{SW}$
end for

One of the downsides of this method is its stochastic nature. The estimation accuracy depends on the uniform sampling of unit vectors, which will yield varying estimates over different runs. As such, this method may not be ideal for situations that require deterministic calculations, e.g., when comparing GAN performance. There exists, however, an estimate of the Sliced-Wasserstein that is deterministic and faster to compute called the *Fast approximation of the Sliced-Wasserstein distance*.

2.3.4 Fast approximation of the Sliced-Wasserstein distance

The Fast approximation of the Sliced-Wasserstein distance (SW) is obtained by sampling projection vectors from a standard Gaussian distribution instead of uniformly on a unit sphere. Using the central limit theorem, the sampling of many such vectors has been shown by Reeves [36] to converge to a Gaussian distribution. Using this result, the non-deterministic Sliced-Wasserstein can be reformulated to consider the distance between two normal distributions, thus allowing the closed-form formula (equation 2.9) for calculating the Wasserstein distance to be used.

The details and proof as to why this is true are left out as it is considered out of scope for this thesis, but we will explain the main concept and show how it is calculated. Details underlying this method are provided on page 101 in Nadjahi [26].

Equation 2.14 represents the Fast approximation of the Sliced-Wasserstein distance, denoted $\widehat{SW}_2(\mu_d, \nu_d)$. The approximation contains two terms, the first being the Wasserstein distance between two Gaussian distributions, and the second term is the squared Euclidean distance between the means of the probability measures μ_d and ν_d, denoted by $\mathbf{m}_{\mu_d}$ and $\mathbf{m}_{\nu_d}$, respectively.

$$\widehat{SW}_2(\mu_d, \nu_d) = \mathcal{W}_2\left(\mathcal{N}\left(0, \frac{1}{d}m_2(\bar{\mu}_d)\right), \mathcal{N}\left(0, \frac{1}{d}m_2(\bar{\nu}_d)\right)\right) + \frac{1}{d}\|\mathbf{m}_{\mu_d} - \mathbf{m}_{\nu_d}\|^2 \tag{2.14}$$

The first term is a consequence of the Sliced-Wasserstein distance approaching a normal distribution when sampling projection vectors from a Gaussian. Calculating the Wasserstein between two normal mean-centered probability measures $\bar{\mu}_d$ and $\bar{\nu}_d$ is then used as an estimate. The estimation error is shown by Nadjahi [26] to approach 0 as $d \to \infty$, but fast enough for the estimator to be useful. The means of these Gaussian distributions are set to zero, while the covariance matrix is set to $\frac{1}{d}m_2(\bar{\mu}_d)$ and $\frac{1}{d}m_2(\bar{\nu}_d)$, respectively. Here, $m_2(\xi)$ represents the covariance of the probability measure ξ, while $\bar{\xi}$ is the centered version of the probability measure $\xi \in \{\mu_d, \nu_d\}$.

To compute the values of m_ξ and $m_2(\xi)$, the mean and covariance of the probability measure ξ containing "n" samples are calculated using equations (1) and (2). Equation (1) involves calculating the mean of the probability measure ξ, while equation (2) involves calculating the covariance used in the normal distribution. Since the covariance is calculated for mean-centered probability measures, it simplifies to being the sum of the squared elements (squared Euclidean distance). The resulting value of $m_2(\xi)$ is then divided by the number of dimensions d.

$$m_\xi = \frac{1}{n}\sum_{j=1}^{n} x_j \qquad (1) \qquad\qquad m_2(\xi) = \frac{1}{n}\sum_{j=1}^{n} \|x_j\|^2 \qquad (2)$$

To provide a clearer understanding of how this is calculated, Algorithm 2 provides an example for calculating the Fast approximation of the Sliced-Wasserstein distance between two discrete probability measures μ_d, ν_d. Both probability measures containing "n" observations $\{\mathbf{x}_d\}_{j=1}^n$ with $\mathbf{x}_d = (x_1, ..., x_d)$ for some random variables x_i.

Algorithm 2 Central Limite Theorem approximation of the Sliced Wasserstein distance

Require: μ: tensor, shape $(n_{samples}, dim)$, samples in the source domain
Require: ν: tensor, shape $(n_{samples}, dim)$, samples in the target domain
Ensure: cost: float, Sliced Wasserstein Cost

function $m_2(X)$
    return $1/n \sum_{i=1}^{n} X_{i,:}^2$
end function
function SW APPROXIMATION(μ, ν)
    $m_\mu \leftarrow \text{mean}(\mu, \text{axis} = 0)$
    $m_\nu \leftarrow \text{mean}(\nu, \text{axis} = 0)$
    $\bar{\mu} \leftarrow \mu - m_\mu$
    $\bar{\nu} \leftarrow \nu - m_\nu$
    $W \leftarrow \left(\sqrt{\text{m}_2(\bar{\mu})} - \sqrt{\text{m}_2(\bar{\nu})}\right)^2$
    $d \leftarrow dim$
    $res \leftarrow \frac{1}{d} \left|m_\mu - m_\nu\right|^2$
    return $|W + res|_2$
end function

Chapter 3

Generative Adversarial Networks

This chapter introduces the fundamental concept and context of the framework investigated in this thesis, namely the Generative Adveserial Network (GAN). The chapter is divided into three sections, starting with an introduction of the GAN framework, followed by a formal definition where we additionally motivate and assemble its objective function, and finally, a review of challenges associated with training them along with some proposed solutions. The sections build on content from the preliminary project by Kohmann [37] but are reformulated and with additional material.

3.1 Overview

A Generative Adveserial Network (GAN) is a deep learning framework that was first introduced in 2014 by Goodfellow *et al.* [1] as a new generative model to generate realistic samples. The method utilizes two models, typically neural networks, that compete with opposing objectives. By having the models compete, the optimization process can be interpreted as a game and designed using game theory such that the best strategy is to produce samples that match the desired target samples.

One of the models is referred to as the generator (G) and is the model which creates new samples. The generator takes as input random noise and produces a new sample that ideally resembles the target dataset. The other model is referred to as the discriminator (D), which aims to distinguish between real samples from the target dataset and fake samples produced by the generator. The discriminator takes as input a sample and outputs a probability score that indicates the likelihood of the sample being real or fake. The generator and discriminator are trained in an alternating fashion, like many two-player games. The generator attempts to produce samples that fool the discriminator into thinking they are real, while the discriminator tries to classify the samples as real or fake correctly. As the training progresses, the generator improves its ability to produce realistic samples, while the discriminator improves its ability to distinguish between real and fake

samples. Once the training is complete, the generator can be used to produce new samples that are similar to the target dataset.

Although the name of the framework state that they are "adversarial", they are instead the opposite, very cooperative. The term "adversarial" is used because GANs can be effectively analyzed using game theory tools.

3.2 Formal definition

A GAN consists of three components: a collection of target samples and two distinct neural networks, a generator, and a discriminator. Considering the data domain $\mathcal{X} \subset \mathbb{R}^d$, the target data can be thought of in probabilistic terms as a collection of n samples $\{\mathbf{x}_i\}_{i=1}^n$ from a probability distribution $p_{\text{target}}(\mathbf{x})$, where $\mathbf{x}_i \sim p_{\text{target}}(\mathbf{x})$. The probability distribution is considered inaccessible, with only the target samples being available, but defined to consider all possible samples over the data domain with probability mass only distributed to the target samples and their reasonable variations.

The generator is a neural network that can be represented as a function $G : \mathcal{Z} \to \mathcal{X}$ from noise space $\mathcal{Z} \subset \mathbb{R}^r$ to the space of all possible samples $\mathcal{X}$. Its purpose is to map random noise to the data domain, which should ideally resemble the target samples. The noise vector $\mathbf{z}$ is drawn from a selected probability density distribution $p_z = \mathcal{N}(0, \mathbf{I}_r)$, where $\mathbf{I}_r$ denotes the identity matrix of size $r \times r$ [1]. Having the generator constantly adapt to different noise vectors is what gives the generator the ability to generalize and produce new, diverse samples. Also, utilizing a simple probability distribution to sample noise vectors causes the generation of new samples to be very easy and is what is considered generative in this framework.

The generation of samples can be thought of as following an implicit probability distribution with density $p_{\text{gen}}(\mathbf{x})$, where "gen" is short for generated. This probability distribution captures the distribution of the generated samples over the same data domain $\mathcal{X}$ through the mapping defined by $G(\mathbf{z})$, where $\mathbf{z} \sim p_z$.

Having two probability densities over the same data domain provides a basis for evaluating how well the generator performs. Specifically, the generator aims to produce samples that approximate the target probability distribution such that $p_{\text{gen}}(\mathbf{x})$ closely matches $p_{\text{target}}(\mathbf{x})$. When this is achieved, the generator can be efficiently used as a process to obtain new samples. The probability distributions do not initially match each other, so tuning the generator network to gradually produce samples closer to the target distribution is required. Assessing how similar two inaccessible probability distributions are is a very challenging task but can be estimated using a neural network, which is the purpose of the discriminator.

The discriminator is a function $D : \mathcal{X} \to [0, 1]$ from the data domain to the set of real values between 0 and 1. It acts as a binary classifier (considering the

[1] Other probability distributions, such as a uniform distribution between -1 and 1, have been considered in the literature. They can be selected arbitrarily but should, most importantly, be fast and simple to sample from.

official GAN introduced by Goodfellow *et al.* [1]) whose objective is to *discriminate* between target and generated samples. For every sample, it produces a single scaler representing the confidence of the sample belonging to the target (=1) or the generated (=0) distribution. The confidence information of the discriminator can be used to obtain a dissimilarity measure between the two probability distributions. We will take a step back to understand why this is true and define what it means and how to measure the similarity between probability distributions.

Let us consider two probability distributions represented by their densities, $p_{\text{target}}(\mathbf{x})$ and $p_{\text{gen}}(\mathbf{x})$. Our objective is to evaluate how similar these distributions are to each other so that we can adjust the generator network to make $p_{\text{gen}}(\mathbf{x})$ more similar to the target distribution. In theory, we can achieve this for every $\mathbf{x} \in \mathcal{X}$ by comparing the two probability densities using either the density difference method, where $r^*(\mathbf{x}) = p_{\text{target}}(\mathbf{x}) - p_{\text{gen}}(\mathbf{x})$, or the density ratio method, which is considered in the original GAN:

$$r^*(\mathbf{x}) := \frac{p_{\text{target}}(\mathbf{x})}{p_{\text{gen}}(\mathbf{x})} \tag{3.1}$$

The density ratio is a typical quantity that is commonly encountered in a variety of statistical divergences [28], for instance, in the Kullback-Leibler divergence (equation 2.3).

The density ratio of an arbitrary random vector $\mathbf{x}$ indicates the extent to which we should modify $p_{\text{gen}}(\mathbf{x})$ to match $p_{\text{target}}(\mathbf{x})$. This knowledge can be utilized to improve the generated sample by guiding the learning of the generative network to minimize this quantity. Despite appearing as a novel technique, the density ratio serve as a significant tool in different branches of statistics and machine learning [38–40].

Calculating the density ratio explicitly is not possible since $p_{\text{target}}(\mathbf{x})$ and $p_{\text{gen}}(\mathbf{x})$ are unavailable to us when working with GANs. Getting around this problem is achievable by directly estimating the density ratio $r^*(\mathbf{x})$ without ever needing to assess $p_{\text{target}}(\mathbf{x})$ and $p_{\text{gen}}(\mathbf{x})$ separately. This turns out to be much easier, and several methods for doing so exist [41].

Estimating the density ratio can be formulated as a probabilistic classification task [39, 41]. By training a discriminator to distinguish between generated and target samples efficiently, the density ratio can be formulated, in probabilistic terms, as a conditional probability $P(Y = y|\mathbf{x})$ with the sample being generated divided by the complementary probability:

$$r^*(\mathbf{x}) = \frac{P(y = 0|\mathbf{x})}{1 - P(y = 0|\mathbf{x})} = \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} \tag{3.2}$$

Here Y indicates whether $\mathbf{x}$ comes from the generated (with $y = 0$) or target distribution (with $y = 1$). The probability function P assigns the probability to the random variable originating from the generator, which is what discriminator D also predicts. This avoids the need for computing the individual densities,

which was previously required (equation 3.1), and only depends on a binary classifier D. Mohamed and Lakshminarayanan [41] and Tiao [42] provides a detailed derivation of this fact.

The final ingredient needed in the GAN framework is constructing an objective function for training the neural networks. We will motivate and define the objective function for the generator and discriminator as introduced by Goodfellow *et al.* [1].

3.3 Mathematical derivation of the objective function

As the discriminator is simply a binary classifier, using the Binary Cross Entropy (BCE) to measure the classification loss is a natural choice. The BCE loss formula is defined as:

$$BCE(\hat{y}, y) = -y \log(\hat{y}) - (1-y)\log(1-\hat{y}), \tag{3.3}$$

where the ground truth label for a sample is represented by $y = \{0, 1\}$, while the predicted probability that y is the true label is represented by $\hat{y} \in [0, 1]$.

The discriminator aims to maximize its ability to correctly classify samples drawn from the target distribution $p_{\text{target}}(\mathbf{x})$ as real. This can be achieved by minimizing the BCE loss. To do this, the ground truth label is set to 1, and the predicted label is set to the discriminator's output for the sample, denoted $\hat{y} = D(\mathbf{x})$. This results in the following expression for the BCE loss:

$$\begin{aligned} BCE(D(\mathbf{x}), 1) &= -1 \log D(\mathbf{x}) - (1-1)\log(1-D(\mathbf{x})) \\ &= -\log D(\mathbf{x}) \end{aligned} \tag{3.4}$$

On the other hand, for generated samples $G(\mathbf{z})$, the discriminator aims to minimize the probability of incorrectly classifying them as real. This can be achieved by minimizing the discriminator's output for the generated samples, denoted as $D(G(\mathbf{z}))$, to be as close to zero as possible. In the BCE loss, this can be achieved by setting the ground truth label to 0 and the predicted label to $D(G(\mathbf{z}))$. The expression for the BCE loss for generated samples is thus:

$$\begin{aligned} BCE(D(G(\mathbf{z})), 0) &= -0 \log(D(G(\mathbf{z}))) - (1-0)\log(1-D(G(\mathbf{z}))) \\ &= -\log(1-D(G(\mathbf{z}))) \end{aligned} \tag{3.5}$$

By merging equation 3.4 and 3.5, we can establish a value function $\mathcal{V}$ that takes into account both situations. To maintain consistency with the original framework and make notation easier, we use the negation of both BCE equations as such:

$$\begin{aligned} \mathcal{V}(D, G) &= -BCE(D(\mathbf{x}), 1) - BCE(D(G(\mathbf{z})), 0) \\ &= \log(D(\mathbf{x})) + \log(1-D(G(\mathbf{z}))) \end{aligned} \tag{3.6}$$

Notice that $\mathcal{V}$ depends on two types of samples - real and generated - making it a function also dependent on network parameters defining D and G. Since we negated the loss for the single cases, the discriminator would now aim to maximize this quantity, precisely $\max_D \mathcal{V}(D, G)$, while keeping the parameters of G constant.

To account for the general loss and not just individual samples, the value function is expanded by evaluating the expected loss for all samples. The general value function can be represented by applying the expected value to the relevant terms with their corresponding probability densities. Theoretically, optimizing this quantity for a fixed generator G should lead to the optimal discriminator. This is the true theoretical quantity the discriminator would like to maximize:

$$\mathcal{V}(D, G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{target}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))] \tag{3.7}$$

$$\text{Discriminator objective} : \max_D \mathcal{V}(D, G) \tag{3.8}$$

When turning theory into practice, an estimation of the value function is instead considered. This is because it is impossible to calculate the expected value across the whole data domain. So to obtain an estimate and for efficiency, Monte Carlo sampling is used to approximate the expected values by averaging over a subset of samples (also described as a minibatch).

The generator's objective function is the opposite of the discriminator, meaning that it aims to minimize the discriminator's objective and make it perform poorly. This can be expressed mathematically as

$$\text{Generator objective} : \min_G \max_D \mathcal{V}(D, G) \tag{3.9}$$

The only aspect of the value function that the generator can control is $\mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]$, which necessitates generating samples that cause $D(G(\mathbf{z}))$ to be high. An approach to achieving this is by producing samples that resemble the target samples.

The generator and discriminator models improve iteratively through the value function as the generator attempts to fool the discriminator, and the discriminator improves its ability to distinguish between the samples. This cycle enables both models to improve progressively and has been proven to converge in an ideal theoretical world[2]. When the generator has learned to fully capture the target distribution, making generated samples indistinguishable from real samples, Goodfellow *et al.* [1] showed that the discriminator would guess 50% for every sample. However, expecting this behavior in practical applications is not realistic [43].

It is not obvious, but this also corresponds to minimizing the dissimilarity between $p_{\text{target}}(\mathbf{x})$ and $p_{\text{gen}}(\mathbf{x})$. To illustrate this, we'll consider the optimal discriminator (shown by Goodfellow *et al.* [1]), denoted $D^* = \frac{p_{\text{target}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}}(\mathbf{x})}$, meaning it can distinguish the samples the best possible way. Plugging the optimal dis-

[2]This is based on two main assumptions. 1) The capacity of the discriminator and generator is sufficient, 2) the discriminator is considered optimal for every generator update[1].

criminator into the objective function of the generator (equation 3.9), $\min_G \mathcal{V}(D^*, G)$, allows us to rewrite it:

$$\begin{aligned}
\min_G \mathcal{V}(D^*, G) =& \mathbb{E}_{\mathbf{x}\sim p_{\text{target}}(\mathbf{x})}[\log D^*(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1 - D^*(G(\mathbf{z})))] \\
=& \mathbb{E}_{\mathbf{x}\sim p_{\text{target}}(\mathbf{x})}[\log D^*(\mathbf{x})] + \mathbb{E}_{\mathbf{x}\sim p_{\text{gen}}}[\log(1 - D^*(\mathbf{x}))] \\
=& \mathbb{E}_{\mathbf{x}\sim p_{\text{target}}(\mathbf{x})}\left[\log \frac{p_{\text{target}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}}(\mathbf{x})}\right] + \\
& \mathbb{E}_{\mathbf{x}\sim p_{\text{gen}}(\mathbf{x})}\left[\log \frac{p_{\text{gen}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}}(\mathbf{x})}\right] \\
=& \int_{\mathcal{X}} p_{\text{target}}(\mathbf{x}) \log \frac{p_{\text{target}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}}(\mathbf{x})} d\mathbf{x} + \\
& \int_{\mathcal{X}} p_{\text{gen}}(\mathbf{x}) \log \frac{p_{\text{gen}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}}(\mathbf{x})} d\mathbf{x}
\end{aligned} \tag{3.10}$$

Using a formulation of the Jensen-Shannon divergence (D_{JS}), which is a symmetric Kullbeck-Leibler divergence (equation 2.3), it is possible to derive the reformulated objective of the generator (equation 3.10) plus some additional constants:

$$\begin{aligned}
D_{\text{JS}}(p_{\text{target}} \parallel p_{\text{gen}}) =& \frac{1}{2} D_{\text{KL}}(p_{\text{target}} \parallel \frac{p_{\text{target}} + p_{\text{gen}}}{2}) + \frac{1}{2} D_{\text{KL}}(p_{\text{gen}} \parallel \frac{p_{\text{target}} + p_{\text{gen}}}{2}) \\
=& \frac{1}{2}\left(\int_{\mathcal{X}} p_{\text{target}}(\mathbf{x}) \log \frac{2p_{\text{target}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}(\mathbf{x})}(\mathbf{x})} d\mathbf{x}\right) + \\
& \frac{1}{2}\left(\int_{\mathcal{X}} p_{\text{gen}}(\mathbf{x}) \log \frac{2p_{\text{gen}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}(\mathbf{x})}(\mathbf{x})} d\mathbf{x}\right) \\
=& \frac{1}{2}\left(\log 2 + \int_{\S} p_{\text{target}}(\mathbf{x}) \log \frac{p_{\text{target}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}(\mathbf{x})}(\mathbf{x})} d\mathbf{x}\right) + \\
& \frac{1}{2}\left(\log 2 + \int_{\mathcal{X}} p_{\text{gen}}(\mathbf{x}) \log \frac{p_{\text{gen}}(\mathbf{x})}{p_{\text{target}}(\mathbf{x}) + p_{\text{gen}(\mathbf{x})}(\mathbf{x})} d\mathbf{x}\right) \\
=& \frac{1}{2}\left(\log 4 + \min_G \mathcal{V}(D^*, G)\right)
\end{aligned} \tag{3.11}$$

By rearranging the terms and isolating the generator's objective, we obtain the following:

$$\min_G \mathcal{V}(D^*, G) = 2D_{\text{JS}}(p_{\text{target}} \parallel p_{\text{gen}}) - \log 4, \tag{3.12}$$

where the Jensen–Shannon divergence measures the dissimilarity between the target and generated probability densities. The Jensen-Shannon divergence is always greater than or equal to zero and only equal to zero when the target and generated distributions are the same, meaning $p_{\text{target}} = p_{\text{gen}}$. This implies that for

the generator to minimize the value function, it must also minimize the dissimilarity between the generated and target samples, which we wanted to show.

Figure 3.1 provides an overview of the concepts related to the GANs framework and the gradients used for updating the neural networks. The generator and discriminator are identified as having parameters θ_g and θ_d, respectively. The gradients used for updating the networks are obtained by differentiating the value function with respect to the appropriate network parameters. During the gradient calculation, the parameters of the opposing network are held constant, similar to an opposing player remaining still when a player is contemplating a "move."

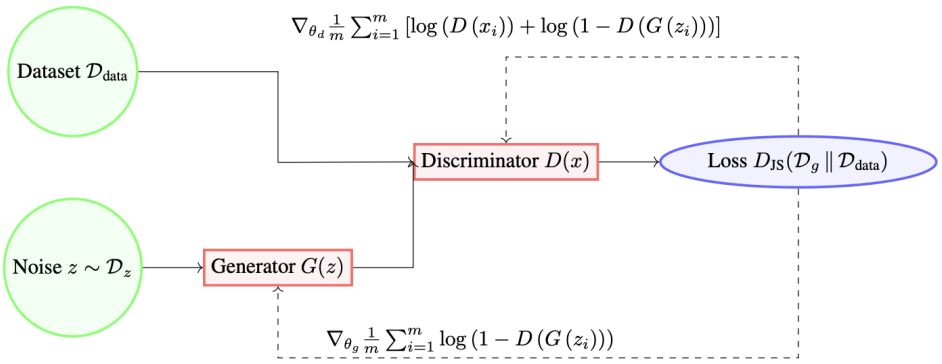


Figure 3.1: A GANs illustration with the forward and backward pass marked as continuous and dotted lines, respectively. The gradients used to update each network are provided on the dotted lines. The expected values are replaced with a Monte Carlo sampling of m samples to estimate the value function.

3.4 Challenges

Despite the promising concept of GANs, they are known for their difficulty in training. This can be attributed to multiple factors, such as the design of the loss function, sensitivity to specific architectures, and the conflicting objective between the generator and discriminator [44]. These instabilities pose significant challenges in developing practical applications, highlighting the need for research to address these obstacles. Since the initial framework was published in 2014, there have been various proposed modifications to make the framework more robust and controllable. Hence, a short review of some proposed solutions will be covered here to understand when GANs fail and how to avoid common mistakes. Such knowledge will be essential when selecting a GAN framework and making them produce the desired results. Much of the content concerning this chapter is based on the preliminary project by Kohmann [37].

3.4.1 Vanishing gradients and instability

The generator objective function presented earlier (equation 3.9) is unstable in practice, as it provides poor-quality, or none, gradients to the generator when the discriminator is too confident. This is often the case in the early stages of training when the generator produces low-grade samples, and the discriminator is trained too quickly such that it manages to distinguish between target and generated samples with high confidence. This causes $\log(1-D(G(z)))$ to saturate and leaves the generator with subtle gradients to update on [1].

To address this issue, Goodfellow *et al.* [1] proposed a new objective for the generator while keeping the discriminator's objective unaltered. Instead of minimizing $\log(1 - D(G(z)))$, they suggested that the generator could rather maximize the failure of the discriminator to recognize generated samples as fake, $\max_G \log D(G(z))$. This new generator objective yields more rich gradients, enabling it to learn effectively even with a confident discriminator. While this solution somewhat mitigates the problem, it is purely heuristically motivated and not based on any theoretical foundation. As a result, several other alternative solutions have been proposed.

The most influential proposed solution has been to adopt a different statistical distance as the objective function, namely the Wasserstein distance (equation 2.6). The Wasserstein distance is always defined, unlike the Jensons-Shannon divergence used in the original GAN, which is only defined if the two probability densities overlap. This completely avoids the vanishing gradient problem for the generator and serves as one of the most influential proposed solutions to the GAN framework.

Arjovsky *et al.* [45] was one of the first to successfully implement the Wasserstein distance into the objective function. It was possible due to a reformulation of the Wasserstein 1-distance (the p-th root equal to 1), referred to as the Kantorovich-Rubinstein duality [46]:

$$\mathcal{W}(p_X, q_Y) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_X}[f(\mathbf{x})] - \mathbb{E}_{\mathbf{y}\sim q_Y}[f(\mathbf{y})] \tag{3.13}$$

With this new approach, the discriminator is seen as an estimator of the best cost function f rather than a classifier. Unlike previous GAN models where the discriminator output is bounded between 0 and 1, this new model considers the discriminator as a critic, estimating a function f instead. The function f represents the cost of loading and unloading mass at the position defined by $\mathbf{x}$ and $\mathbf{y}$. The function must be 1-Lipschitz continuous (represented as $\|f\|_L \leq 1$), meaning it has to be smooth and not change rapidly (the derivative of f at any point is less than or equal to 1).

The redefined discriminator changes the training process of the GAN. It is now more beneficial to update the discriminator (critic) multiple times before updating the generator. A more optimal discriminator (critic) provides a better approximation of the Wasserstein distance, hence more accurate gradients for the generator.

Although the Wasserstein distance has gained significant recognition as an objective function for GANs, it is not without flaws. Several alternative objective functions, some of which are based on the same theory as Wasserstein, have been proposed. However, there is still no agreement on which objective function is the most practical, and selecting a suitable one remains an unresolved issue [44].

However, various straightforward solutions exist to overcome the vanishing gradient problems in GANs. For example, a strategy suggested by Salimans *et al.* [47] proposes modifying the BCE loss function (equation 3.4) by using a slightly smaller value than 1, such as 0.9, as the real target label. This method helps prevent the discriminator from being too confident in its predictions. Another way to reduce the confidence of the discriminator is by adding Gaussian noise to both the target and generated samples and gradually decreasing the amount of noise over time to stabilize the training process. According to Sønderby *et al.* [48], this technique can prevent the early overfitting of the discriminator.

3.4.2 Mode collapse

Mode collapse, or mode drop, occurs in GANs when the generator only produces a limited subset of the modes or classes in the target distribution. When this happens, the generator has learned to generate only parts of the target distribution, leaving out the specific regions or modes we want to produce. The reason behind this behavior is that the generator is able to trick the discriminator with some generated samples, which the discriminator cannot distinguish from the real ones, and then continue only to generate these samples. Since the generator has no reason to switch to or generate other modes or classes, the GAN will only optimize for a subset of the modes or classes. This is equivalent to the minimax game between the discriminator and the generator ending up in a local Nash equilibrium. All other strategies would result in a worse state for both players [49].

One of the significant issues with mode collapse is that it occurs often and unexpectedly. This has made it an important research topic with numerous proposed solutions to tackle this issue, introducing fixes for almost every component in the GAN framework. In one such solution, Arjovsky *et al.* [45] introduced the Wasserstein distance as a new objective function to achieve more stable gradients for the generator. However, they also found that their GAN became more prone to mode collapse, which they attributed to the quality of gradients that the critic (discriminator) produced for the generator. This led to the suggestion that the robustness of the critic is linked to mode collapse [44, 45, 50].

Liu *et al.* [51] investigated the robustness of the discriminator in GANs more closely and found that the spectral distribution of the weights in the discriminator, precisely the singular values of $\overline{W}_{SN}(W)$ (with W representing a weight-matrix), was strongly linked to mode collapse. They noticed that mode collapse caused a dramatic drop in the singular values, a phenomenon they named spectral collapse. To address this problem, they introduced a spectral regularization term in the objective function, which prevented spectral collapse and improved performance.

Kodali *et al.* [43] investigated the reasons behind mode collapse in GANs by analyzing their convergence behavior. They discovered that the discriminator produces sharp gradients around some target sample points, encouraging the generator to make the same output for slightly different noise vectors. To overcome this issue, they proposed a gradient penalty term to avoid moving quickly to local equilibria, which improved the GAN stability.

Similarly, Durall *et al.* [52] tackled the problem of local equilibria using a different approach. They found that gradients near local equilibria produce high eigenvalues in the Hessian of the loss, which is a sign of premature mode collapse. To address this, they developed the NuGAN optimizer, which uses second-order derivative information to avoid local Nash equilibria and prevent mode collapse by steering away from such steep gradients.

Other proposed approaches to combat mode collapse include using multiple generators (MGAN)[53], conditioning on labels (cGAN)[54] and unrolling to see future behaviors (Unrolled-GAN)[55]. Saxena and Cao [44] lists many other and unique methods but acknowledges that there are still several unsolved issues and few GAN frameworks are robust against all errors.

Despite the numerous proposed solutions, mode collapse is not yet fully understood. Since most research on this topic has been in the image domain, it is unclear whether these solutions will be effective in other domains, such as sequential data. Nevertheless, since many of the proposed solutions are not domain-specific, there is potential to modify and test these methods for different types of GANs.

3.4.3 Other factors

In addition to the problems discussed is their sensitivity to architectural design and hyperparameter choice. Slight changes to the initial weights or learning rate can significantly affect the model's performance, and it is not clear what the optimal settings should be. Therefore, training GANs often involves much trial and error in finding a good combination of hyperparameters, which makes it common for the model to fail or produce unsatisfactory results.

Chapter 4

Related work

This chapter presents the literature review on two main subjects: GANs for time series generation and previous studies focusing on constrained and physics-informed GANs. The first topic discusses various significant frameworks that enable the generation of time series by employing different objective functions and architectural designs, aiming to present a comprehensive overview of viable methods. The second topic provides a summary of past research in this area, highlighting existing knowledge and identifying areas that require further investigation.

4.1 GANs for time series generation

In this section, we will cover several different architectures that have achieved noteworthy results concerning their date of publicity and improvement over previous state-of-the-art designs. Each framework has a focus on architecture design choices, objective function, and evaluation metrics used. Much of this content was also present in the preliminary project [37].

4.1.1 Continuous-RNN-GAN (C-RNN-GAN) (2016)

One of the earliest attempts to extend the GAN framework to operate on sequential data was by Mogren [56] in an attempt to generate new classical music. They modified the original framework to function on continuous real-valued time sequences and incorporated modules to identify temporal patterns, which enabled them to create new musical compositions.

The model architecture was relatively straightforward, comprising a generator with an Long Short-Term Memory (LSTM) and a discriminator with a bidirectional LSTM. The output of each LSTM cell in the discriminator was passed through a linear layer with sigmoid activation to determine the confidence level for the classification of the generated sample at each time step and then return the average confidence. This means the discriminator $D : \mathbf{X}^T \rightarrow [0, 1]$ returns a single value for each sample $\mathbf{x}_{1:T}$, where T is the sequence length. They then used the original BCE objective functions to train the models:

$$\begin{aligned} \max_D \mathcal{V}(D,G) &= \mathbb{E}_{x_{1:T}\sim p_{\text{target}}}[\log D(\mathbf{x_{1:T}})]] + \mathbb{E}_{z_{1:T}\sim p_z}[\log(1-D(G(\mathbf{z_{1:T}})))] \\ \max_G \mathcal{V}(D,G) &= \mathbb{E}_{z_{1:T}\sim p_z}[\log(D(G(\mathbf{z_{1:T}})))] \end{aligned} \tag{4.1}$$

Despite facing obstacles such as vanishing gradients and mode drop in the generated samples, Mogren [56] employed simple techniques such as freezing the discriminator when it became too confident, as suggested by Salimans *et al.* [47], to tackle these challenges. The generated samples were evaluated primarily using subjective music-specific metrics and listening impressions. Although the results were not exceptional, Mogren [56] demonstrated that the model could capture important components in the training data, which inspired further exploration of the idea.

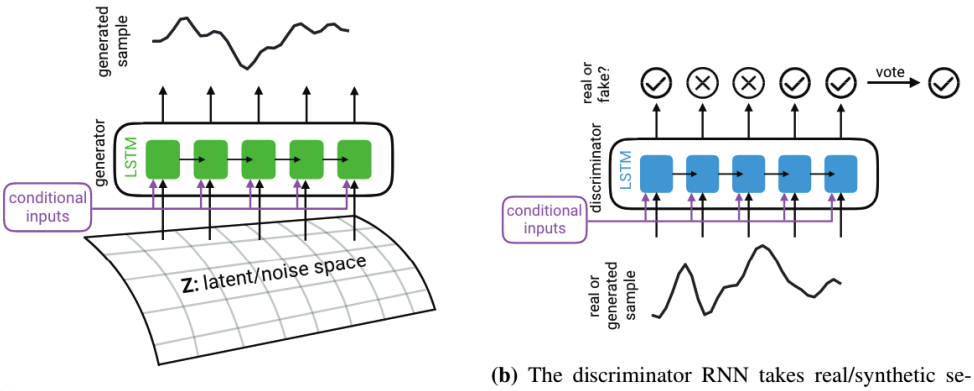
4.1.2 Recurrent GAN (RGAN) (2017)

A succeeding study that received much attention and became the state-of-the-art architecture for generating time series was Recurrent GAN (RGAN) by Esteban *et al.* [57]. Esteban *et al.* [57] was the first to investigate the generation of real-valued multivariate time series and proposed evaluation metrics to measure the quality and diversity of the generated samples.

They modified the C-RNN-GAN architecture by replacing the recurrent modules in the generator and discriminator with stacked uni-directional LSTMs followed by a single dense layer. The discriminator utilized the sigmoid function on each time step, similarly as Mogren [56], but Tanh in the generator and training using the same objective function as Mogren [56] (equation 4.1).

One of the main contributions of Esteban *et al.* [57] was to expand the use of labels in order to enhance the efficiency of learning the target distribution and to have more control over the generation process. This concept was first demonstrated on image data by Mirza and Osindero [54]. For time series data, Esteban *et al.* [57] proposed a solution by concatenating the label c to each noise and target data time step. This was achieved by using $\mathbf{x}_{1:T} = (\mathbf{x}_i, c_i)_{i=1}^T$, and $\mathbf{z}_{1:T} = (\mathbf{z}_i, c_i)_{i=1}^T$, where $\mathbf{x}_i$ and $\mathbf{z}_i$ are random vectors with the same dimensions as the feature and noise dimensions, respectively. This had the same effect of adding an extra data dimension to the time series and did not require modification to train using the same objective function. The conditional framework was named Recurrent Conditional GAN (RCGAN), illustrated in Figure 4.1.

Esteban *et al.* [57] conducted experiments using their architecture on increasingly complex datasets, incorporating temporal correlations and multivariate sequences. The datasets included independent sinus waves with varying frequencies and phases, arbitrary smooth signals with local correlations, MNIST represented as multivariate sequences (treating each 28x28 image as 28 sequences of length 28), and real ICU (Intensive Care Unit) data. To address the lack of robust evaluation metrics in prior research, the authors introduced two novel metrics



(a) The generator RNN takes a different random seed at each temporal input, and produces a synthetic signal. In the case of the RCGAN, it also takes an additional input on each time step that conditions the output.

(b) The discriminator RNN takes real/synthetic sequences and produces a classification into real/synthetic for each time step. In the case of the RCGAN, it also takes an additional input on each time step that conditions the output.

Figure 4.1: The architecture of RGAN/RCGAN as presented in the paper by Esteban *et al.* [57].

for assessing the generated samples: the Maximum Mean Discrepancy (MMD), a statistical test for comparing distributions, and the Train-on-Synthetic-Test-on-Real (TSTR), which measures the classification difference between models trained and tested on synthetic versus real data. By employing these evaluation metrics and presenting useful distribution plots of the generated data, the authors demonstrated that their models successfully generated data that closely resembled real data on all datasets.

Esteban *et al.* [57] also demonstrated the increased performance of their architecture by comparing it to the C-RNN-GAN by Mogren [56], for learning the target distribution without additional loss tricks used in the loss function. However, it remains unclear whether this performance increase was due to conditioning on labels or the architecture itself, as a direct comparison between RGAN and RCGAN was not discussed. The authors did not provide a detailed analysis of how well the model captured the spatial and temporal relations among the multivariate sequences, relying mainly on TSTR scores to assess sample quality. Despite the architecture's simplicity and potential in generating time series data, RGAN/RCGAN is a simple and valuable helpful model for our experiment.

4.1.3 TimeGAN (2019)

Previous architectures in the field of time series generation mainly extended the standard GAN framework by incorporating temporal elements, such as using RNN [56–58] or CNN [20, 59] modules. However, Yoon *et al.* [60] took a different approach with their Time Series GAN (TimeGAN). They fused representation learning and autoregressive models within the GAN framework to better capture the complex temporal dependencies in the data.

TimeGAN was compared against several previous methods, notably C-RNN-

GAN Mogren [56] and RGAN [57] and demonstrated significant improvements in sample quality. This gained TimeGAN considerable attention and established it as one of the state-of-the-art models.

Before we present how TimeGAN operates, an important disclaimer is that version presented here is a simplified version of the original TimeGAN framework, which aimed to support a wide range of datasets, including static variables and variable time series lengths. Since our data do not possess these characteristics, we focus on the core intuition and motivation behind the architecture.

Given the high dimensionality associated with time series data, Yoon *et al.* [60] drew inspiration from works that combined autoencoders with GANs to operate on data in lower-dimensional representation space. Instead of directly operating on raw samples, TimeGAN's generator and discriminator create and classify latent representations of the target data produced by an encoder. The reconstruction of these representations back to the original feature dimension is performed by a separate recovery network within the autoencoder, distinct from the GAN components.

The autoencoder in TimeGAN consists of an embedding network and a recovery network, which facilitate the mapping between the feature space and latent representations for the GAN learning process. The embedding network, denoted as E, is implemented using a Gated Recurrent Unit (GRU) and generates a latent representation $h_{1:T} = E(x_{1:T})$ for input samples $x_{1:T}$ from the training data.

To reconstruct the input samples from their latent representations, the recovery network R performs the reverse process by mapping the latent representation back to the feature representation. This is achieved by reconstructing the input as $\hat{x}_{1:T} = R(h_{1:T})$. The architecture of the recovery network is commonly a mirrored version of the embedding network, also utilizing GRU-based modules.

To evaluate the quality of reconstruction for each sample, mean squared error is employed to measure the difference between the input $x_{1:T}$ and the reconstructed $\hat{x}_{1:T}$. This gives rise to the reconstruction loss $\mathcal{L}_R$, which is utilized for training the autoencoder. The reconstruction loss is defined as follows:

$$\min \mathcal{L}_R = \mathbb{E}_{x \sim p_{\text{data}}} \left[\sum_t \|\mathbf{x}_t - \hat{\mathbf{x}}_t\|_2 \right] \tag{4.2}$$

It is worth noting that the latent representation $h_{1:T}$ has the same length as the time series. Therefore, when specifying the number of hidden features for the latent representation, each time step t in $h_{1:T}$ will have the same number of dimensions, for instance, 20 dimensions to encode information. Additionally, the architecture used for both the embedding and recovery networks can be of any arbitrary module which adheres to causal ordering, ensuring that outputs at each step depend only on previous information. Hence, a range of modules, such as LSTMs, temporal convolutions, and transformers, can be employed.

Yoon *et al.* [60] identified a limitation in previous methods where they solely relied on the discriminator's binary adversarial feedback to learn the target distribution. They assumed this was insufficient for efficiently capturing the target

distribution because of the complexity of time series data. To address this, Yoon *et al.* [60] gave greater importance to modeling the dependencies across time, explicitly focusing on accurately representing the conditional distribution $p(x_t|x_{1:t-1})$ for each time step t.

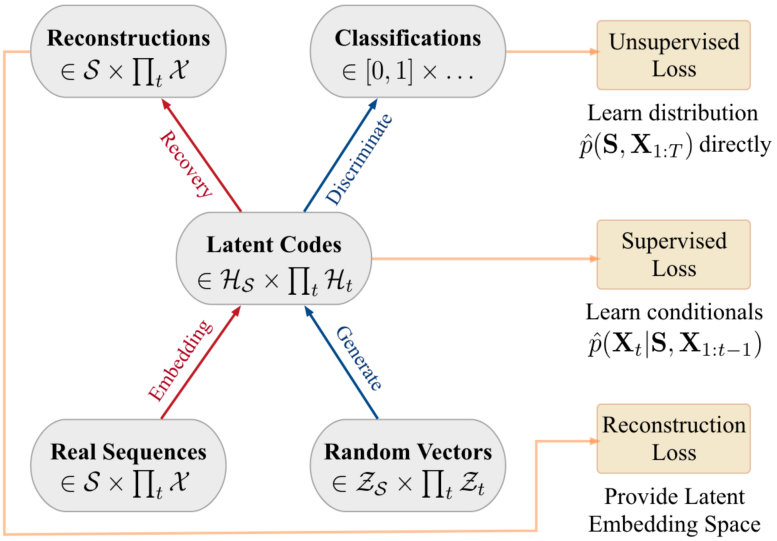


Figure 4.2: The networks that constitute the TimeGAN architecture. This illustration is sourced from the original paper[60].

An additional autoregressive network S is introduced to act as a supervisor to improve the generator's ability to capture the conditional distributions better. This network effectively learns the factorized version of the joint distribution by conditioning on previous time steps: $p(x_{1:T}) = \prod_t p(x_t|x_{1:t-1})$. Notably, the autoregressive model S learns from training data alone, which provides access to the ground truth conditional distribution $p(x_t|x_{1:t-1})$ for any time step t.

Yoon *et al.* [60] implements S using GRU modules with a final linear layer and operates on the latent representations. It takes in encoded samples and outputs the same samples recurrently. This is done using teacher forcing, a technique where the correct input is provided at each time step during training instead of using the predicted output from the previous time step. Teacher forcing accelerates the learning process and speeds up training.

A supervised loss combining the two is constructed to utilize the autoregressive model S to guide the generator to produce samples resembling the target distribution. This involves estimating the distribution $\hat{p}(x_t|x_{1:t-1})$ by feeding the model S with generated samples. The error between the true conditional distribution and the estimate is then computed using the squared error over all time steps:

$$\min \mathcal{L}_S = \mathbb{E}_{h_{1:T} \sim E(p_{\text{data}})} \left[\sum_t \| h_t - S(h_{t-1}, z_t) \|_2 \right], \tag{4.3}$$

where S represents the autoregressive model that outputs the estimate $\hat{h}_t$,

and true h_t, and computes the squared error across all time steps. Further details regarding this specific loss can be found in the research paper.

The final components of the architecture consist of the generator and discriminator networks, both implemented as GRU modules with a linear final layer and sigmoid activation. These networks operate solely with latent representations and employ a similar loss function to other mentioned architectures. However, in TimeGAN, the input sequence $\mathbf{x}_{1:T}$ is first encoded using the embedding network E, resulting in the following loss equations:

$$\begin{aligned} \max \mathcal{L}_D &= \mathbb{E}_{h_{1:T} \sim E(p_{\text{data}})} [\log D(\mathbf{h_{1:T}})]] + \mathbb{E}_{z_{1:T} \sim p_z} [\log(1 - D(G(\mathbf{z_{1:T}})))] \\ \max \mathcal{L}_G &= \mathbb{E}_{z_{1:T} \sim p_z} [\log(D(G(\mathbf{z_{1:T}})))] \end{aligned} \tag{4.4}$$

These equations represent the discriminator loss $\mathcal{L}_D$ and the generator loss $\mathcal{L}_G$, respectively. To summarize, the four losses utilized to train TimeGAN are the reconstruction loss $\mathcal{L}_R$, the supervised generator loss $\mathcal{L}_S$, the unsupervised generator loss $\mathcal{L}_G$, and the discriminator loss $\mathcal{L}_D$.

They train the models in three separate stages. The autoencoder (consisting of the embedding network E and the recovery network R) is trained first to learn proper latent representations of the target data, then the autoregressive model S. When the autoregressive model has learned a good estimate of the target distribution, all models are trained jointly. This sequential training approach ensures that the generator and discriminator begin their training with meaningful representations of the target data rather than starting with random and meaningless representations.

Yoon *et al.* [60] extensively tested their proposed framework by comparing it to previous state-of-the-art models using four different datasets. The evaluation methods used to assess the models focused on the generated samples' diversity, fidelity, and usefulness. Diversity was subjectively assessed using PCA and t-SNE, fidelity using the discriminative score, and usefulness using the predictive score.

The discriminative score quantified the classification error of an external model trained to distinguish between generated and target data. The predictive score, similar to TSTR introduced by Esteban *et al.* [57], predicts the next time-step for all time steps on real data and computes the mean absolute error as the loss.

The reported results revealed that TimeGAN consistently outperformed C-RNN-GAN in terms of generating realistic and correct samples and slightly outperformed RGAN. The improvements were attributed to incorporating a supervised loss to enhance the modeling of the conditional distribution and the utilization of different data representations.

However, it is important to note that the evaluation scores relied solely on two external neural networks, which may introduce assumptions about their design and capabilities. The lack of visual evidence showcasing the realism of the generated data is another limitation of the paper.

4.1.4 COTGAN (2020)

Xu *et al.* [61] investigated a different aspect of generating time series data not considered by previous methods, which is that time series data needs to follow a causal relationship. This means that data at each point in time should only be influenced by data from the past, which is a true phenomenon when sampling data through time. Combined with optimal transport theory and a mathematical definition of causality, they define a new learning objective that produces causal transport plans for the generator to minimize; hence they named it a Causal Optimal Transport GAN (COTGAN). They show that this objective function can be implemented to a broader range of different data types by showing models able to generate autoregressive processes and image sequences of animated game characters and human actions.

To achieve this, Xu *et al.* [61] takes a very different approach from the other proposed time series GANs as they calculate an approximated Wasserstein distance without using a neural network but directly using the Sinkhorn divergence algorithm. A property of the Sinkhorn divergence algorithm is that it allows for gradient flows such that the error can be propagated through the transport plan and reach the generator for gradient updates. This frees the discriminator, which usually is used for this calculation. Xu *et al.* [61] uses the discriminator instead for measuring the error of causality in the transport plan.

Xu *et al.* [61] starts by considering the Sinkhorn divergence (equation 2.12) and adds an additional constraint on the transport plan π to make it causal. They mathematically consider a transport plan causal $\pi \in \Pi(\mu, \nu)$ if

$$\pi(y_t|x_1, \cdots, x_T) = \pi(y_t|x_1, \cdots, x_t), \quad \text{for all } t = 1, \cdots, T-1$$

This equality states that the amount of mass transported by π from positions $x_1, \cdots, x_T$ to a single position y_t only to be dependent on the source x up to time t. Meaning that future events do not influence the result of the transport plan. To impose the causality constraint, an equivalent characterization of causality is considered

$$\pi \text{ causal} \Leftrightarrow \mathbb{E}^{\pi}[l(x, y)] = 0, \quad \text{for all } l \in \mathcal{L} \tag{4.5}$$

where $\mathcal{L}$ is some well-defined space of linear functions. This equivalent statement shifts the problem of finding causal transport plans to find instead functions $l \in \mathcal{L}$ such that the equation holds, which is more feasible. Adding the causality constraint to the regularized optimal transport (equation 2.12 with the exponent $p = 1$) gives the reformulated optimal transport:

$$\begin{aligned} \text{COT}_{c,\epsilon}(p_X, q_Y) &= \inf_{\pi \in \Pi(p_X, q_Y)} \left\{ \mathbb{E}^{\pi}[c(x, y)] - \epsilon H(\pi) + \sup_{l \in \mathcal{L}} \mathbb{E}^{\pi}[l(x, y)] \right\} \\ &= \sup_{l \in \mathcal{L}} \widehat{\mathcal{W}}_{c+l,\epsilon}(p_X, q_Y) \end{aligned} \tag{4.6}$$

where $\text{COT}_{c,\epsilon}(p_X, q_Y)$ denotes the causal optimal transport between two probability distributions, sup is the operator selecting the function $l \in \mathcal{L}$ in which

$\mathcal{W}_{c+l,\epsilon}(p_X, q_Y)$ is the largest, and c the transportation cost. Xu *et al.* [61] provides proof of why this equality holds. As they utilize the Sinkhorn divergence, ϵ and L are additional hyperparameters that control the entropic regularization and the number of iterations to use in the Sinkhorn algorithm, respectively.

The set of functions $\mathcal{L}$ serves as a test for causality, as presented in equation 4.5. An oversimplified overview using a similar notation to [61] is presented here to see how causality is measured, with details provided by Xu *et al.* [61]. Each $l \in \mathcal{L}$ is composed of two special continuous functions denoted as an ordered pair (h, M), where $h = (h_t)_{t=1}^T$ and each $h_t \in C_b(\mathbb{R}^{d\times t})$ where $C_b(\mathbb{R})$ the set of continuous bounded functions on $\mathbb{R}$. The function M measures causality with a stricter definition of being a *martingale* over the probability density p_X, precisely a martingale in the set of $\mathcal{M}(\mathcal{F}^X, p_X)$, which is the set of $(X, \mathcal{F}^X, p_X)$-martingales. A martingale can be defined as the conditional expected value over a time-dependent variable $X = (X_1, \cdots, X_T)$, which always equals the previous value: $\mathbb{E}[X_{n+1}|X_1, \cdots, X_n] = X_n$ for all $n < T$. This can be understood as the expected value of the next observation, given all the past observations, is equal to the most recent observation. The filtration $\mathcal{F}^X = \{\mathcal{F}_t^X\}_{t=1}^T$ is a filter on the probability space X that permits only information up to time t to be available at any instance, effectively filtering out information about the future.

The function M is constrained to be a martingale over the filtered probability space, denoted as $M = (M_t)_{t=1}^T \in \mathcal{M}(\mathcal{F}^X, p_X)$, with each $M_t \in C_b(\mathbb{R}^{d\times t})$. By considering the pairs (h, M) associated with each $l \in \mathcal{L}$, the causality test (equation 4.5) can be reformulated as follows:

$$\mathbb{E}^\pi\left[\sum_{t=1}^{T} h_t(y_{\leq t})\Delta_{t+1}M(x_{\leq t+1})\right] = 0 \quad \text{for all} \quad (h, M) = l \in \mathcal{L} \tag{4.7}$$

In this equation, the function h operates on the generated data, while M operates on the target data, which will be estimated using separate neural networks. This is where the discriminators come into play, with one dedicated to h and the other to M. The goal is to estimate functions (h, M) that satisfy this equation, which becomes an additional objective of the proposed method. It's important to note that understanding the intricate details of how this equation tests causality can be challenging due to the lack of clarity in Xu *et al.* [61] and the complexity of the referenced proof by Veraguas *et al.* [62].

Since equation 4.7 considers all functions L, it is approximated using a fixed number of functions denoted by the integer J. The causal optimal transport objective can then be expressed with a modified cost function that incorporates the causality test for J functions, as shown in equation 4.8.

$$c_\varphi(x, y) = c(x, y) + \sum_{j=1}^{J}\sum_{t=1}^{T} h_t^j(y_{\leq t})\Delta_{t+1}M^j(x_{\leq t+1}) \tag{4.8}$$

A martingale regularization is introduced to ensure that the martingale discriminator M satisfies the criteria. This regularization penalizes the discriminator M for

not satisfying the martingale properties. It is achieved by calculating the average non-zero value of M for a minibatch of size m:

$$p_M(x) = \frac{1}{mT} \sum_{j=1}^{J} \sum_{t=1}^{T} \left| \sum_{i=1}^{m} \frac{\Delta_{t+1} M^j(x^i)}{\sqrt{\text{Var}[M^j]} + \eta} \right| \tag{4.9}$$

where $\text{Var}[M]$ denotes the empirical variance of M over batch and time, and η is a small constant to avoid dividing by zero.

Combining equations 4.6 and 4.9, the adversarial objective function is defined as $\widehat{\mathcal{W}}_{c+l,\epsilon}(p_X, q_Y) - \lambda p_M$, which the discriminators aim to maximize while the generator aims to minimize. The λ parameter controls the importance of martingale regularization.

Additional techniques are employed to improve the estimation of Sinkhorn divergence for batch sizes. These techniques include adding bias correction terms and simultaneously computing the distance on two distinct batches.

They compare their framework to three other generative models, including TimeGAN. The comparison focuses on the generation of autoregressive processes, noisy oscillations, and brain signals (EEG). They also demonstrate COTGAN's ability to be used on a wider range of data types by generating video animations of video characters. However, their evaluation mainly relies on visual information related to the auto-correlation between data channels, which does not provide strong evidence for the performance boost of COTGAN. The discussion and model comparison in their study are considered limited and would benefit from further investigation. Nevertheless, using a unique approach in time series generation by COTGAN is intriguing and offers the possibility of addressing issues present in other frameworks.

4.1.5 RTSGAN (2021)

Real-World Time Series GAN (RTSGAN), proposed by Pei *et al.* [63], is a modern architecture specifically designed to generate realistic real-world time series (RTS) data. It addresses the challenges of supporting variable time series sequence lengths and generating time series data with missing values. Similar to TimeGAN, RTSGAN also utilizes an autoencoder network to produce compact representations of the target data. However, it differs from TimeGAN in that it allows for flexible dimensionality of the latent space independent of sequence length and does not incorporate a supervisor network. This flexibility enables RTSGAN to have a smaller latent space dimension and employ a simpler architecture utilizing only fully-connected layers for the generator and discriminator. Since they are not dependent on using recurrent layers, such as LSTMs, they also utilize the Wasserstein distance instead of the BCE losses as the objective function. Through experiments on various datasets, including comparisons with TimeGAN and RGAN, the authors demonstrate the better performance of their architecture.

It's worth noting that the RTSGAN architecture includes methods and techniques specifically tailored to handle challenges related to generating RTS data

with variable time series lengths and missing values. However, these techniques are not required for datasets that do not have missing values and where every time series have equal lengths. In such cases, a simplified version of the framework can be used, which is discussed here. Please refer to the work by Pei *et al.* [63] for the original architecture.

The design of RTSGAN was primarily driven by the objective of generating high-quality and concise latent space representations of time series data that are independent of sequence length and can be easily reconstructed. Pei *et al.* [63] achieved this by incorporating an autoencoder structure with certain modifications to the encoder component to generate feature-rich representations.

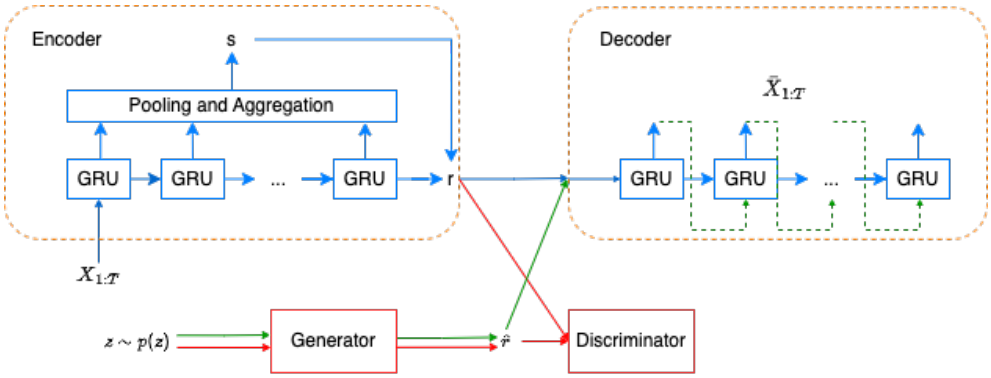


Figure 4.3: Illustration of the RTSGAN architecture. The blue lines represent the training phase of the autoencoder, while the red lines indicate the training phase of the GANs. Once the training is completed, sample generation occurs along the green path. The generated representation, denoted as $\hat{\mathbf{r}}$, is constructed autoregressively within the decoder, shown by the dashed lines. This diagram is a modified version of the original figure from Pei *et al.* [63] to align with our modified explanation.

The encoder creates latent representations through a sequence of steps. It starts with a time series sample $\mathbf{x}_{1:T}$ and passes it through an N-stacked Gated Recurrent Unit (GRU) module with a hidden dimension defined by d_{AE}. The GRU module generates hidden states $\mathbf{h}_i^n$ for each time step i and each layer n of the GRU module. The calculation of the hidden states is listed as:

$$\mathbf{h}_i^n = \text{GRU}(\mathbf{e}_i), \quad i \in [1, T], \quad n \in [1, N],$$

where T represents the length of the time series, and N is the number of stacked recurrent units (GRU modules). The encoder then applies average pooling and max pooling operations to the hidden states from the last layer of the GRUs, denoted as $\mathbf{h}_i^N$. The results of these pooling operations are concatenated with the final hidden state $\mathbf{h}_T^N$ and passed through a fully connected (FC) layer with the LeakyReLU activation function, enriching the representation as follows:

$$\mathbf{s} = \text{FC}([\text{AvgPool}(\mathbf{h}_i^N), \text{MaxPool}(\mathbf{h}_i^N), \mathbf{h}_T^N])$$

Finally, the encoder combines the enriched representation with the last hidden state of the GRU module to produce a latent representation $\mathbf{r}$ of the time series, given by:

$$\mathbf{r} = [\mathbf{s}, \{\mathbf{h}_T^n\}_{n=1}^N] \tag{4.10}$$

The resulting latent representation $\mathbf{r}$ has a dimension of $(N+1)d_{AE}$.

The decoder reconstructs the entire time series using an autoregressive approach with GRU modules, similar to the encoder with N-stacked GRU modules with a hidden d_{AE}. The latent representation, r, includes the final hidden state of the previous GRU or a noise-generated state. The hidden state is then used as the initial $\mathbf{h}_T^n$ for the first GRU module in the decoder. The reconstruction process aims to model $p(\mathbf{x}_i|\mathbf{x}_{1..i-1})$ for each time step i and follows these steps:

$$\hat{\mathbf{e}}_i = [\mathbf{x}_{i-1}, \mathbf{s}], \quad \hat{\mathbf{h}}_i^n = \text{GRU}(\hat{\mathbf{e}}_i, \hat{\mathbf{h}}_{i-1}^n), \quad n \in [1, N]$$
$$\hat{\mathbf{x}}_i = \text{Sigmoid}(\mathbf{W}_x \hat{\mathbf{h}}_i^N + \mathbf{b}_x),$$

Here, $\hat{\mathbf{e}}_i$ represents the autoregressive latent representation input to the GRU, initially set to $\hat{\mathbf{e}}_1 = [0, \mathbf{s}]$. $\hat{\mathbf{h}}_i^n$ denotes the hidden state at time step i of the GRU, and n ranges from 1 to N. $\hat{\mathbf{x}}_i$ is the reconstructed value for time step i. The weights and biases in a dense layer are represented by $\mathbf{W}_x$ and $\mathbf{b}_x$, respectively.

The autoencoder is trained using the mean squared error (MSE) reconstruction loss, defined as $L_{re} = \text{MSE}(\hat{\mathbf{x}}, \mathbf{x})$, and this occurs before the GAN training to ensure that the training data is accurately mapped to meaningful latent representations.

Since the latent representation has a low dimension and the generator and discriminator operate on it rather than the high-dimensional time series, the generator can easily synthesize new latent representations. The low-dimensional representation simplifies the architecture of the generator and discriminator significantly. The generator consists of five fully connected layers with layer normalization, while the discriminator has three fully connected layers. Both networks use LeakyReLU as the activation function.

To ensure stable training, Pei *et al.* [63] incorporated the Wasserstein distance with the Gradient Penalty technique proposed by Gulrajani *et al.* [64] to enforce 1-Lipschitz continuity in the discriminator. By employing the Wasserstein distance, their optimization objective was formulated as follows:

$$\min_G \max_D \mathbb{E}_{r \sim \text{encoder}(X)}[D(\mathbf{r})] - \mathbb{E}_{z \sim p(z)}[D(G(\mathbf{z}))], \tag{4.11}$$

where G represents the generator, D represents the discriminator, $\mathbf{r}$ denotes real samples from the encoder, and $\mathbf{z}$ denotes random noise samples. The maximum and minimum are with respect to the whole expression.

Pei *et al.* [63] compared their RTSGAN model with other models, namely C-RNN-GAN, RGAN, and TimeGAN. They used two datasets used in TimeGAN[60]

for their tests. Evaluation of the models involved similar methods, including the discriminative score, prediction score, as well as PCA and t-SNE for visualization. The findings revealed that the RTSGAN model outperformed the others regarding discriminative and prediction scores while demonstrating a greater diversity in generated samples. Although the results section of their study provided a concise model comparison, the evaluation might be influenced by potential discrepancies in the training efforts and implementation among the frameworks. Additionally, the absence of visual plots comparing the generated samples to the target data limits the comprehensive assessment of their results.

4.1.6 Other works

So far, only architectures designed for generating multivariate data have been introduced, an area that has received less attention than the generation of univariate data. However, numerous noteworthy frameworks on univariate data can provide valuable insights, for instance, their methods and experiments.

For instance, Hartmann *et al.* [20] investigated the use of a convolution-based Generative Adversarial Network (GAN) for generating new univariate brain signals (EEG data). Instead of employing recurrent modules, they utilized a recurrent-free architecture and adopted the Wasserstein distance to stabilize the training process. Their goal was to model long-range EEG data with over 1000 time steps, and they achieved this by progressively scaling the generator's capabilities, drawing inspiration from GANs used for generating high-resolution images.

Zhu *et al.* [65] aimed to determine the most advantageous combination of generator and discriminator architectures for generating univariate EEG signals. They explored various combinations and found that the best results were obtained using a convolution-based discriminator with a bi-directional LSTM generator. The generator architectures they experimented with included CNN, LSTM, GRU, and MLP discriminators.

Brophy *et al.* [66] investigated the generation of multivariate EEG data. They addressed the vanishing gradient issue by employing a different objective function called Least Squares GAN and introduced a regularizer that penalized the generation of samples that significantly differed from the target samples. To measure the similarity between time series, they used a technique called Dynamical Time Warping (DTW). The generator in their architecture consisted of stacked LSTMs, while the discriminator utilized 2-dimensional convolution. Although they did not compare their results with other models, their approach of incorporating 2-dimensional convolution in generating multivariate data is noteworthy.

4.2 Constrained and physics-informed GANs

GANs have become popular in generating realistic samples from a given data distribution. Integrating physical laws into GANs is a growing research topic. While many studies have presented physics-informed GANs [67–74], there is a lack of

comprehensive comparisons between physics-informed GANs and standard GANs. Such comparisons are essential for understanding the benefits of incorporating physics constraints and the limitations of standard GANs in capturing the underlying physics. Defining valuable constraints for the underlying dynamics in the data may be difficult or impossible in real-world scenarios, making standard GANs the only available option. Hence, understanding the boundary between what GANs can and cannot learn is crucial for selecting the appropriate model for any particular application. A deeper understanding of the GAN framework would help choose the correct model for the appropriate data and application, preventing the adoption of flawed models that do not achieve desired outcomes.

The work of Stinis *et al.* [24] was among the first to investigate the difference between physics-informed GANs and standard GANs. Stinis *et al.* [24] explored enforcing constraints for extrapolation in time series data, meaning using the GAN to predict the next step in the sequence, single-step prediction. The models only operated using the current input and its rate of change (gradient), similar to the forward Euler method. They performed tests on a system of ODEs exhibiting chaotic behavior and found that enforcing constraints significantly improved the results. Although their results were promising, they emphasized the need for future research to explore more intricate dynamical systems, including Hamiltonian systems and higher-dimensionality problems. They also hinted at the potential application of GANs in solving Partial Differential Equation (PDE) with random coefficients.

In a succeeding study, Zeng *et al.* [23] extended the research conducted by Stinis *et al.* [24] by examining the impact of incorporating imprecise constraints on image data, specifically focusing on geometric constraints in a dataset of circles and divergence-free flow velocity fields. The authors observed that while the standard GAN generated thick rings that resembled circles to some extent, the informed GAN produced significantly more accurate circles with minimal spacing between the points. Similar improvements were observed in the case of flow velocity fields, where the informed GAN outperformed the standard GAN. Notably, both models generated samples that appeared realistic and approximated the target data distribution well, highlighting that it is the finer details that the standard GAN struggles to capture accurately. The study's findings indicate that incorporating constraints, even imprecise, enhances convergence speed and elevates the quality of generated samples.

However, there is still much to be investigated, as these studies focused on specific examples and data types. Stinis *et al.* [24] exclusively explored the effects on one-dimensional time series data and did not consider the incorporation of past time steps in the generation process. Incorporating a broader temporal context provides more information and enhances the prediction of future values in the sequence. Extending the analysis to more complex time series data with multiple variables and longer sequences remains thus as an open area for investigation. Additionally, Zeng *et al.* [23] did not explore the implications of imprecise constraints on time series data, leaving room for further exploration in this domain.

Consequently, investigations need to discern the limitations and challenges that GANs encounter when addressing intricate problems. Such inquiries will contribute to a more comprehensive understanding of the capabilities and potential biases of GANs, helping to select more robust models for various applications.

4.3 Evaluation methods

The GAN framework often faces challenges in finding suitable evaluation methods to assess the quality of generated samples. This challenge becomes even more complex when applying GANs to time series generation. Evaluation metrics commonly used for GANs have primarily relied on visual inspection and metrics specifically designed for images, which are not directly applicable to time series data. The lack of proper evaluation metrics has resulted in a diverse range of metrics being employed in published works on GANs for time series. This variation in metrics arises from the introduction of inadequate metrics and metrics tailored to specific datasets, which has hindered the establishment of a clear consensus on which metrics to use. The task of defining a satisfactory metric for comparing two sets of samples without labels is inherently challenging. It is thus a significant factor contributing to the wide array of different metrics being used.

This section aims to provide an overview of different evaluation methods used in the literature and help in finding a suitable evaluation method to be utilized for our experiments. A preferred evaluation method should be interpretable and reproducible, avoiding excessive assumptions and biases. It should not rely on other models and should produce consistent results. Furthermore, since the field of GANs in time series is divided into univariate and multivariate data, the evaluation method should be applicable to multivariate data.

Table 4.1 comprises a list of the different evaluation methods used in the literature of GAN for time series generation. The evaluation methods are marked with an "x" to denote which ones which do not satisfy the criteria of a desirable method. It is clear that there certainly exists an evaluation method problem, as many of them being based are based on the results from other natural networks. This does not mean they are not useless, but rather not ideal according to the desired criteria. There exist, however, three methods that do satisfy the desired criteria. These include the Maximum Mean Discrepancy (MMD) [75], the Wasserstein distance, and the Dynamic Time Warping (DTW) [76].

However, many of the listed evaluation metrics in Table 4.1 rely on other neural networks, introducing additional reproducibility issues. The training and fine-tuning of these models can be sensitive to factors like hyperparameters, initialization, and training data. Reproducing the exact conditions and achieving identical performance becomes challenging, especially when working with different datasets or experimental setups. This lack of reproducibility hinders the comparability and generalizability of the evaluation results, making it difficult to establish consistent benchmarks and accurately assess the model's performance. Furthermore, the lack of interpretability in these model-based evaluation methods

adds to the challenge of understanding and interpreting the results.

Despite these drawbacks, model-based evaluation methods can still serve a purpose during training and as supplementary evaluation methods for identifying correlations between different metrics. Using a range of diverse evaluation methods that capture different aspects can be more powerful than relying on a single metric.

Among the evaluation metrics that met the desired criteria, only the Wasserstein distance was chosen for further investigation and deemed suitable as an evaluation metric. Dynamic Time Warping (DTW) was excluded since its original purpose is to measure differences between two time series rather than two collections of time series. The possibility of extending DTW to compare collections and its applicability to multivariate time series were not explored. Maximum Mean Discrepancy (MMD) was also not selected because it only compares the means of two sample sets after applying a "kernel" function to transform them into another space.

Further examination of the Wasserstein distance revealed its intractability when dealing with high-dimensional data. However, estimation methods have been developed to address this challenge, including the Sinkhorn divergence [34] (section 2.3.2) and the Sliced-Wasserstein distance [35] (section 2.3.3).

The Sinkhorn divergence was not considered highly useful as an evaluation method due to introducing additional hyperparameters, namely the entropy regularization parameter ϵ and the number of iterations in the algorithm. Similarly, due to its non-deterministic algorithm, the Sliced-Wasserstein distance was not chosen because it produces different results for identical runs. However, a more in-depth investigation into optimal transport theory and the search for faster, more accurate, and deterministic algorithms led to the discovery of a deterministic version of the Sliced-Wasserstein distance known as the Fast approximation of the Sliced-Wasserstein distance, as described in the section 2.3.4.

The Fast approximation of the Sliced-Wasserstein distance (SW) provides the advantage of fast computation, allowing for real-time measurement during GAN training to monitor performance and detect early mistakes or suboptimal hyperparameters. However, there is a lack of discussion regarding the accuracy of the estimate when handling high-dimensional data with numerous variables and the number of samples required for reliable estimates. Previous research by Nadjahi [26] has demonstrated that the accuracy of the estimate improves with an increasing number of features (a 1x28x28 image sample has 784 features). The performance of the estimate in scenarios involving a high correlation between features and its applicability to time series data remains unexplored.

4.3.1 Visualization techniques

The evaluation of GAN models presents a challenge in terms of selecting appropriate metrics to assess performance. This has led many researchers to rely on visualization techniques to gain insights into the quality and characteristics of

generated samples [60, 63, 78, 79, 84–86]. While visualization methods can be subjective and challenging to interpret consistently across different models and datasets, they remain very useful for capturing large errors in generated samples during tuning and general performance. Traditional techniques like PCA and t-SNE have long been popular for visualizing high-dimensional data, but a more recent addition, UMAP, has gained significant traction since its introduction in 2018. These visualization methods serve as valuable tools in overcoming the complexities of high-dimensional data by reducing dimensionality and revealing patterns and relationships between variables [87].

PCA

Principal Component Analysis (PCA) is a data compression and visualization technique introduced by Pearson [88]. Its primary purpose is to reduce the dimensions of a dataset while preserving the most critical information. This is accomplished by determining the linear combination of the original variables that account for most of the data's variance. Sorting these variables based on the amount of variance they explain in the data allows for identifying the most important features or patterns. Additionally, the sorted variables can be used to visualize the data in a lower-dimensional space. For example, picking two of the most significant variables and visualizing them with a scatter plot reveals information about the higher inaccessible dimensional structure of the data. As a result, PCA is a valuable tool for visualizing high-dimensional datasets and identifying inter-variable relationships.

t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE), by Maaten and Hinton [89], is a nonlinear dimensionality reduction technique primarily used to visualize high-dimensional data in a two- or three-dimensional space. It works very differently from PCA as it constructs a nonlinear mapping from the high to the low-dimensional space with an intent to preserve the local structure. It defines a low-dimensional probability distribution aimed at estimating the high-dimensional distribution and making them similar by minimizing the Kullbeck-Leibler divergence (equation 2.3) using gradient descent. While the t-SNE algorithm has gained popularity for its effectiveness in data visualization, it has specific limitations, such as its non-deterministic nature and its inability to preserve the global structure of the data.

UMAP

Uniform Manifold Approximation and Projection (UMAP) by McInnes *et al.* [87] is also a dimensionality reduction method, visually similar to t-SNE. It constructs visualizations of high-dimensional data in a lower-dimensional space but uses ideas from topological data analysis and manifold learning to identify similarities

in the data. The exact workings of UMAP are out of the scope of this thesis. Despite being non-deterministic like t-SNE, UMAP is more effective at preserving the global structure of the data, meaning that distant points in the low-dimensional visualization also are distant and unique in the dataset.

4.3.2 Prediction Score

The Prediction Score (PS), proposed by Yoon *et al.* [60], serves as a metric to assess the quality of generated samples by examining their temporal dynamics. It involves training an autoregressive model on the generated data and evaluating its performance on separate hold-out datasets consisting of real and generated data. By comparing the errors between these datasets using metrics like mean absolute error, insights can be obtained regarding their similarity. A low relative error suggests a resemblance between the generated and real data, while a high error indicates differences [60].

However, it is important to note that the Prediction Score does not capture certain aspects such as mode dropping and overfitting [77]. Overfitting occurs when the generated samples do not exhibit a uniform distribution of different modes, with some modes being more likely to occur than others. Furthermore, the metric heavily relies on the outcomes of a neural network, which may introduce biases and assumptions that can impact the evaluation. For instance, the accuracy of the metric assumes the neural network has the appropriate architecture to capture the data probability distribution accurately [77].

Table 4.1: Overview of evaluation metrics used in GAN time series literature. Metrics that rely on other neural networks, only support univariate data, or are non-deterministic are marked with an "x". The purpose of this table is to identify evaluation methods that are not desirable, and even though multiple categories may apply, only one "x" is considered. Metrics that meet the desired conditions are displayed in bold font and are not marked.

Evaluation metric	**Reference**	**Model-based**	**Only univariate**	**Non-deterministic**
1D Fréchet Inception Score (FID)	[20, 65]	x		
Inception Score (IS)	[20]	x		
Maximum Mean Discrepancy (MMD)	[57, 59, 66, 77]			
Discriminative Score (DS)	[60, 63, 77, 78]	x		
Predictive Score (PS)	[60, 63, 77, 78]	x		
Train on Synthetic Test on Real (TSTR / TRTS)	[57, 60, 63, 77–81]	x		
Sliced Wasserstein distance (SWD)	[20]			x
Wasserstein / Earthmover distance (EMD)	[82, 83]			
Autocorrelation Function (ACF)	[82]		x	
Kruskal-Wallis H Test and Mann-Whitney U test	[79]		x	
K-means Clustering using DTW	[59]	x		
ARIMA	[59]		x	
Kullback-Leibler Divergence (KL)	[83]	x		
Jensen-Shanon Divergence (JSD)	[78, 83]	x		
Kolmogorov-Smirnov test statistics	[83]		x	
Dynamic Time Warping (DTW)	[58, 66]			

Chapter 5

Method

This chapter contains high-level information about the essence of our methodology and the reasoning behind the experiments. Details for reconstructing experiments and the evaluation methods used are provided in sections 6 and 7.

The first experiment, named **Framework testing**, involved exploring various frameworks to assess their capabilities in generating time series. This examination allowed the selection of the most compelling framework, subsequently employed in the main experiment. The primary experiment, named **Learning of underlying physical laws**, delved deeper into the chosen framework's learning capabilities to address our primary research questions.

Given the demanding nature of GANs, involving extensive testing and parameter tuning, a separate section (section 5.3) is provided to outline the comprehensive tuning procedure employed for all experiments.

Python 3.9 served as the primary programming language for our experiments, while the PyTorch 1.13 framework was employed for model creation and training. The selection of PyTorch was motivated by its automated differentiation capabilities, which greatly facilitated the design of novel loss objectives. Furthermore, we leveraged Neptune for logging, visualization, and monitoring of model performance and hyperparameters during the training process. Code to reproduce experiments is publically available at GitHub: `https://github.com/Kohmann/master-GAN`.

5.1 Experiment: Framework testing

This experiment aims to explore various proposed frameworks for generating multivariate time series. It involves evaluating these frameworks using various evaluation methods on a simple dataset. The results will guide the selection of which model to utilize in the main experiment (Learning of underlying physical laws). Additionally, this experiment aims to validate the relevance and applicability of the chosen evaluation methods to complex data. Thus, this experiment will evaluate both the respective frameworks' performance and the evaluation methods' utility.

To conduct the experiment, a set of selected frameworks will be implemented and trained on a synthetic dataset. The models will then be evaluated on the same test data and a range of evaluation methods. The success of the experiment relies on addressing several questions, including the selection of frameworks, the properties of the dataset, and which appropriate evaluation methods to utilize.

5.1.1 Defining dataset properties

A simple synthetic multivariate time series dataset was considered for investigating the respective frameworks' generating abilities. It was designed to focus on several essential time series properties such as sequence length, long-term dependent variables, and variable correlation. All of these are considered important properties and are also present in the dataset employed in the main experiment. Accurately modeling time series data across multiple time steps is essential and challenging. The complexity increases as the time series length grows, posing difficulties for models to capture the underlying patterns and dynamics effectively. A framework able to model long time series is preferable as it allows extra flexibility for the dataset utilized in the main experiment, assuming it can also adeptly model shorter sequences with comparable performance.

Another critical aspect of time series data revolves around the correlation observed between variables across temporal and spatial domains. While obtaining a model that generates realistic-looking data is important, it is equally important that the dynamics of the time series are captured accurately. Consequently, the dataset should contain spatial and temporal correlations, including long-term dependencies in single features and combination with other features.

5.1.2 Framework selection

The framework selection process involved thoroughly examining the existing literature to identify techniques that satisfied our specific criteria for generating multivariate time series data. These criteria encompassed a range of architectural complexities and objective functions, ensuring a diverse selection of frameworks.

The choice of objective function holds significant importance as it serves as the foundation for the optimization process, directly influencing the attainment of desired results. Given the known challenges of training GANs, many modified objective functions have been proposed to address these issues. However, a consensus on the optimal objective function for a particular case is not yet established. Therefore, including a diverse array of objective functions in the framework selection process is considered important in order to identify high-performing options and gain insights into their respective strengths and weaknesses.

The architectural complexity associated with each framework is equally significant to the objective function. The architecture of a model establishes the constraints and assumptions of the task at hand, ultimately dictating its performance capabilities. While simpler architectures are easier to interpret, they may struggle

to capture complex relationships within the data. On the other hand, overly intricate models can introduce unnecessary complexity. To account for variations in model complexity, a balanced approach is adopted, including both relatively simple architectural models and those of greater complexity in the collection of frameworks.

An additional criterion influencing framework selection is the quality associated with the proposed techniques. Only published and cited works with source code are considered, indicating the proposed techniques' reliability and usefulness. Conducting a quality check ensures confidence in the effectiveness of the frameworks and minimizes the likelihood of encountering unexpected implementation issues.

Selected frameworks

The collection of frameworks that met our criteria consisted of four models, namely RGAN [57], TimeGAN [60], RTSGAN [63], and COTGAN [61]. The selection was limited due to time limitations, as each framework required considerable architecture and hyperparameter tuning to produce desirable results.

RGAN was selected due to its simplicity and similarity to the regular GAN framework. It utilizes the classic GAN objective function and operates using a straightforward architecture. It was chosen as a baseline for one of the simplest time-series GANs to implement and aid in evaluating the improvements that other frameworks may introduce.

TimeGAN was selected due to its complex architectural design, compared to RGAN, specifically aimed at allowing the model to capture the complex dependencies in the data. Most of the changes are in the architecture, which utilizes the same baseline loss function as RGAN. However, it incorporates some additional terms in the loss to learn the time-dependent features better and improve stability. Moreover, it is a well-cited[1] framework that has been considered state-of-the-art for some time.

RTSGAN shares architectural similarities with TimeGAN but employs a different implementation that enables the use of a more stable loss function. It utilized the Wasserstein distance as the objective function and was selected for its balance between a relatively complex architecture and a sensible objective function.

Lastly, COTGAN was selected for its high focus on an objective function tailored specifically for time-series data. The objective function is based on optimal transport, the Sinkhorn divergence, but modified to account for causal data. This allows the use of a much simpler architecture, close to RGAN's, which is more interpretable. Compared to TimeGAN, it focuses on a more appropriate objective function instead of a complicated architecture.

[1] 488 citations according to Google Scholar

5.2 Experiment: Learning of underlying physical laws

The primary objective of this experiment is to determine whether GANs follow the underlying constraint present in the data. A synthetic dataset that adheres to measurable physical laws is utilized to investigate this, along with the best-performing framework from the "Framework testing" experiment (as described in section 5.1).

The experiment is divided into four parts, each assessing the model's performance under varying circumstances and with additional guidance. The first two experiments evaluate how well the model performs when presented with two datasets of different complexity. The remaining two experiments will serve as supporting experiments to determine the plausible performance of the model and establish an upper limit to compare the GAN's performance against.

5.2.1 Defining dataset properties

The key to this experiment lies in the choice of data on which to base the experiment on. It must simulate a physical system obeying well-defined laws, and it must be possible to validate or measure to what extent the data obeys these laws.

A mathematical model which satisfies these criteria is the simulation of ocean waves following the Korteweg-de Vries (KdV) equation [90]. This model simulates the interaction of multiple waves in a closed physical system and ensures that the simulations follow the energy conservation law, which means that the energy at each point in time remains unchanged. Energy conservation can be validated efficiently as explicit formulas exist for this mathematical model. These formulas consider mass, momentum, and energy quantities, all conserved in the physical system. Using these formulas, the energy associated with a simulation can be calculated directly from the generated simulations. This means the measurement of conserved energy is independent of another ground truth sample, allowing us to avoid steps that could introduce errors.

Another important characteristic of simulating ocean waves using the KdV equation is that it involves solving a non-linear problem. Such problems are non-trivial and can not always be solved analytically but require numerical approximations. For the KdV equation, this is true for most initial waves, but it also has a perfect analytical solution in some special cases. An advantageous feature of the KdV equation is its compatibility with certain numerical methods when solving it through numerical approximations. These methods exhibit a relatively low relative error in comparison to other mathematical simulations.

Two datasets based on this physics-based simulation will be used as the basis for all experiments. The first dataset involves a single wave's evolution over time, and the second dataset is the more complex scenario of two colliding waves. These datasets will serve as the foundation for all experiments.

5.2.2 Experiments

This section outlines the experiments designed to assess whether the GAN can capture the physical laws underlying the data. The experiments utilize two datasets of varying complexities. The first experiment considers the simple case of a single wave propagating through time. The second experiment involves the collision of two waves, considered the primary test to determine whether the GAN model is able to adhere to the underlying physical laws.

Two supporting experiments are conducted to provide solid facts regarding the GAN architecture and establish a performance bound to evaluate the GAN against. The first supplementary experiment aims to demonstrate the potential of the generator network by utilizing an autoencoder. The autoencoder consists of an encoder network and the GAN generator as the decoder. The experiment evaluates the generator's ability to reconstruct samples from their latent representations by training the autoencoder to encode and decode target samples. This test assumes that the encoder can provide meaningful representations, and by using the same latent space and generator architecture as the GAN, it provides evidence of a plausible latent space and generator weights capable of generating realistic samples that conform to the physical laws.

In the second supplementary experiment, the GAN used to generate colliding waves is informed about the errors associated with the underlying physical laws. Informing the model is accomplished by incorporating a conservation regularization term into the objective function. By explicitly considering conservation errors, the GAN is expected to perform better and generate data that more closely adheres to the conservation laws. This experiment aims to establish an upper limit on performance, providing a benchmark against which the uninformed GAN's performance can be evaluated and compared.

5.3 Tuning procedure

Training the models consisted of modifying the architecture and tuning the hyperparameters accordingly in a circular process. Because GANs are known for their stability issues, finding a functional architecture with good hyperparameters requires a lot of trial and error [91, 92]. Training runs often lead to non-convergence or mode collapse with little intuition as to why, and uncertainty of how the architecture and the hyperparameters behave together adds to the problem. The number of configurations to test becomes very large and difficult to navigate. To attempt to address these issues, a circular training approach is followed to organize the training process systematically and avoid redundant runs.

The circular approach consisted of 4 high-level steps: 1) choose an initial architecture, 2) exhaustive hyperparameter search, 3) modify architecture, 4) jump to step 2. As each step consists of many sub-steps and techniques, we will go into further detail on steps 1, 2, and 3 below.

5.3.1 Initial architecture

The architecture proposed in the relevant paper is used as the initial architecture for each type of model. These have been proven to work for similar datasets and serve as a good starting point. Not all architectures proposed in the relevant papers directly apply to our datasets, so some minor modifications are applied to make them compatible.

5.3.2 Hyperparameter search

To assess the stability of the architecture, it is first tested on the chosen dataset with some initial set of hyperparameters. These are initially selected using the hyperparameters listed in the relevant paper or based on empirical knowledge from previous experiments. Some architectures introduce new hyperparameters specific to the architecture modules or objective functions used. In these cases, the reported values are used.

Not all hyperparameters may drastically change and affect the model, but experience indicates that the learning rate and batch size stand out as the most important. They determine if the model will be subject to mode collapse and non-convergence and follow a complex relationship that is important to settle before fine-tuning with the other hyperparameters. Because of their effect on the model, they are prioritized above all the other hyperparameters and assumed to provide enough information to determine whether the architecture is sufficient. When a model shows signs of some proper learning, the other hyperparameters are also considered.

Because GANs are highly sensitive to different combinations of hyperparameters, a hierarchical structure is used to organize the hyperparameters based on their impact on sample quality. The hierarchy is used to structure which hyperparameter to first fine-tune after making new adjustments and help to determine the modification's gain faster. Initially, this hierarchy is determined through guesswork, but it is refined as further information is gathered during training to produce a more detailed list. The learning rate and batch size are typically ranked as the top two most significant hyperparameters across all architectures.

When tuning the hyperparameters, it happens in a controlled manner where the focus is on one hyperparameter at a time and freezing the others. This makes understanding how specific hyperparameters affect the training process and result easier. Using more sophisticated hyperparameter optimization tools such as grid search, evolutionary algorithms, or Bayesian optimization is not feasible because these algorithms require many more training runs and good evaluation metrics to guide the search.

If the model does not converge, the experiment is repeated with more epochs, usually double the previous number. This is especially important for standard BCE loss architectures, which may experience sudden random learning spikes after extended periods of seemingly no learning. In the case of architectures using optimal transport, this is more stable, and more epochs generally only increase the sample

quality.

Some hyperparameters were split into predefined ranges to quickly test a range of values and get a notion of the architecture stability. The most notable hyperparameter ranges were the learning rates $\{0.0001, 0.0005, 0.001, 0.01\}$ and batch sizes $\{8, 16, 32, 64\}$. The number of epochs varied depending on the learning rate and batch size and was adjusted accordingly. The minimum number of epochs employed was 100 and was usually increased for lower learning rates.

Several evaluation metrics and visualization methods are used to evaluate the performance and effects of the hyperparameters. More specifically, the Sliced-Wasserstein distance (section 2.3.4), discriminator and generator loss, the sample distribution using PCA, t-SNE, and UMAP (section 4.3.1), and traditional sample assessment to spot irregularities.

5.3.3 Modifying the architecture

If the architecture performs poorly on the evaluation metrics after several trials of different hyperparameters, it is a sign that it is insufficient and unable to learn properly. In these cases, the architecture is modified. We first try small modifications and, if unsuccessful, move on to larger changes.

The small changes generally include increasing the size of hidden layers, the number of layers in fully-connected modules, and introducing batch normalization between layers. These are intended to improve the network's ability to combine representations to produce the desired output.

The more extensive changes aim to improve the network's ability to create better representations and learn the data dynamics. These modifications include increasing the number of RNN layers, using another type of RNN (e.g., LSTM or GRU), replacing RNN with convolution, and rearranging specific modules.

Chapter 6

Experiment: Framework testing

This chapter provides comprehensive coverage of the framework testing experiment, including an overview of the dataset and frameworks used, an analysis of the results, and identifying the most suitable framework for the main experiment. It is organized into three sections: Experimental Setup, Results and Discussion, and a concluding section summarizing the findings.

The presented content builds upon the preliminary project [37] but has been extensively revised to include additional information and improve overall clarity.

6.1 Experimental Setup

This section presents essential information required to reproduce the experiments. It contains specific details about the dataset, framework architectures, and the hyperparameters used for training.

6.1.1 Dataset

The considered dataset was created using explicit functions to guarantee precise knowledge of the underlying dependencies and comprised of 1500 samples. The data consisted of three multivariate series following a path defined by sinusoidal waves with different frequencies and phases. Each sample contained three distinct variables that evolved over time following three different paths, defined by three functions, namely $\sin_1(t)$, $\sin_2(t)$, and $\sin_3(t)$. In this context, t represents a time step in the sequence, bounded by the total number of time steps to consider, T, which we set to 100. The sinusoidal functions, $\sin_1(t)$, and $\sin_2(t)$, were generated independently using the wave function

$$\sin_i(t) = \sin(\omega_i \frac{2\pi t}{T} + \varphi_i), \quad t = (0, 1, \dots T-1), \tag{6.1}$$

where $i = 1, 2$ determines which function to use and controls the set of frequencies ω_i and phases φ_i to consider. The parameters for $\sin_1$ were uniformly sampled from $\omega_1 \in (1, 3]$ and $\varphi_1 \in (-\frac{\pi}{2}, 0]$, while for $\sin_2$ they were sampled from $\omega_2 \in$

$(4,6]$ and $\varphi_2 \in (0, \frac{\pi}{2}]$. The last sinusoidal wave ($\sin_3$) was created by taking a linear combination of the two waves and applying an exponential moving average across previous time steps. This is calculated as:

$$\begin{aligned} \sin_3(t) &= \frac{1}{2}\sum_{j=0}^{t} \alpha^{t-j}\left(\sin_1(j) + \sin_2(j)\right)\right) \\ &= \frac{1}{2}\sum_{j=0}^{t} \alpha^{t-j}\left(\sin(\omega_1 \frac{2\pi j}{T} + \varphi_1) + \sin(\omega_2 \frac{2\pi j}{T} + \varphi_2)\right), \end{aligned} \tag{6.2}$$

where α represents the weighting importance of the current time step, and the subscripts $1, 2$ denote each of the two independent waves. The exponential used in the moving average was set to $\alpha = 0.7$. After all of the samples were created, the data were scaled to fit inside the bounds $[0, 1]$. The complete dataset was a matrix of dimensions $[N, T, C] = [1500, 100, 3]$, where N represents the number of samples in the dataset, T is the sequence length, and C is the number of variables.

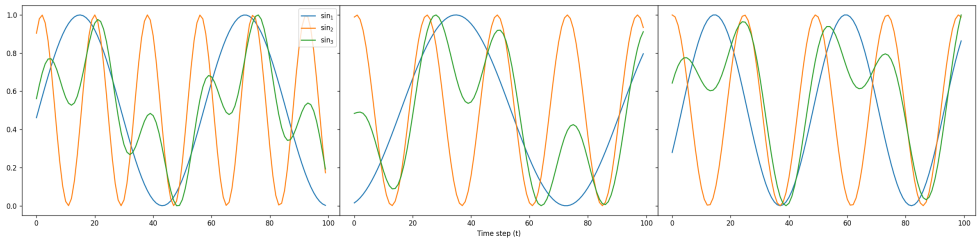


Figure 6.1: Each panel shows the three time-dependent variables $\sin_1$, $\sin_2$, and $\sin_3$ of three unique samples.

6.1.2 Framework details

All frameworks, namely RGAN, TimeGAN, RTSGAN, and COTGAN, was implemented in PyTorch with the aid of the publicly available code provided in the respective research papers. Initially, RGAN, TimeGAN, and COTGAN were developed using TensorFlow, but they were translated to PyTorch to maintain consistency.

The model architecture and hyperparameters of each framework were then fine-tuned to the best of our abilities using the tuning procedure described in section 5.3 until acceptable results were obtained or numerous relevant modifications were exhaustively tested.

RGAN

The first architecture, RGAN, was implemented using a very similar version proposed by Esteban *et al.* [57], using two stacked LSTMs for both the generator and

discriminator and the hidden dimension set to 20. It was trained for 1000 epochs with a learning rate of 0.0001 using the Adam optimizer, batch size of 32, and sampling noise from a standard normal random variable $\mathbf{Z} : \mathbb{R}^{T \times D}$. Here T denotes the number of time steps in the dataset, which equals 100 for the sinusoidal dataset, and D represents the number of noise dimensions associated with each time step, which we set to 100. For clarity, sampling a batch of B samples $\mathbf{z} \sim p_{\mathbf{Z}}$ takes the form of a matrix $[B, T, C]$. A detailed overview of the GAN network architecture is outlined in Table 6.1.

Discriminator

Layer	Input	Output	Activation
LSTM	B, T, C	B, T, 20	
LSTM	B, T, 20	B, T, 20	
Linear	B, T, 20	B, T, 1	LeakyReLU
Flatten	B, T, 1	B, T	Sigmoid

Generator

Layer	Input	Output	Activation
LSTM	B, T, 100	B, T, 20	
LSTM	B, T, 20	B, T, 20	
Linear	B, T, 20	B, T, C	Sigmoid

Table 6.1: RGAN discriminator and generator network architectures for the sinusoidal dataset. Static dimensions are denoted by a capital letter.

TimeGAN

TimeGAN was implemented using the same architectural design as Yoon *et al.* [60]. The five networks, namely the embedding, recovery, supervised, generator, and discriminator, all employ a 2-layered GRU followed by a linear layer. Similarly to RGAN, TimeGAN samples from the same noise variable $\mathbf{Z} : \mathbb{R}^{T \times D}$ and uses the same number of nodes in the hidden dimension. Table 6.2 provides an overview of the architecture for all networks except the supervised network, which was designed to be identical to the embedding architecture except for the input of the first layer, which was set to $[B, T, 20]$ instead of $[B, T, 3]$. The hyperparameters for all networks were set to a batch size of 20 and a learning rate of 0.0005 using the Adam optimizer. During training, the embedding and recovery networks were first trained as an autoencoder to learn useful representations of the data and recover the data from those representations. Once the autoencoder was able to encode and decode the data, the supervisor network was added to the training process and trained using teacher forcing to learn the conditional distribution of the representations. Finally, when both the autoencoder and the supervisor network achieved satisfactory results, the discriminator and generator were added to the training loop. All networks were then trained jointly until the generator reached the desired results. The number of epochs for each stage of the three

training phases was set to 1000, 1000, and 2000, respectively.

Embedding

Layer	Input	Output
GRU	B, T, 3	B, T, 20
GRU	B, T, 20	B, T, 20
Linear	B, T, 20	B, T, 20
Sigmoid	B, T, 20	B, T, 20

Recovery

Layer	Input	Output
GRU	B, T, 20	B, T, 20
GRU	B, T, 20	B, T, 20
Linear	B, T, 20	B, T, C
Sigmoid	B, T, C	B, T, C

Discriminator

Layer	Input	Output
GRU	B, T, 20	B, T, 20
GRU	B, T, 20	B, T, 20
Linear	B, T, 20	B, T, 1
Flatten	B, T, 1	B, T
Sigmoid	B, T	B, T

Generator

Layer	Input	Output
GRU	B, T, 100	B, T, 20
GRU	B, T, 20	B, T, 20
Linear	B, T, 20	B, T, 20
Sigmoid	B, T, 20	B, T, 20

Table 6.2: TimeGAN network architectures for the sinusoidal dataset. Static dimensions are denoted by a capital letter.

RTSGAN

The original implementation of RTSGAN [63] included support for varying sequence lengths. However, this feature was removed in our implementation to enhance readability without compromising the model's capacity. The encoder and decoder networks consist of a 2-layer GRU, with the hidden dimension for the autoencoder set to 40, resulting in a latent representation dimension of 120. The generator and discriminator networks are composed of fully-connected layers and specified in detail in Table 6.3. A standard normal noise vector of 120 dimensions, $\mathbf{Z} : \mathbb{R}^{120}$, was used since the latent representations are not restricted by the dataset's time series length. Firstly, the autoencoder was trained for 300 epochs using the Adam optimizer with a learning rate of 0.001 to learn meaningful representations for the data. Following this, the generator and discriminator were trained for 500 epochs with the RMSProp optimizer using a learning rate of 0.0001 while using only the encoder to encode the real data into the representation space. The decoder network was only used for visualizing results once the generator and discriminator were trained. Since RTSGAN utilizes the Wasserstein distance as the objective function, the discriminator and generator were updated at a ratio of 10 : 1. This means that the discriminator was updated 10 times before the generator was updated once. All models were trained with a batch size of 60.

Discriminator

Layer	Input	Output	Activation
Linear	B, 120	B, 80	LeakyReLU
Linear	B, 80	B, 40	LeakyReLU
Linear	B, 40	B, 1	

Generator

Layer	Input	Output	Activation
Linear	B, 120	B, 120	LayerNorm, LeakyReLU
Linear	B, 120	B, 120	LayerNorm, LeakyReLU
Linear	B, 120	B, 120	LayerNorm, LeakyReLU
Linear	B, 120	B, 120	LeakyReLU

Encoder

Layer	Input	Output	Activation
GRU	B, T, C	B, T, 40	
Concat	B, 20, 20, 20	B, 120	
Linear	B, 120	B, 120	LeakyReLU
Linear	B, 120	B, 120	LeakyReLU

Decoder

Layer	Input	Output	Activation
Reshape	B, 120	B, T, 20	
GRU	B, T, 20	B, T, 40	
Linear	B, T, 40	B, T, C	Sigmoid

Table 6.3: RTSGAN generator and discriminator network architectures for the sinusoidal dataset. It is worth noting that the discriminator and generator are not bound to the sequence length. Static dimensions are denoted by a capital letter.

COTGAN

The best-performing architecture and hyperparameters considering the COTGAN framework [61] consisted of a generator using two stacked GRU modules and a 3-layered fully connected output layer with 64 nodes in each layer. The discriminator utilizes one-dimensional convolutions with stride 1, padding set to "same", and kernel size 5, in combination with the stack of two GRU modules. Its output dimension J is set to 32. Details regarding the number of connections in each layer and the architecture structure are shown in table 6.4. The model was trained for 1200 epochs with the Adam optimizer using a learning rate of 0.0005, and a batch size of 72 was used. A learning rate scheduler was used to decrease the learning rate over time. The scheduler decayed the learning rate by multiplying the current learning rate by 0.8 every 200 epochs. Similarly to RGAN, noise is sampled from a standard normal random variable $\mathbf{Z} : \mathbb{R}^{T \times 50}$, with 50 noise dimensions for each time step.

COTGAN includes additional parameters which control the martingale regularization and the Sinkhorn algorithm, which computes the estimated distance between the generated and target samples. The number of iterations used to estimate the Sinkhorn divergence was set to 200 with an ϵ set to 10. The martingale regulator λ was set to 0.01.

Discriminator

Layer	Input	Output	Activation
Reshape	B, T, C	BT, 1, C	
Conv1d	BT, 1, C	BT, 64, C	LeakyReLU
Conv1d	BT, 64, C	BT, 128, C	LeakyReLU
Reshape	BT, 128, C	B,T, 128C	
GRU	B,T, 128	B,T, 64	LeakyReLU
GRU	B, T, 64	B, T, J=32	

Generator

Layer	Input	Output	Activation
GRU	B, T, Z	B, T, 64	
GRU	B, T, 64	B, T, 128	
Linear	B, T, 128	B, T, 64	LeakyReLU
Linear	B, T, 64	B, T, 64	LeakyReLU
Linear	B, T, 64	B, T, C	Sigmoid

Table 6.4: The COTGAN architecture. All linear layers include the bias term, and LeakyReLU activations have a negative slope set to 0.01.

6.1.3 Evaluation methods

Several evaluation methods are employed to assess the performance of various models in matching the target distribution. These include the Fast approximation of the Sliced-Wasserstein distance and Prediction Score (PS) (as defined in section 2 and section 4.3.2), as well as a dataset-specific test to measure the $\sin_3$ error, which evaluates the error in the relationship between variables. Additionally, three visualization techniques (PCA, t-SNE, and UMAP) are employed to visualize differences between the generated and target distributions. The tests are performed using 1000 test samples from the target distribution that were not previously used in any GAN model and 1000 generated samples from the GAN model being evaluated.

The Fast approximation of the Sliced-Wasserstein distance was calculated using Algorithm 2 with the test and generated data reshaped to $[NT, C] = [1000 * 100, 3]$, where N is the number of samples considered in the test, T the number of time steps and C the number of data variables. Reshaping the data in this order significantly increases the accuracy and reduces the variance, as illustrated in the Appendix in Figure A.5. To account for the fact that two different sets of a finite number of samples from an arbitrary distribution do not have an SW distance equal to 0, a baseline SW distance, denoted $SW_{baseline}$, is computed as a reference value. The $SW_{baseline}$ provides the distance between two different sets of samples from the target distribution, $\alpha, \beta \sim P_{data}$, and $SW_{baseline} = \widehat{SW}_2(\alpha, \beta)$ represents the lowest possible distance that can be expected. This is performed by having α and β, each consisting of 1000 samples, with the shape $[1000 * 100, 3]$.

The autoregressive model used in calculating the Prediction Score (PS) consisted of two LSTM modules followed by a single linear layer of size 20 following the Sigmoid activation function. The model was trained on a set of 1000 generated samples from the GAN of interest and tested on 1000 samples from the target distribution. To avoid overfitting the generated data, the training set was split into a train set of size 800 and a validation set of size 200, which were monitored during training using early stopping. Early stopping stops the training process if the validation loss keeps increasing while the training loss continues to decrease, avoiding overfitting. The model was trained for 300 epochs for all PS runs using the Adam optimizer with a learning rate of 0.001 and a batch size of 32.

Learning of the spatial-temporal dependencies: The $\sin_3$ error

The $\sin_3$ error is a measure used to assess a model's ability to learn the spatial and temporal dependencies in the sinusoidal dataset. Since $\sin_1$ and $\sin_2$ are independent, only the error in the $\sin_3$ wave is considered. To properly construct $\sin_3$, the model must recognize that $\sin_3$ is an exponential moving average of the other two curves. Since all three curves are generated simultaneously[1], both $\sin_1$

[1]To be precise, they are generated sequentially, but the mentioned curves are the trace of the three points evolving through time. Since the path of each point is predetermined to follow an exact function, it is natural to consider the whole curve that each point traces out instead of just a point

and $\sin_2$ can be used to calculate the $\sin_3$ manually using the following equation:

$$\overline{\sin}_3(t) = \frac{1}{2}\sum_{j=0}^{t} \alpha^{t-j}\left(\sin_1(j) + \sin_2(j)\right), \tag{6.3}$$

where the same exponential α used to create the training dataset is used. This is the same equation as equation 6.2 but with generated $\sin_1$ and $\sin_2$ terms, which are vectors containing values for each time step instead of explicit functions.

The $\sin_3$ error is then calculated as the mean squared error of N-generated samples with the generated $\sin_3$ and the true $\overline{\sin}_3$ as such:

$$\sin_3 \text{ error} = \frac{1}{N}\sum_{N} \left|\left|\sin_3 - \overline{\sin}_3\right|\right|^2 \tag{6.4}$$

at different time steps.

6.2 Results and discussion

The results from this experiment indicate that COTGAN and TimeGAN are the only models which can generate samples that closely match the target distribution. This is supported by the relatively low error in constructing $\sin_3$, the small distance between the generated and target distributions according to the Wasserstein distance estimate (SW), and the fact that the Prediction Score (PS) model does not perform poorly when predicting on the test set containing real data, as indicated by its small error (see Table 6.5). These numerical results are also consistent with the visual results shown in Figures 6.2 and 6.3, which show that the generated samples closely match the target samples in many regions, indicating that the models can cover a significant portion of the target distribution without focusing on specific modes. However, small clusters of target and generated samples do not entirely overlap, suggesting that the models still have room for improvement. The SW distance also supports this, as the $SW_{baseline} = .00034373$ is significantly lower than the reported distance for COTGAN and TimeGAN.

Experiment 1: Numeric results

	$\sin_3$ error	PS* (gen)	PS* (target)	SW*	Time spent training
RGAN	.0328	.0081±.0004	2.1253±.1068	.3819	0h 16m (0.9 s/epoch)
TimeGAN	.0057	.0068±.0005	.0259±.0035	.0233	2h 45m (4.9 s/epoch)
RTSGAN	.006	.0024±.0002	.0628±.0043	.0514	0h 10m (0.6 s/epoch)
COTGAN	**.0053**	.0511±.0034	.0236±.0021	**.0039**	2h 38m (6.6 s/epoch)

Table 6.5: The results of the standard implementation of three models trained on the sinus dataset and tested on a test set of 1000 samples. The Prediction score (PS) model was trained and validated using an 800 : 200 split on generated data and displayed with the standard deviation along with the score when tested on the target dataset. All models had the same baseline SW distance*, equal to $SW_{baseline} = .00034373$. *The PS and SW results are multiplied by a factor of 100 for presentation purposes.

RGAN and RTSGAN suffer from mode collapse but is much more apparent in RGAN. RGAN performs poorly on all tests, which makes this obvious, but the RTSGAN results require further inspection. While the SW results for RTSGAN indicate that the generated and target distributions differ some, it is not that different when compared to TimeGAN and COTGAN. The $\sin_3$ error results do not offer strong evidence of mode collapse either. However, the PS score jump between fake and real datasets (.0024 and .0628, respectively) suggests it, and sample visualizations clearly capture the mode collapse. The high SW distance for RGAN confirms the significant difference between distributions, as seen in Figure 6.2. The PCA plots for RGAN and RTSGAN reveal highly different and concentrated distributions, with RTSGAN displaying more diversity and much more realistic-looking samples. This is clear in the UMAP plot, where the two distributions are globally closer to each other than in RGAN. The same conclusion is reached when looking at actual generated samples in Figure 6.3.

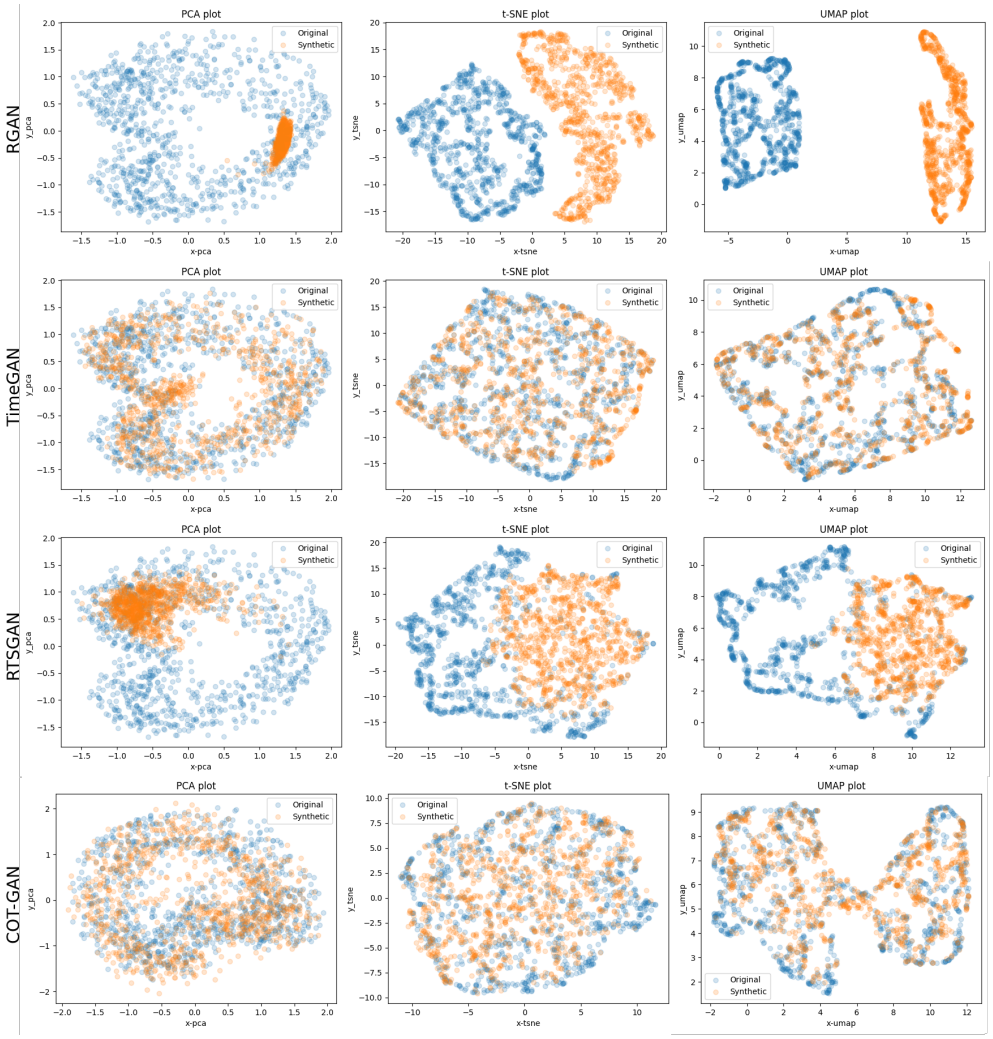


Figure 6.2: Visual distribution plots of real and generated samples utilizing PCA, t-SNE, and UMAP, arranged in a left-to-right column sequence. Each row corresponds to the experimental results obtained from RGAN, TimeGAN, RTSGAN, and COTGAN arranged in a top-to-bottom order.

The t-SNE and UMAP plots provide information about the similarities between the samples and give a sense of the local and global structure of the high-dimensional data. The t-SNE plot for RGAN shows that the generated samples are very different from the target samples, with few samples resembling the target samples. The UMAP plot, which is better at capturing the global structure, shows that the distance between the generated and target samples is considerable, indicating that RGAN generated significantly different samples. For RTSGAN, the generated samples are closer to the target samples but still miss many modes.

Of all the models, COTGAN produces the lowest $\sin_3$ error and SW distance

results. This is apparent in the sample plot comparing generated and target data, where most points overlap or are very close to each other. However, there is still room for improvement despite the low SW distance. COTGAN could achieve a lower $\sin_3$ error and a more accurate shape of the sinusoidal waves, which RTSGAN manages better. As visible in Figure 6.3, the waves generated by COTGAN are almost straight and take sharp turns rather than smooth transitions found in true sinusoidal waves.

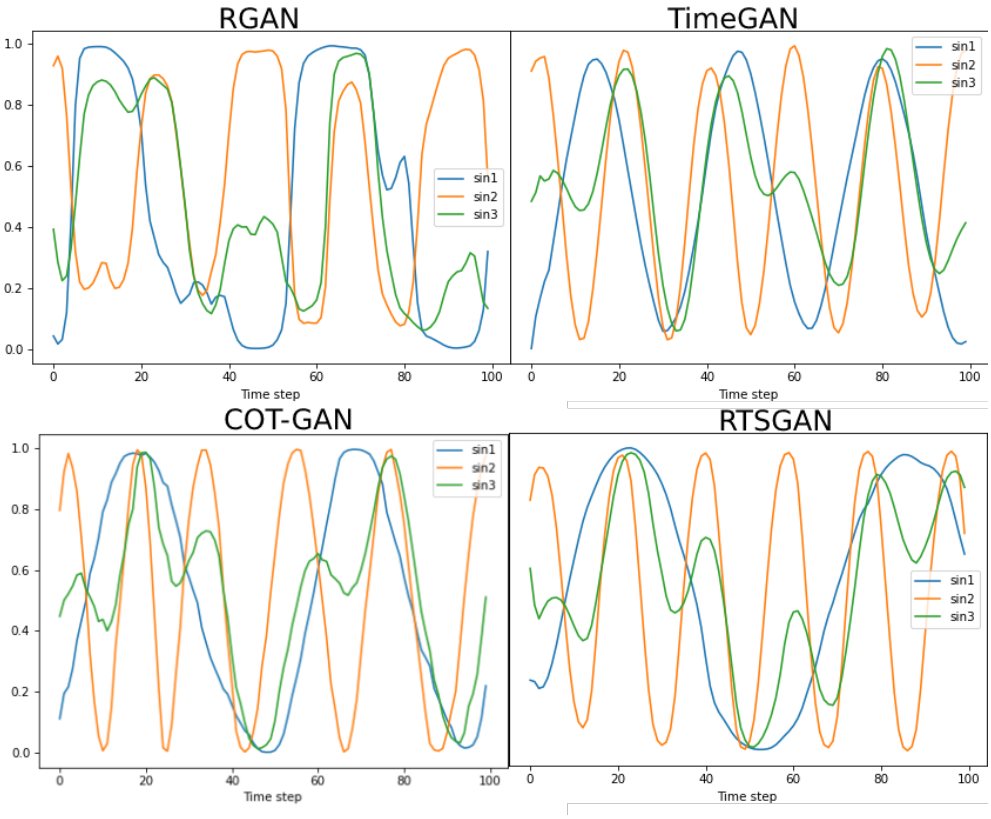


Figure 6.3: Hand-picked generated samples from each of the models. More examples that are randomly selected samples are provided in appendix A.

The RGAN model struggled with mode collapse, but it did learn some aspects of the training data, as evidenced by the smooth, not noisy, oscillating curves in Figure 6.3. Although it was difficult to stabilize and sensitive to hyperparameter adjustments, the RGAN model had a simple architecture that made it fast to train and test, with training time slightly under 1 second per epoch.

TimeGAN and COTGAN are the best-performing models, but they take approximately eight times longer to train per epoch[2] than RTSGAN. Since all models are quite different, it is not fair to assume that they would need to be trained for the

[2]For TimeGAN, this is calculated by the time spent training divided by the number of epochs used to train. The time spent training includes the whole training procedure, such as pre-training of the autoencoder and the supervised network in TimeGAN.

same number of epochs to learn the data. Both TimeGAN and COTGAN typically needed to be trained for more epochs due to their slower convergence rate as a result of the computationally demanding architecture in TimeGAN and the explicit calculation of the Sinkhorn divergence in COTGAN.

Despite this, the overall quality of the samples produced by RTSGAN is high, with a low $\sin_3$ error and PS (real) loss, and they are visually accurate, appearing smooth and within the bounds of 0 and 1 (as shown in Figure 6.3). Resolving the mode-dropping issue for RTSGAN would be highly interesting, as it can learn much faster and utilizes a relatively simple architecture and objective function.

The reason why RTSGAN suffers from mode dropping is not fully understood, as it includes mechanisms intended to combat this problem: Wasserstein distance with gradient penalty [64]. Despite this, RTSGAN still exhibits mode dropping, indicating that other factors may control this. Small experiments, such as investigating if the time series is poorly encoded into the latent space or if the model architecture is inadequate, may shed light on why mode drops occur in RTSGAN. Further analysis of RTSGAN is needed to fully understand the underlying causes of this problem and how to prevent it.

Conducting an ablation study to identify the sources of performance gains in the models would be interesting. However, our results do not align with those reported in the original papers for TimeGAN [60] and RTSGAN [63]. The authors of RTSGAN [63] conducted a comparison study including TimeGAN and RGAN and reported having a better architecture than both. TimeGAN performed a similar study but only with RGAN. Several factors can affect model performance, such as the dataset, implementation, hyperparameters, and training environment, so we do not have enough evidence to confidently argue about the relative performance of the models shown here and why they perform differently. The authors of TimeGAN claim that their model's performance is due to its autoencoder structure and the use of a supervised loss. This argument is consistent with our results, but we do not provide evidence to support this claim. Since we could not solve the mode collapse issue with RTSGAN, we do not discuss its results compared to the others based on architecture design.

6.3 Concluding the preliminary experiment

Choosing an appropriate GAN framework for generating time series poses difficulties due to the lack of comprehensive documentation on their performance with diverse datasets beyond the ones presented in the original papers. Uncertainties persist regarding their capability to generate lengthy sequences, the intricacies of their learning and training processes, and their overall performance. To address this issue, we conducted an experiment on four different time series GAN models—RGAN [57], TimeGAN [60], COTGAN [61], and RTSGAN [63]— and evaluated their performance on a sinusoidal wave dataset.

RQ1.1: What is the comparative performance of different time series GAN models when generating long sequence lengths?

The results indicate that COTGAN and TimeGAN outperformed the other models in terms of generating samples that closely match the target distribution.

COTGAN emerged as the best-performing model, generating samples that closely resembled the target distribution and exhibited dynamics similar to the target data. The visual results demonstrated a close match between the generated and target samples. However, there is still room for improvement as the generated sinusoidal waves deviated from true sinusoidal curves. Despite taking the longest training time, COTGAN achieved the best overall performance.

TimeGAN also performed well in generating long sequence lengths, with relatively low $\sin_3$ error and SW distance results. The generated samples showed a significant overlap with the target samples, indicating good target distribution coverage. However, it trained for a similar amount of time as COTGAN but achieved a bit poorer results with a much more complicated architecture and training instabilities.

On the topic of training instabilities, RGAN and RTSGAN suffered from mode collapse, with RGAN exhibiting more severe issues. The generated samples from RGAN significantly differed from the target samples, as indicated by a high SW distance and poor performance in all tests. RTSGAN exhibited some mode dropping but showcased more diversity and generated more realistic-looking samples compared to RGAN.

In summary, COTGAN and TimeGAN performed better in generating long sequence lengths than RGAN and RTSGAN. COTGAN achieved the lowest $\sin_3$ error and SW distance with a relatively simple architecture, while TimeGAN demonstrated competitive performance but with a much more complex architecture. COTGAN's simpler architecture is thus more desirable for future experiments where interpretability is essential.

Additional analysis and experiments are required to comprehensively understand the performance differences among these models and tackle the challenges of mode collapse and mode dropping, thereby possibly utilizing frameworks that can train much faster and provide better results. The current experiment provided a preliminary overview of the performance of different frameworks, aiding in selecting a suitable framework for the main experiment.

Chapter 7

Experiment: Learning of underlying physical laws

This chapter provides a comprehensive overview of the main experiments conducted, encompassing details regarding the datasets utilized, the evaluation techniques employed, and specific architecture details.

7.1 Datasets

To evaluate the performance of the models, we construct two datasets of dissimilar complexity containing simulations of waves based on the Korteweg-de Vries (KdV) equation. The KdV equation is a fundamental equation in the study of non-linear waves and has been used to model physical systems such as waves in shallow water, optical fibers, and plasma [90]. It is Partial Differential Equation (PDE) describing the evolution of a one-dimensional wave through time and is defined as

$$u_t + \eta u u_x + \gamma u_{xxx} = 0. \tag{7.1}$$

Here $u = u(x,t)$ denotes a function of two variables, x and t, representing space and time dimensions, respectively. Further, η and γ are two arbitrary real-valued scalar parameters that define the simulation's behavior. These will take on the $\gamma = 1$ and $\eta = -6$. The subscripts used on function u denote the partial derivative with respect to that variable.

One special property of the KdV equation is that it has an exact analytic solution for certain initial conditions $u(x,0)$, which describes a wave of a specific shape, called a *soliton*. A soliton is a single wave with a bell-like shape and no ripples that move at a constant speed proportional to its height through time and space. An example of a solution to the KdV equation can be obtained from the initial condition

$$u(x,0) = \frac{1}{2} c \operatorname{sech}^2\left(-x + \frac{P}{2}\right),$$

with a periodic boundary condition $u(P,t) = u(0,t)$ for $t \geq 0$ and P is a period in space. The boundary condition makes the wave appear at the left when exiting on the right boundary. This gives the general analytic solution for all times

$$u(x,t) = \frac{1}{2}c\,\mathrm{sech}^2\left(\left|(x - ct) \bmod P - \frac{P}{2}\right|\right), \tag{7.2}$$

where the sech function is the hyperbolic secant, defined as $\mathrm{sech} = \frac{1}{\cosh}$, c is a positive real number that describes the wave's amplitude, speed, and steepness, $\bmod P$ being the modulo operator limiting the function to be defined within the range $[0, P)$, and $-\frac{P}{2}$ being a term that specifies the initial spatial position of the wave. The fractions and roots in the equation scale the parameters properly to such that it is an exact solution of the KdV equation.

All of the proceeding datasets are based on the soliton wave equation (equation 7.2), but not all are analytical solutions to the KdV equation. This is a consequence of the initial condition not being a single soliton but a combination of two solitary waves or something more complicated. Details around this are discussed in the relevant dataset section, but the soliton wave equation is the backbone for all of them.

7.1.1 Soliton

Representing the simple phenomenon of a single wave moving from left to right, we construct a dataset containing $N = 2000$ soliton waves based on an exact analytic solution to the KdV equation through

$$u(x,t) = \frac{1}{2}c\,\mathrm{sech}^2\left(\left|(x - ct + \frac{P}{4}) \bmod P - \frac{P}{2}\right|\right). \tag{7.3}$$

The solitons are simulated in a spatial domain of $x = [0, 50]$ and a time domain of $t = [0, 10]$, discretized with 120 uniformly-distributed points, respectively. This gives a spatial resolution of $\Delta x = \frac{50}{120} = 0.41\bar{6}$ and a temporal resolution of $\Delta t = \frac{10}{120} = 0.08\bar{3}$. The period P equals the upper boundary of x, $\max(x) = 50$. Initially, the soliton wave described in equation 7.2 centers at $\frac{P}{2}$, so a wave-shifting term $\frac{P}{4}$ is added to move the initial position further to the left. This allows for a longer simulation without the base of the wave hitting the boundary.

The height of each soliton wave is controlled by the parameter c, sampled from a uniform distribution, $c \sim \mathrm{Uniform}(0.5, 2)$. This translates to all solitons waves having their maximal amplitude in the range $[0.25, 1]$.

We structured the dataset as a 3-dimensional matrix with dimensions of $[N, T, C] = [2000, 30, 120]$, where N represents the number of waves, T is the number of time steps, and C represents the number of spatial points of each wave.

7.1.2 Colliding solitary waves

This dataset considers the more difficult problem; simulations of the overtaking collision of two solitons. The simulations are created by defining the initial condition and using numerical methods to approximate the solution for a number of time steps. The initial condition $u(x, t = 0)$ is a sum of two solitons, i.e., $u = u_1 + u_2$, where the subscript $i = \{1, 2\}$ is used to distinguish between the two. A single soliton is expressed as:

$$u_i(x,0) = \left(\frac{-6}{-\eta}\right) \cdot 2 \cdot k_i^2 \cdot \operatorname{sech}^2\left(\left|k_i \cdot \left(\left(x + \frac{P}{2} - P \cdot d_i\right) \% P - \frac{P}{2}\right)\right|\right), \tag{7.4}$$

where the parameters $k_i \sim \text{Uniform}(0.2, 0.7)$ affect the amplitude and steepness of the wave, and d_i controls where the peak of the wave is positioned. For this dataset, they start in the same location in every simulation with $d_1 = 0.3$ and $d_2 = 0.5$. This means that the only parameters which vary between the simulations are the peak and shape of the soliton waves.

We generate $N = 2000$ initial conditions $u(x, 0)$ and model the evolution of the system using the implicit midpoint method [93]. The numerical solutions are conducted on initial conditions with a spatial grid $x \in [0, P], P = 50$, and over a time interval $[0, \mathcal{T}], \mathcal{T} = 10$, with step sizes $\Delta x = \frac{P}{360} = 0.13\bar{8}$ and $\Delta t = \frac{T}{360} = 0.02\bar{7}$. The simulations are solved using a higher spatial and temporal resolution to reduce approximation error. Thus the simulations used when considering this dataset are downsampled versions that fit the desired resolutions $\Delta x = \frac{P}{120} = 0.41\bar{6}$ and $\Delta t = \frac{\mathcal{T}}{30} = 0.\bar{3}$. Solving the simulations using a resolution of 360 allows *perfect* downsampling to many other resolutions, as 360 has many divisors. Thus, it should be flexible and avoid needing to interpolate between values.

Similar to the soliton dataset, we structure the dataset as a 3-dimensional matrix with dimensions of $[N, T, C] = [2000, 30, 120]$, where N represents the number of waves, T is the number of time steps, and C represents the number of spatial points of each wave.

Since the initial conditions of the dataset are created randomly, there will be numerous instances where the two waves do not collide or overtake at any point. If we define the moment when the peaks of the waves are directly aligned above each other (as depicted at time step 20 in Figure 7.2) as the point of overtaking, then there are 253 cases in the dataset where this does not occur. This accounts for approximately 12% of the overall dataset. For a sample to satisfy this definition, it means that the waves at least reach a state where their peaks are over each other during the simulation time. This means it will also cover samples that undergo the complete overtaking. A sample that does not satisfy this overtaking definition can still include a significant interaction between the waves, as demonstrated in rows 4, 5, and 6 in Figure 7.1.

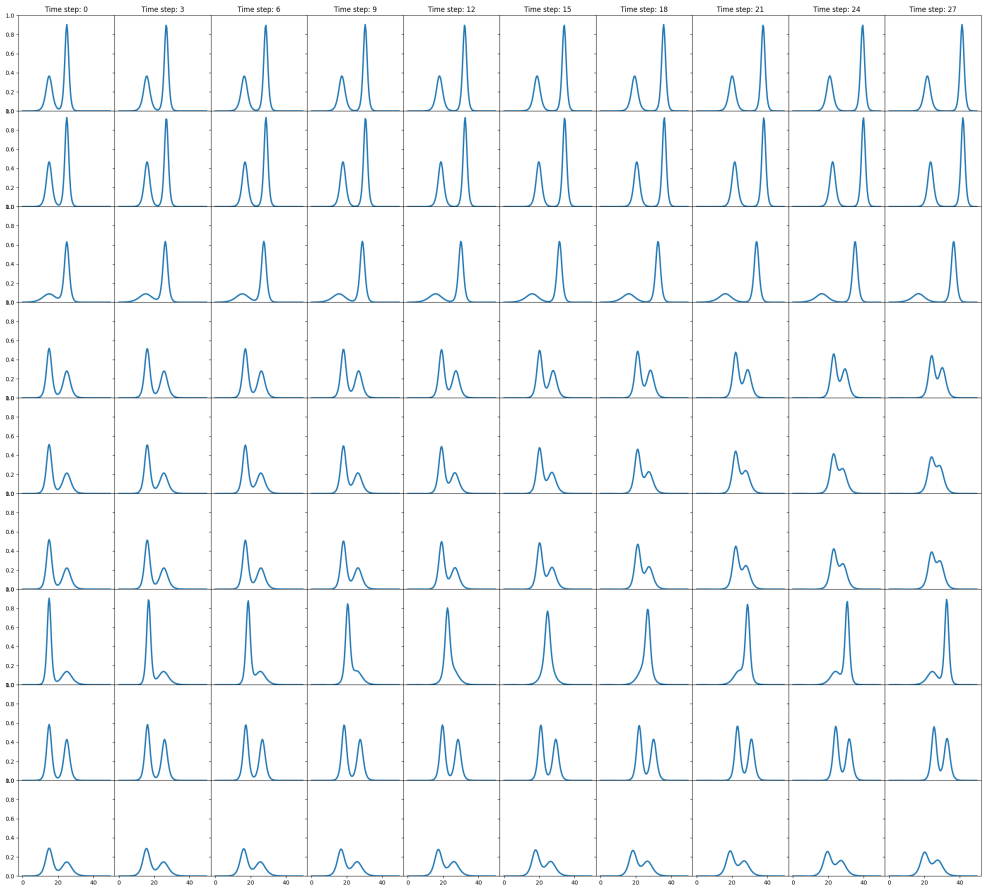


Figure 7.1: Numeric solutions of two colliding solitons for different initial conditions (Time step: 0). The solutions are modeled for 30 time steps ($T = 30$), but only select time steps are shown.

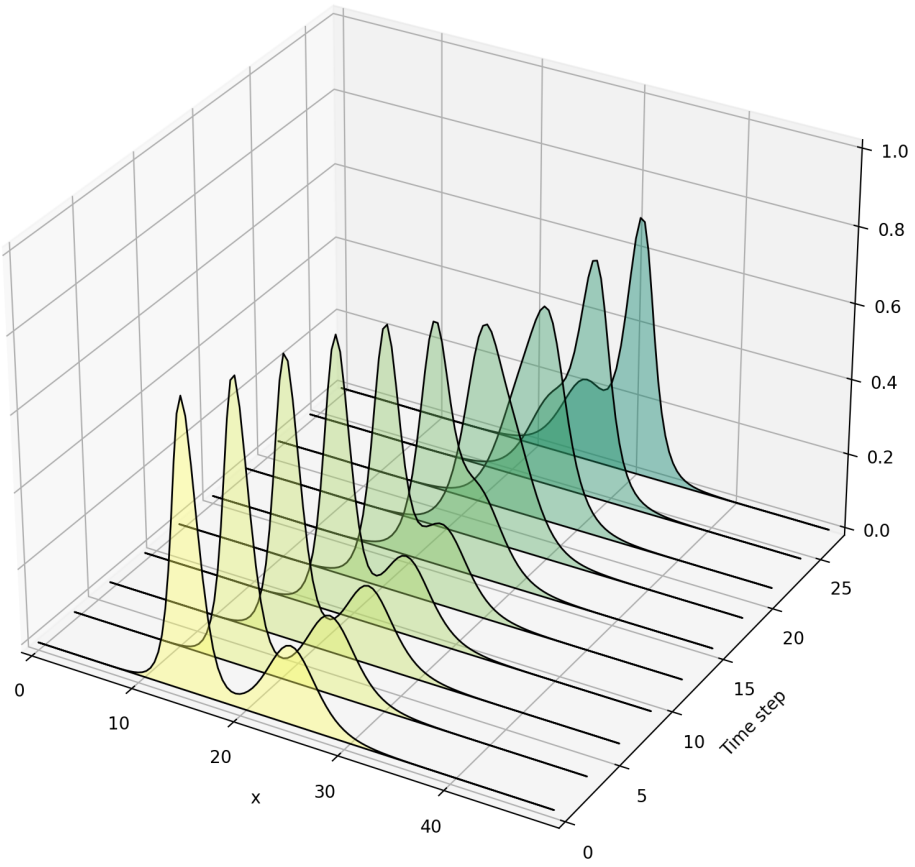


Figure 7.2: An alternative plot of an arbitrary initial solution that shows the evolution and collision of the waves more clearly. Here only every third-time step is visualized to avoid too much overlapping between time steps.

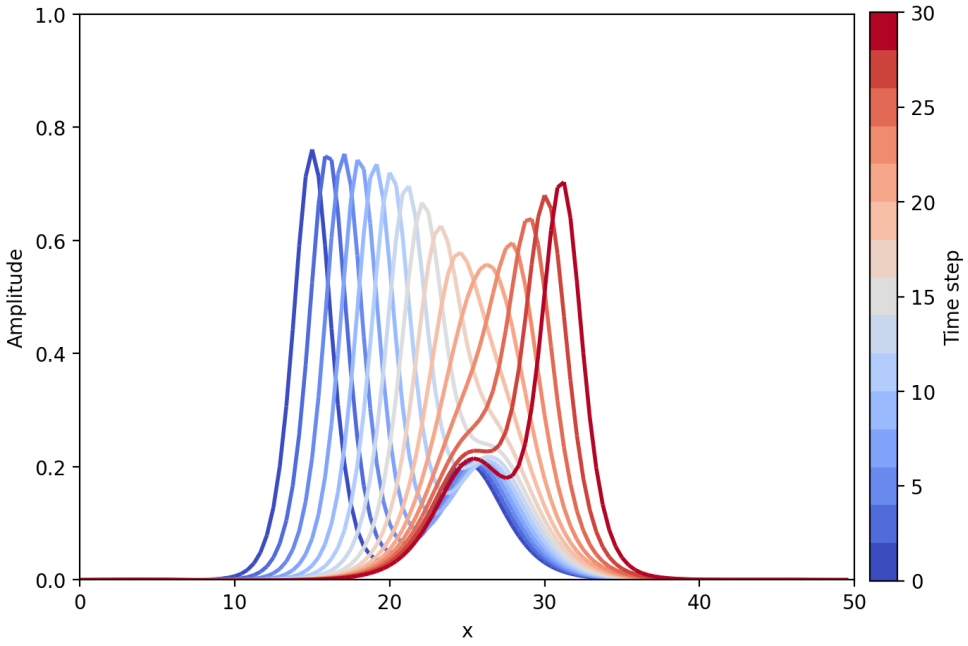


Figure 7.3: The same initial condition is used in 7.2 but visualized in two dimensions with the state of the system for each time step in different colors. The dynamics of the soliton peaks are better captured when plotting this way.

7.2 Evaluation techniques

To evaluate the GAN model during training for architecture modifications and hyperparameter tuning, we employ several techniques to monitor different performance aspects. This includes the general ability to match the target distribution using the Fast approximation of the Sliced-Wasserstein distance (SW) and four custom techniques to measure dataset-specific properties. Three different measurements of conservation error and one estimating the distribution associated with the soliton peaks. Exactly how they are defined is dependent on the dataset, which is detailed in the following section.

7.2.1 Distribution of soliton heights

A dataset-specific test is considered to check whether the models are subject to mode drop and capture the target distribution correctly. The test assesses the null hypothesis that the target and generated wave peaks from the initial condition follow the same distribution. The wave peak is the only variable that is randomly assigned when creating the initial conditions and should follow a uniform distribution in a specific range.

A set of 1000 initial condition wave peaks from the target and generator is considered for testing the null hypothesis. The target samples are easily obtained by creating new initial conditions (no need to solve the KdV equation) and generated samples by using the generator and only considering the first time step.

To compare the two sets of data, the Kolmogorov-Smirnov (KS) test [94] is used. The test is non-parametric, which means it does not make any assumptions about the underlying distribution of the data and allows for comparing any two sample sets. The KS test will be used to test the null hypothesis that the two collections follow the same distribution. In the evaluation of the model, a two-tailed p-value is utilized with a confidence level of 0.05. However, the p-value is not weighted as a significant factor in the overall assessment. This is because the objective of the test is not to ascertain the truth of a specific hypothesis but rather to indicate the probability that the two distributions conform to the same underlying distribution. In this context, a higher p-value is more favorable.

Soliton

To obtain the wave peaks in the initial conditions, the maximal value of each initial state is considered, $\max\ u_i(x,0)$, where $i = 1, 2, \cdots, 1000$ for both generated and target samples. The p-value is then obtained by calculating the KS test.

Colliding solitary waves

Since the initial condition here consists of two distinct solitons, the KS test is performed two times: one for each soliton wave. Then the average p-value of the

two is reported[1]. Obtaining the wave peaks is, however, not easily achieved since there is an issue of the initial state containing two solitary waves: each soliton influences the peak and position of the peak of the other. This is true regarding all systems containing two solitons because any solitary wave is always positive and non-zero, $\text{sech}(x) > 0$, making it challenging to obtain the actual parameters that perfectly describe each soliton based on the initial condition.

A simple but imperfect solution is to disregard the dependence between the two waves, as the impact often is very tiny. The maximal peak increase is when both solitons are made with the lowest amplitude and steepness factor ($k_i = 0.2$). In this scenario, the soliton peaks are increased by 7%.

By assuming independence, the wave peaks are obtained by indexing the same index used to define where each soliton should be positioned when creating the dataset, precisely $d_1 = 0.3$ and $d_2 = 0.5$. The wave peaks from each soliton, K_1, and K_2, are then obtained for each sample as such:

$$\begin{aligned} K_1^{\xi} &= u_i(d_1 * 120, 0) \\ K_2^{\xi} &= u_i(d_2 * 120, 0), \end{aligned} \tag{7.5}$$

where 120 is the total number of indexes used to discretize the spatial domain x, and $\xi = \{\text{target}, \text{gen}\}$ represents which sample type to consider, and $i = 1, 2, \cdots, 1000$ being the sample index. The reported p-value is a result of the average when computing the statistic for equal-positioned soliton waves, performed as such:

$$\begin{aligned} p_1 &= KS(K_1^{\text{target}}, K_1^{\text{gen}}) \\ p_2 &= KS(K_2^{\text{target}}, K_2^{\text{gen}}) \\ \text{Reported } p\text{-value} &= \frac{p_1 + p_2}{2} \end{aligned} \tag{7.6}$$

7.2.2 The measure of energy conservation loss

For a physical system governed by the KdV equation, conserved quantities are independent of time, such as mass, momentum, and energy [25]. These are known as conservation integrals and can be efficiently calculated for all generated samples. Let the discrete spatial grid be denoted by $u(x = i\Delta x, t)$, where $i = 0, 1, ..., P-1$. Then, the conservation integrals (sums in the discrete case) can be estimated using:

[1]The average of the p-values is employed as a heuristic measure and is not literally meant to test the null hypothesis. Instead, it is recommended to utilize Fisher's combined probability test [95] for an accurate evaluation.

$$\mathcal{H}_{\text{Mass}} = \Delta x \sum^{P} u \tag{7.7}$$

$$\mathcal{H}_{\text{Momentum}} = \Delta x \sum^{P} u^2 \tag{7.8}$$

$$\mathcal{H}_{\text{Energy}} = \Delta x \sum^{P} -\frac{1}{6}\eta u^3 + \frac{1}{2}\gamma^2 u_x^2 \tag{7.9}$$

Each conservation integral $\mathcal{H}$ (short for Hamiltonian) represents an array of the corresponding conserved quantity associated with each time step. That is, $\mathcal{H}(t_0)$ and $\mathcal{H}(t_5)$ give the energy of the system at time steps 0 and $5\Delta t$, respectively. The final conservation integral, $\mathcal{H}_{\text{Energy}}$ (equation 7.9) is also dependent on the initial values for η, and γ, which are equal to 6 and 1, respectively.

The $\mathcal{H}_{\text{Mass}}$ conservation integral is the easiest to grasp and does not require delving into the theory. The area under the curve of a wave or any other solution is directly proportional to its mass. As a result, if one point in the wave decreases in height, the height of another point or multiple other points must simultaneously increase by the same amount, similar to how an old-fashioned scale works, where pressing one end results in the opposite end moving in the opposite direction. This means that any energy used to change the height of one point has an inverse effect on the other points.

We define the total error of energy conservation for the different quantities as the deviation of the energy at each time step from the initial condition, given by:

$$\text{Conservation error} = \sum_{t=0}^{T} |\mathcal{H}(t) - \mathcal{H}(t_0)| \tag{7.10}$$

We denote the mean error associated with each conservation integral with a hat "^," as such: $\hat{\mathcal{H}}_{\text{Mass}}$, $\hat{\mathcal{H}}_{\text{Momentum}}$, and $\hat{\mathcal{H}}_{\text{Energy}}$.

7.3 Architecture details

This section contains essential implementation details needed to reconstruct the experiments, such as the architecture design and choice of hyperparameters. We only present details of the best-performing models as this is the most relevant for the analysis. Still, a visual collection of other models that have been tested is provided in the Appendix.

There are a total of four models that will be showcased, consisting of two main experiments for each dataset - soliton and colliding solitary waves. And in addition, two supporting experiments aimed at determining the maximum potential performance of the model using the colliding solitary waves dataset.

7.3.1 Soliton and colliding solitary waves

The models used for the soliton and colliding solitary wave datasets shared much of the same architecture and were trained similarly. This enables a more precise comparison of their results instead of using two very different ones. The best-performing model for each of the datasets comprised a generator with two GRU layers and three fully-connected layers. The discriminator was more optimized for finding features, including two one-dimensional convolutions and two GRU layers. Specific details concerning the arrangement of the modules and the number of neurons used in each respective module are summarized in Table 7.1.

The models were trained for 600 epochs using the Adam optimizer with betas $\beta_1 = 0.5$ and $\beta_2 = 0.9$ and an initial learning rate of 0.01 for both the generator and discriminator with a mini-batch size of 90 samples. To assist the models in fine-tuning the samples after being able to create general features, the learning rate was decayed every 200 epochs by multiplying it by 0.8. For the model trained on the soliton dataset, input consisted of a noise vector $\mathbf{Z} : \mathbb{R}^{T\times 10}$ drawn from a standard normal distribution with 10 dimensions per time step, where T denotes the number of time steps used in the dataset, e.g., T=30. For the model trained on colliding solitary waves, the noise vector was instead drawn from a uniform distribution in the range -1 and 1, as this resulted in better results.

Most hyperparameters were similar, but the soliton and colliding wave models varied in two sensitive hyperparameters related to the COTGAN objective function. Specifically, the soliton model employed an entropic regularizer ϵ of 0.7, while the colliding waves model used 0.8. Also, the soliton model utilized a martingale penalization of λ equal to 0.01, whereas the colliding waves model used 0.05. Both models utilized 200 iterations for calculating the Sinkhorn divergence using the Sinkhorn algorithm.

Figures A.11 and A.7 in the Appendix presents a comprehensive view of the other models that were examined, along with their respective hyperparameters and performance metrics.

7.3.2 Physics-informed GAN

A new model was developed to test how the best-performing colliding solitary waves model performs when being informed about the errors in the underlying physical laws. The model utilized the same architecture as the best-performing colliding solitary waves model, described in Table 7.1, and was trained with identical hyperparameters. The new model included an additional term in the objective function to penalize samples for not following the conservation laws, called the conservation regularization term.

The conservation term was defined as a summation of all the mean conservation errors for a minibatch of generated samples $\mathbf{y}$, precisely as follows:

$$\mathcal{H}_{error}(\mathbf{y}) = \hat{\mathcal{H}}_{\text{Mass}} + \hat{\mathcal{H}}_{\text{Momentum}} + \hat{\mathcal{H}}_{\text{Energy}} \tag{7.11}$$

Discriminator

Layer	Input	Output	Activation
Reshape	B, T, C	BT, 1, C	
Conv1d	BT, 1, C	BT, 64, C	LeakyReLU
Conv1d	BT, 64, C	BT, 128, C	LeakyReLU
Reshape	BT, 128, C	B,T, 128C	
GRU	B,T, 128S	B,T, 64	LeakyReLU
GRU	B, T, 64	B, T, J=32	

Generator

Layer	Input	Output	Activation
GRU	B, T, Z	B, T, 64	
GRU	B, T, 64	B, T, 128	
Linear	B, T, 128	B, T, 64	LeakyReLU
Linear	B, T, 64	B, T, 64	LeakyReLU
Linear	B, T, 64	B, T, C	Sigmoid

Table 7.1: COTGAN architecture utilized for main experiments. The capital letters denote static quantities, where B(=90) denotes a batch of samples and Z(=10) is the noise dimension associated with each time step. All linear layers include the bias term, and all LeakyReLU activations have a negative slope equal to 0.01.

This term was then added to the COTGAN objective function, forming the new objective:

$$\widehat{\mathcal{W}}^{mix,L}_{c^{\kappa}_{\varphi},\epsilon}(\mathbf{x},\mathbf{y}) - p_{M_{\phi_2}}(\mathbf{x}) - \psi\mathcal{H}_{error}(\mathbf{y}), \tag{7.12}$$

Here, ψ denotes the weighting term that determines the importance of the conservation error. For the best-performing model, ψ was set to 10. A relatively high value was found to be beneficial as it forces the model to consider minor conservation errors.

The conservation regularization was not applied in the GAN training for the first 160 epochs to allow the model to first learn the global features before fine-tuning the samples. The model was trained for a total of 800 epochs.

7.3.3 Autoencoder

The autoencoder used to assess the capacity of the generator network for the colliding solitary waves dataset was constructed with the identical generator architecture as described in Table 7.1. The generator functioned as the decoder and operated with an appropriate encoder. The encoder consists of a GRU and three dense layers. The final layer employs the *Tanh* activation function to map the output to the range of -1 to 1, equivalent to the generator's sampling space in the GAN. This means the generator operates on the same information space but with different sample representations. Further details about the encoder architecture are listed in Table 7.2.

Encoder

Layer	Input	Output	Activation
GRU	B, T, C	B, T, 64	
Linear	B, T, 64	B, T, 256	LeakyReLU
Linear	B, T, 256	B, T, 256	LeakyReLU
Linear	B, T, 256	B, T, Z	Tanh

Table 7.2: The encoder in the Autoencoder architecture. The capital letters denote static quantities, where B(=32) denotes a batch of samples, C(=120) is the feature dimension, and Z(=10) is the noise dimension associated with each time step. All linear layers include the bias term, and LeakyReLU activations have a negative slope set to 0.01.

The autoencoder was trained to encode and reconstruct samples from the colliding solitary waves dataset. The encoder takes in target samples and encodes them into a latent representation of the same size as the noise dimension used in the GAN. The generator then reconstructs the samples from the latent representations.

The mean squared error loss was used to train the autoencoder, with a learning rate of 0.001, the Adam optimizer with betas $\beta_1 = 0.5$ and $\beta_2 = 0.9$, and a batch size of 32 for 600 epochs. The colliding solitary waves dataset was split into training and validation datasets, with the validation set accounting for 20% of the samples. The validation set was used during training to monitor the model performance and avoid overfitting.

Chapter 8

Results and Discussion

In this chapter, the performance results of the models on two datasets, Soliton and Colliding solitary waves, are presented and interpreted. Furthermore, two supplementary experiments are discussed, followed by a general discussion on the GAN's capability to adhere to conservation laws.

This chapter only reports results obtained from the best-performing COTGAN models. Information about the performance of other suboptimal models with different hyperparameters is displayed in Appendix A.

8.1 Soliton

By visually inspecting the generated samples in Figure 8.1, they can be identified as solitons that keep their structure and evolve rather stable through time. The samples presented appear to be of different heights and evolve according to their initial height; a short soliton travels a short distance, while taller solitons move further. Not all heights seem to remain unchanged when evolving through time; for instance, the second sample in the first column appears to shorten for each time step. Initially, the results seem quite promising, but to investigate what such results hold in general, we evaluate the performance on the metrics used.

Model	SW	$\hat{\mathcal{H}}_{\text{Mass}}$	$\hat{\mathcal{H}}_{\text{Momentum}}$	$\hat{\mathcal{H}}_{\text{Energy}}$	p-value
Soliton	6.43e-5	.0418	.0353	.0168	.1557

Table 8.1: Test results when learning the propagation of a single soliton wave. The SW is compared against the baseline, $SW_{\text{baseline}} = 3.86\text{e-}5$.

An explanation for why the height of the soliton wave sometimes appears inconsistent when evolving through time, as shown in Figure 8.1, is because of the low spatial resolution. The spatial domain $x = [0, 50]$ is discretized with step-size $\Delta x = 0.13\bar{8}$, representing the whole domain by only 120 discrete points. As a result, the peak of a soliton may not appear precisely on our defined discrete points but rather in between two adjacent points. Instead of visualizing the actual soliton

peak, points close to the peak are visualized instead, creating the illusion that the soliton's height is fluctuating. The solitons are also relatively spiked, which adds to the effect.

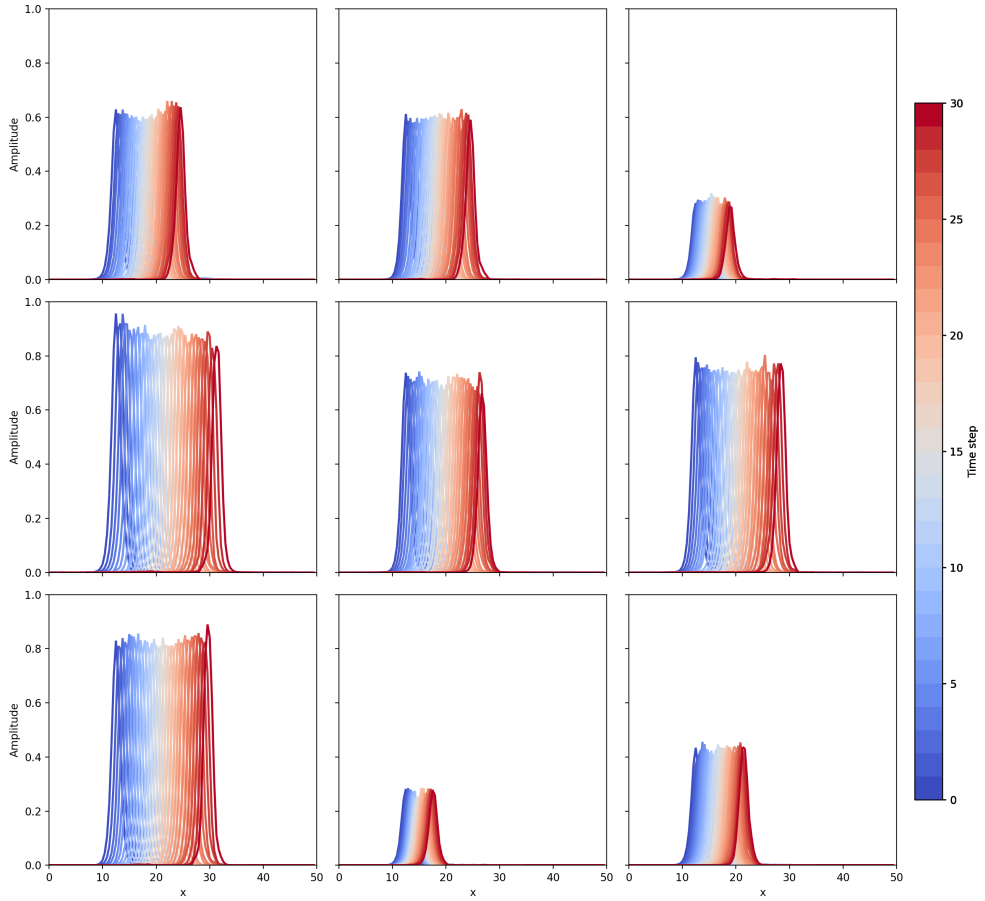
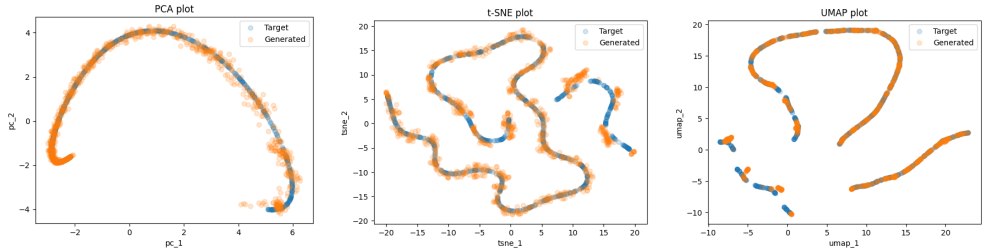


Figure 8.1: A random selection of nine soliton samples that were randomly generated, with each time step of the wave state represented in a different color, ranging from the initial state (blue) to the final state (red).

According to Table 8.1, the computed Sliced-Wasserstein distance (SW) between the generated and target soliton simulations was 6.43e-5, which is very small. The difference between this value and the baseline value of $SW_{\text{baseline}} = 3.86\text{e-}5$ is negligible, and it is possible that the metric may not be sensitive enough to detect such minor differences. The fact that the computed SW value is so similar to the measured baseline suggests that the generated data closely resembles the target data. However, additional specific and accurate metrics are needed for more precise results. A test was performed to evaluate the accuracy of the SW distance calculation between two sample collections from the same distribution, and the results of several runs revealed a relatively high variance. The model result reported here falls within this range, indicating that the SW calculation is not precise

enough to assess the model performance accurately.

The distribution plots in Figure 8.2 demonstrate the similarities between data at different scales. UMAP provides a global comparison and shows that the generated samples mainly capture significant differences from the target samples, except for some small target stripes. The generated and target samples are close to each other when they are similar and far apart when they are different, which is desirable. On the other hand, t-SNE does not capture the global structure but gives more detailed local information on how well the generated samples fit. The local information is visible in more detail as the plot does not need to be zoomed out, unlike UMAP, which remains compact. The local structure reveals that many generated samples overlap with the targets, and several samples differ slightly. A small portion of target samples appear not covered (clean blue stripes), and several small dense groups of generated samples are visible. These groups could indicate the overproduction of a specific soliton wave, which also can be seen in the slight overproduction of high peaks in Figure 8.3b. The PCA plot broadly conveys the same information.



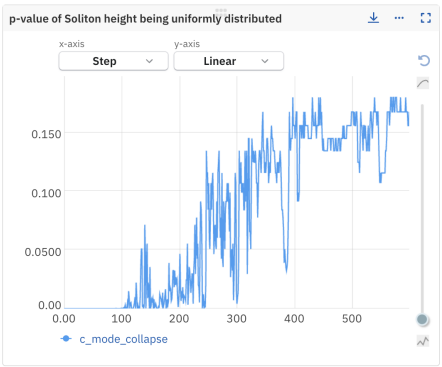
(a) PCA **(b)** t-SNE **(c)** UMAP

Figure 8.2: High-dimensional target (blue) and generated (orange) samples visualized on a 2-dimensional plane using different methods. Each plot suggests the generated samples are fairly close to the target samples.

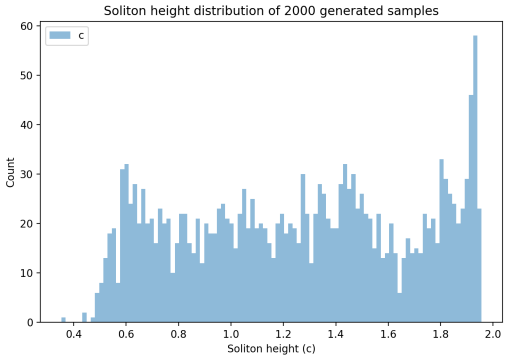
The model is able to produce solitons with various heights, and the distribution of these heights is almost uniform. The p-value of the model's ability to generate uniformly increases over time, as shown in Figure 8.3a, indicating that the model is getting better at generating solitons of different heights. The height distribution of the trained model, as displayed in Figure 8.3b, is very similar to a proper uniform distribution, with a p-value of 0.1557 and covering almost all modes. However, some heights are below the target range ($c < 0.5$), and there is a shortage of peaks very close to the upper limit. There is a slight over-representation of soliton heights in the upper range, but overall, the model captures the continuous range of soliton heights well.

The authors of COTGAN intended to develop a GANs that employs optimal transport while also respecting causality. They included a causality term, the martingale regularization p_M, in the objective function to prioritize causal transport plans by measuring the associated causality error. By tracking p_M, we can determine how well the model respects the causality constraint during training. The

graph in Figure 8.4 displays the relative causality error p_M during training, which plateaus after 200 epochs. The plateau consists of unnoticeable improvements until the final epoch, where $p_M = 2.13$. This suggests that the model can produce relatively causal transport plans or that the discriminator capacity is insufficient for the error to decrease further. Regardless of the discriminator capacity, the model seems to learn to respect causality more effectively during training until it reaches the plateau. This indicates that the model has hit a barrier in which the transport plans are not able to be more casual. Whether or not the use of causal optimal transports has had significant importance for the generator in learning to create good samples is not entirely certain, but Xu *et al.* [61] reported that causal transport plans are beneficial.



(a) The evolution of the p-value obtained from the KS test during the training process of 600 epochs. The increasing p-value with each epoch indicates that the generated soliton heights are getting closer to the desired distribution.



(b) A visual plot of the density distribution of the generated soliton heights for the fully-trained model. The KS test associated with this distribution presented a p-value of .1557, indicating that the generated sample does resemble some of the target distribution.

Figure 8.3: The p-value evolution (**a**) and density distribution (**b**) of 2000 generated soliton heights. The target soliton heights follow a uniform distribution, precisely $c \sim Uniform[0.5, 2]$.

We analyze how conservation errors change throughout the training process of the regular GAN to determine whether the model follows the underlying conservation laws. The conservation errors (displayed in Figure 8.5) for the three quantities, namely $\hat{\mathcal{H}}_{\text{Mass}}$, $\hat{\mathcal{H}}_{\text{Momentum}}$, and $\hat{\mathcal{H}}_{\text{Energy}}$, seem to be closely connected and vary together in distinct ranges. For the first epochs, the errors fluctuate considerably but gradually decrease and stabilize over time while continuing to reduce the error. This is due to the model adjusting to finer details and a decreasing learning rate (learning rate decays at epochs 200 and 400), intentionally implemented to prevent over-correction and ensure smoother solitons. The fact that the model can improve the conservation losses without being explicitly informed about them is a positive sign, suggesting that it can identify and correct critical underlying elements in the data.

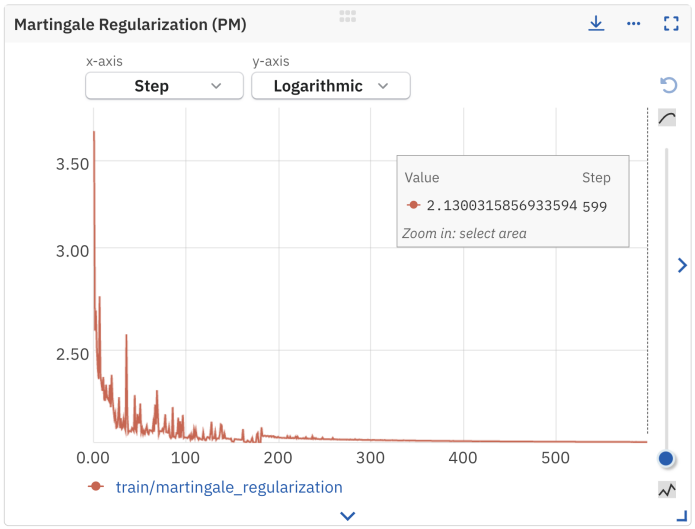


Figure 8.4: The causal error tracked by the martingale regularization p_M quickly drops and stabilizes to values near two. Even though the y-axis is logarithmic to visualize small differences more clearly, the graph appears to flatten out rather quickly with unnoticeable improvements. The box presents the p_M error at the last epoch.

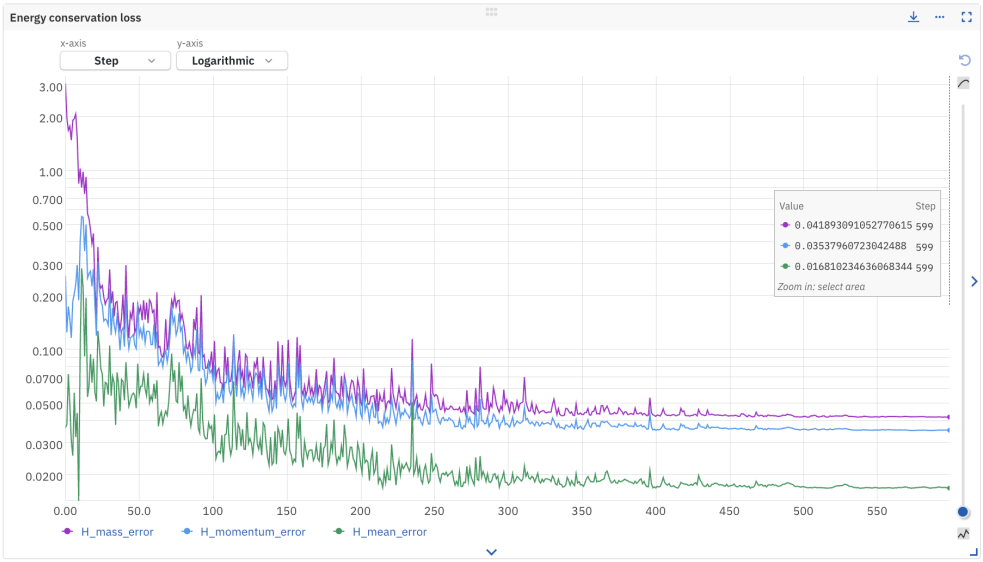


Figure 8.5: During the 600-epoch training period on the soliton dataset, all three measures of energy conservation loss decreased and stabilized. The values at the last epoch are reported in Table 8.1. The y-axis is in the logarithmic scale to make small changes more prominent.

Parts of the conservation errors stem from the model generating unrealistic soliton dynamics and difficulties modeling the last few time steps. Figure 8.6

shows that the mean and median errors generally stay consistent until the final time steps, where they suddenly increase. Since both the mean and median in all plots grow, not just a few samples contribute a lot to the loss, but the general sample is generated poorly. It can be concluded that the final time steps are more problematic to produce and contribute negatively to the conservation errors. The difference between the mean and median also suggests that samples with exceptionally high relative errors (anomalies) exist, but many samples are more consistent in achieving a low error. Figure A.8 in the Appendix displays a collection of these anomalies, where the model deviates from the initial condition and rapidly transitions to modeling smaller or larger waves.

Thinking that the conservation losses could be minimized by enhancing the network architecture is a natural thought. Still, we found that several experiments with larger capacity generators and different hyperparameters did not obtain better results, considering the performance of the conservation losses and other evaluation methods. Figure A.7 provides a broad overview of some of the architecture changes and hyperparameters tested along with their performance.

Regular GAN: random samples

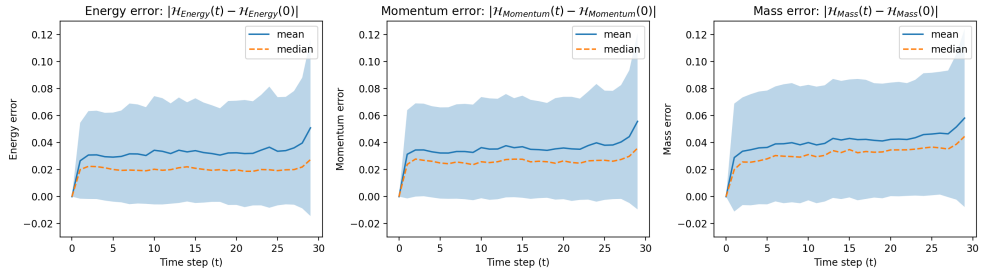


Figure 8.6: Statistics considering the different conserved quantities with respect to every time step of 2000 samples. The blue-shaded region represents one standard deviation from the mean. The mean and median errors show that the error increases with each time step.

Overall, the model performs exceptionally well on the soliton dataset. It can generate numerous solitons that appear smooth, with the peaks remaining relatively stable and following the expected evolution over time, closely resembling the actual soliton solutions. The generated solitons are uniformly distributed, suggested by the relatively large p-value. However, visualizations using PCA and t-SNE demonstrate that some differences exist among samples, meaning that the model does not entirely capture all soliton solutions and could be improved further. The conservation losses are moderately low, given that errors are amplified by some imperfection resulting in anomalies. There is also a slight accumulation of errors during the simulation, but not too significant.

8.2 Colliding solitary waves

By inspecting a few generated samples in Figure 8.7, it is clear that the GAN gets several properties right. The two waves resemble smooth solitons and evolve across with speed proportional to their height, as demonstrated in the top right. The nine samples appear evenly distributed with different initial conditions and evolve proximate to the expected behavior. It is also clear that the evolution of the collision is not perfect, as the bottom row of samples reveals ripples in the waves during the collision and spontaneous drops in the peak during no collision.

Model	SW	$\hat{\mathcal{H}}_{\text{Mass}}$	$\hat{\mathcal{H}}_{\text{Momentum}}$	$\hat{\mathcal{H}}_{\text{Energy}}$	p-value
Soliton	6.43e-5	.0418	.0353	.0168	.1557
Colliding	8.27e-5	.1294	.0853	.0535	.2723

Table 8.2: Test results when learning to simulate two colliding solitary waves, with the Soliton results reported in Table 8.1 included for comparison. The SW is compared against the baseline, $SW_{\text{baseline}} = 2.47\text{e-}5$.

Similar to the soliton dataset, the SW(= 8.27e-5) obtained is considered low and very close to the $SW_{\text{baseline}} = 2.47\text{e-}5$. This indicates that the generated samples are so similar to the target samples that the metric is no longer accurate enough to assess their similarity further, or insufficient samples were used. Figure A.12 in the Appendix illustrates this further as the monitored SW during training achieves a distance smaller than the baseline at some point.

Compared to the single soliton dataset, the distribution plots shown in Figure 8.8 are very different. This is due to the collision of solitary waves causing a much more diverse set of scenarios than just a single propagating wave, remembering that each dot represents a complete simulation. The UMAP plot demonstrates that the generated samples are highly similar to the target samples, with no apparent outliers, covering all distinct cases. t-SNE provides more detailed local information and reveals a similar relationship between generated and target samples. However, t-SNE also allows us to identify slightly uncovered target samples close to generated samples. It is unclear whether this is due to the generated samples having initial conditions that are very similar but not precisely the same, resulting in non-overlapping points, or if the generated samples were falsely simulated. The PCA plot highlights additional lonely targets and generated samples but also a hat-shaped boundary of target samples that is not evenly covered. It also shows three highly dense regions of generated samples, which could indicate the uneven distribution of initial conditions or be an artifact of the PCA algorithm. Similar dense regions are not visible in the other plots.

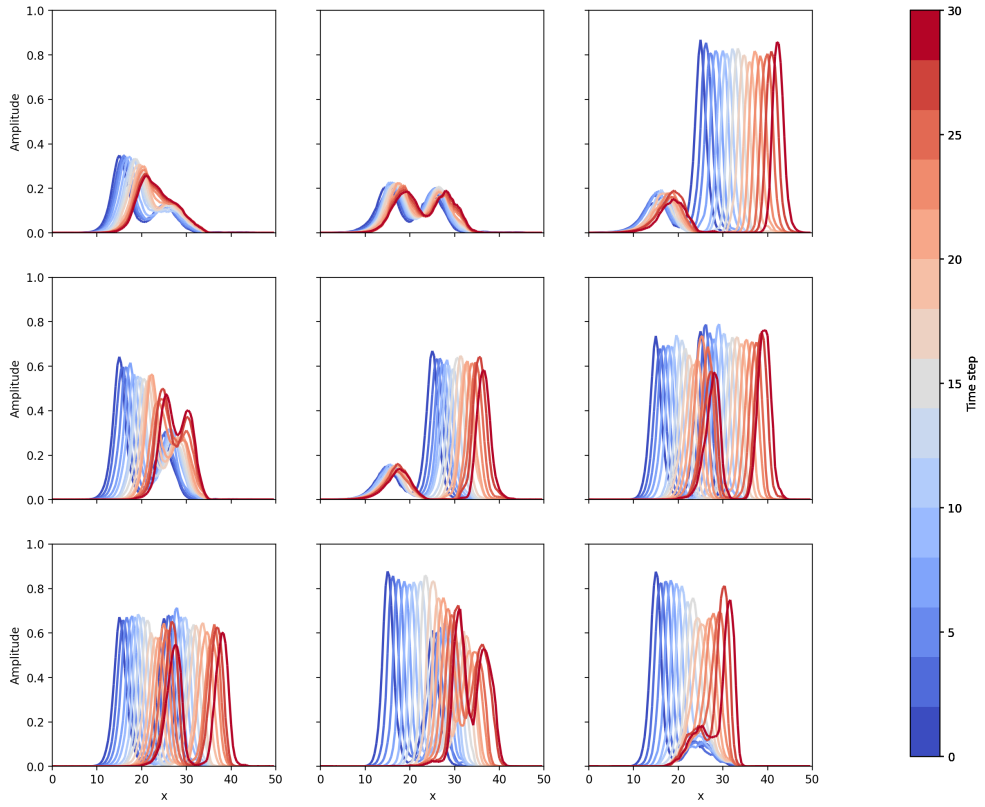
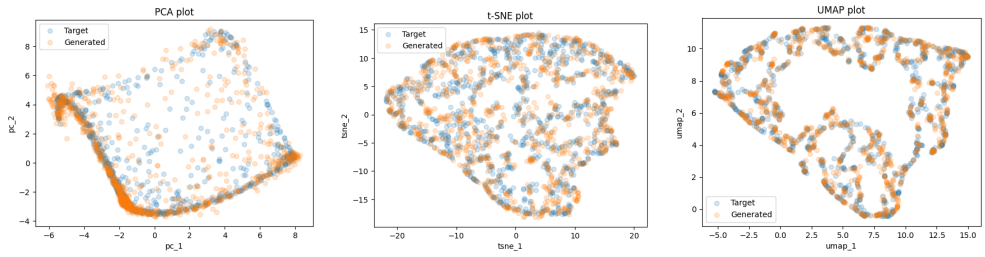


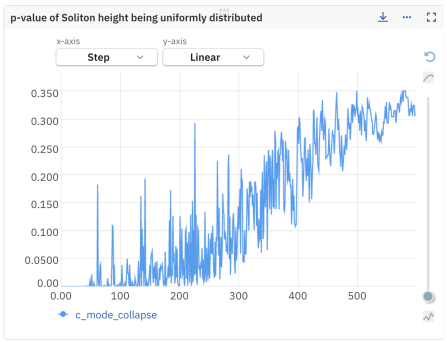
Figure 8.7: A random selection of nine colliding solitary wave samples that were randomly generated, with each time step of the system represented in a different color, ranging from the initial state (blue) to the final state (red). Only every second time step is displayed to visualize wave interactions more clearly.



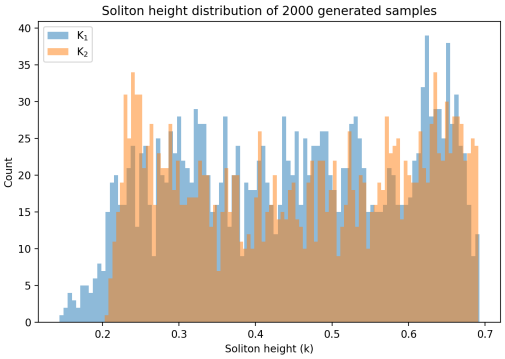
(a) PCA **(b)** t-SNE **(c)** UMAP

Figure 8.8: High-dimensional target (blue) and generated (orange) samples visualized on a 2-dimensional plane using different methods. Each plot suggests the generated samples are fairly close to the target samples.

The idea that the dense regions of generated samples seen in the PCA plot (Figure 8.8a) are due to something other than the generation of uneven modes is supported by the soliton peak distributions (displayed in Figure 8.9b). Visually, the distributions appear to be evenly distributed across different peaks. This corresponds well with the p-value of 0.2723, indicating that we can not confidently reject the hypothesis that the distributions are even. Additionally, the evolution of the p-value over time (Figure 8.9a) indicates that the generated samples became more consistent with the target distribution. The distribution plot (Figure 8.9b) shows that the first soliton wave (denoted K_1) includes soliton peaks below the target range, which shows that the model could still be improved. The distribution of the second soliton wave (denoted K_2) is however much more accurate and fits the target distribution well.



(a) The evolution of the p-value obtained from the Kolmogorov-Smirnov (KS) test during the training process of 600 epochs. The increasing p-value with each epoch indicates that the generated initial conditions are getting closer to the desired distribution.



(b) A visual plot of the density distribution of the generated soliton heights for the fully-trained model. The KS test associated with the two distributions (K_1, K_2) received an average p-value of 0.2723, indicating that the generated sample does resemble some of the target distribution.

Figure 8.9: The p-value evolution (**a**) and density distribution (**b**) of 2000 generated initial conditions and their respective soliton heights. The target soliton heights follow the same uniform distribution, precisely $k \sim Uniform[0.2, 0.7]$.

Matching results are obtained from the causality error p_M for the colliding solitary wave dataset as the single soliton dataset. Figure 8.10 displays the same dynamics of the monitored loss, a quick drop before stabilizing with unnoticeable improvements for the rest of the training period. As a smaller causality error ($p_M = 0.386$) has been observed when training other models with different hyperparameters but obtained poorer results on the SW and conservation errors, it is hence unclear if the causality term contributes to the learning process.

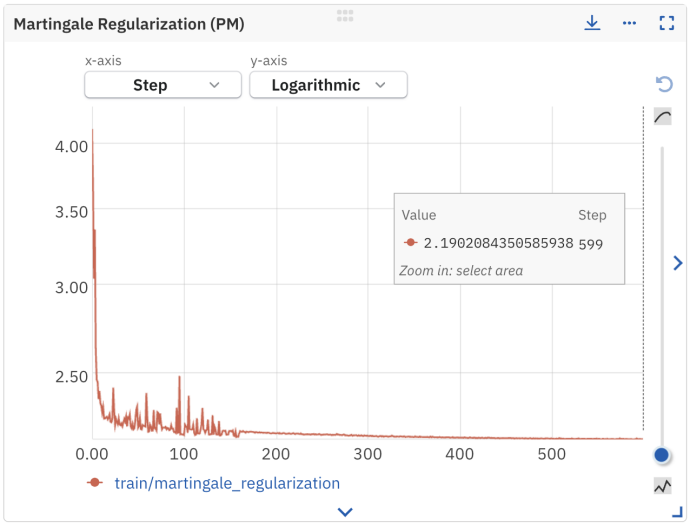


Figure 8.10: The causal error tracked by the martingale regularization p_M quickly drops and stabilizes to values near two in the colliding solitary waves dataset. Even though the y-axis is logarithmic to visualize small differences more clearly, the graph appears to flatten out rather quickly with unnoticeable improvements. The box presents the p_M error at the last epoch.

We analyze how conservation errors change throughout the training process to determine whether the model follows the underlying conservation laws. Figure 8.11 displays the conserved error related to the different quantities, $\hat{\mathcal{H}}_{\text{Mass}}$, $\hat{\mathcal{H}}_{\text{Momentum}}$, and $\hat{\mathcal{H}}_{\text{Energy}}$. Similar to the soliton dataset (as shown in Figure 8.5), these three quantities are highly correlated and show similar fluctuations. The errors fluctuate significantly during the first epochs but gradually decrease and stabilize over time while continuing to reduce the error. Compared to the conservation errors observed in the soliton dataset, the errors in the colliding solitary waves dataset are slightly higher, which is expected as simulating the interaction of two waves is considered to be a more challenging task.

Figure 8.12 visualizes some statistics regarding the conservation losses concerning each time step. The most notable result is that all the statistics (mean, median, and standard deviation) take on larger values than the soliton dataset. The mean and median increase steadily over time, and their difference indicate that most of the samples have a lower error, which is more steady over time for both the energy and momentum error. The mass means and median increase evenly together, providing evidence that the generation of each time step generally induces some error.

The energy and momentum errors show an interesting bump in their mean values, which is more noticeable in the standard deviation around time step 17. This time step is around when multiple waves start colliding with each other, and the frequency of collisions increases as the simulation progresses. In order to examine how well the models handle wave overtaking, which is regarded as the most

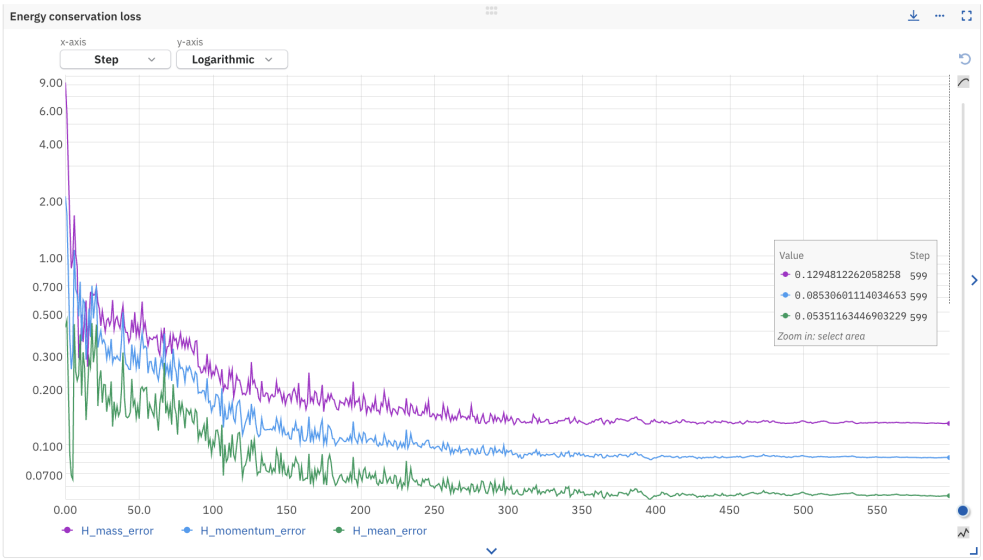


Figure 8.11: During the 600-epoch training period on the colliding solitary waves dataset, all three measures of energy conservation loss decreased and stabilized. The values at the last epoch are reported in Table 8.2. The y-axis is in the logarithmic scale to make small changes more prominent.

Regular GAN: random samples

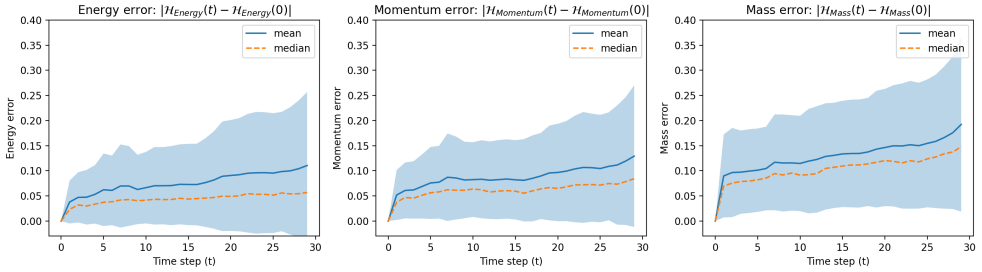


Figure 8.12: Statistics considering the different conserved quantities with respect to every time step of 2000 samples (226 or ≈ 12% of them considered overtaking). The blue-shaded region represents one standard deviation from the mean. The mean and median errors show that the error increases with each time step.

significant moment of nonlinear interaction, we categorize the generated samples into two groups: overtaking and non-overtaking. We group using the overtaking definition presented earlier in section 7.1.2 but summarized as follows: The scenarios where the peaks of the two waves align above each other before the 25th time step are considered to be overtaking samples. The rest is non-overtaking.

Regular GAN: non-overtaking solitons

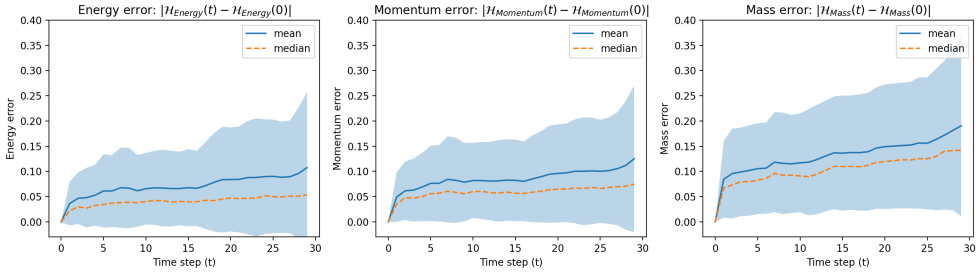


Figure 8.13: Statistics of 2000 generated non-overtaking solitons. The statistics consider the different conserved quantities with respect to every time step. The blue-shaded region represents one standard deviation from the mean. The mean errors over all time steps are as follows: $\hat{\mathcal{H}}_{\text{Energy}} = 0.070$, $\hat{\mathcal{H}}_{\text{Momentum}} = 0.084$, and $\hat{\mathcal{H}}_{\text{Mass}} = 0.1295$.

Comparing the overtaking and non-overtaking statistics displayed in Figures 8.13 and 8.14, it becomes evident that there are significant differences between the two scenarios. The non-overtaking statistics resemble the randomly generated sample statistics shown in Figure 8.12, whereas the overtaking statistics exhibit more dynamic behavior. In particular, the energy error remains small until time step 11, after which it increases rapidly, even though the fastest possible overtaking occurs at time step 14 (when the left soliton is the tallest and the other very short). This suggests that generation flaws happen before the waves collide or their peaks align. On the other hand, the momentum and mass errors decrease.

Regular GAN: overtaking solitons

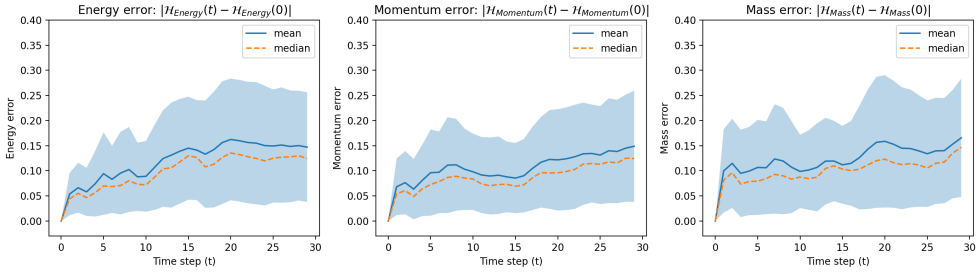


Figure 8.14: Statistics of 2000 generated overtaking solitons. The statistics consider the different conserved quantities with respect to every time step. The blue-shaded region represents one standard deviation from the mean. The mean errors over all time steps are as follows: $\hat{\mathcal{H}}_{\text{Energy}} = 0.1184$, $\hat{\mathcal{H}}_{\text{Momentum}} = 0.1044$, and $\hat{\mathcal{H}}_{\text{Mass}} = 0.1214$.

Comparing the mean energy error between non-overtaking and overtaking scenarios reveals a substantial increase, rising from 0.07 to 0.1184, as reported in Figures 8.13 and 8.14. Although the other conservation losses also increased, their rise was not as significant. The increase in energy conservation error indicates that

the model does not accurately capture the physical behavior of the system and is unable to simulate wave overtaking precisely.

As the number of collisions increases after time step 15, all errors begin to rise. This time period is considered the most significant moment of nonlinear interaction. The fact that the errors increase suggests that the model struggles with the nonlinear interaction of the colliding waves. Further investigation into whether this is due to insufficient generator capacity is discussed in the next section.

The non-overtaking statistics closely resemble the general statistics of randomly sampled data (Figure 8.12) because of an uneven distribution of overtaking and non-overtaking samples. Non-overtaking samples are generated more frequently and constitute most of the sample distribution. In contrast, the overtaking waves, which account for approximately 12% of the generated samples, are not generated often enough to have a noticeable impact on the general sample statistics. Only certain combinations of soliton heights in the initial condition ensure an overtaking scenario. Figure A.13 in the Appendix presents a visual representation of the small percentage of the initial conditions that result in overtaking waves.

8.3 Autoencoder and physics-informed GAN

The visual samples displayed in Figure 8.15 from the physics-informed model are very similar to the regular GAN. Both models produce relatively smooth and visually similar samples that follow similar dynamics. Some samples still exhibit small ripples and soliton peak jumps, e.g. the central right-most column sample in Figure 8.15. However, the conservation errors in Table 8.3 reveal that the model trained with physics information performs much better at conforming to the conservation laws. All conservation errors are considerably lower and in the same range as the autoencoder, which was explicitly trained on the target samples. The autoencoder experienced some difficulties in reconstructing the samples during training as the errors plateaued for several epochs but eventually learned to reconstruct the samples. Visualizations of the mean squared error and conservation errors during autoencoder training are provided in the Appendix as Figure A.17 and A.16. The SW distance appears to be in the same range as the regular GAN. Providing more evidence that it is not sensitive enough to be used as a useful metric in our scenario, as it does not provide reliable information to differentiate between the two clearly different performing models.

Model	SW	$\hat{\mathcal{H}}_{\text{Mass}}$	$\hat{\mathcal{H}}_{\text{Momentum}}$	$\hat{\mathcal{H}}_{\text{Energy}}$
Autoencoder		.0361	.0174	.0125
Physics-informed GAN	4.41e-5	.0352	.0251	.0168

Table 8.3: The results of the autoencoder trained to decode and deconstruct target samples and the GAN trained while being informed about its conservation errors. Both obtained very similar results.

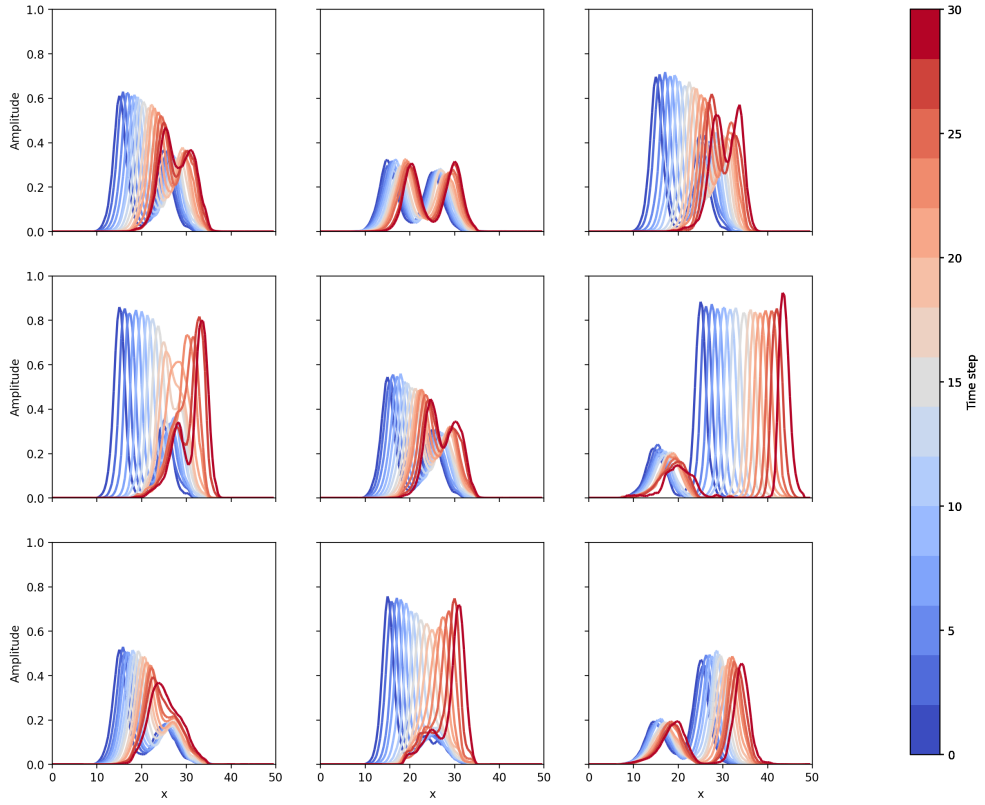


Figure 8.15: A random selection of nine colliding solitary wave samples that were randomly generated from the physics-informed model. Each time step of the system is represented in a different color, ranging from the initial state (blue) to the final state (red). Only every second time step is displayed to visualize wave interactions more clearly.

Due to the sample being visually similar to the regular GAN, visualization of the generated samples using PCA, t-SNE, and UMAP is considered redundant for this model's performance. The same holds for the distribution of soliton heights in the initial condition. These results are instead available in Figure A.14. in the Appendix.

The conservation regularization was initiated at epoch 160, which is also clearly visible in the error drop in Figure 8.16. The error quickly drops and stabilizes, slowly decreasing the conservation errors until convergence. The behavior of the conservation loss is similar to the regular GAN but decreases for a longer period before converging. This behavior is logical, considering that the model receives explicit information about the conservation errors.

Inspecting the conservation error in the physics-informed model for both overtaking and non-overtaking samples shows the two scenarios behave very differently While the non-overtaking samples have a slow and stable error increase over time, the overtaking statistics exhibit more dynamic behaviors. All conservation

errors visualized in non-overtaking consist of a much smaller standard deviation that gradually increases over time but indicates that most non-colliding waves have been seemingly accurately generated. The overtaking samples displayed in Figure 8.18 share a similar trend but is less significant in the mass and momentum errors.

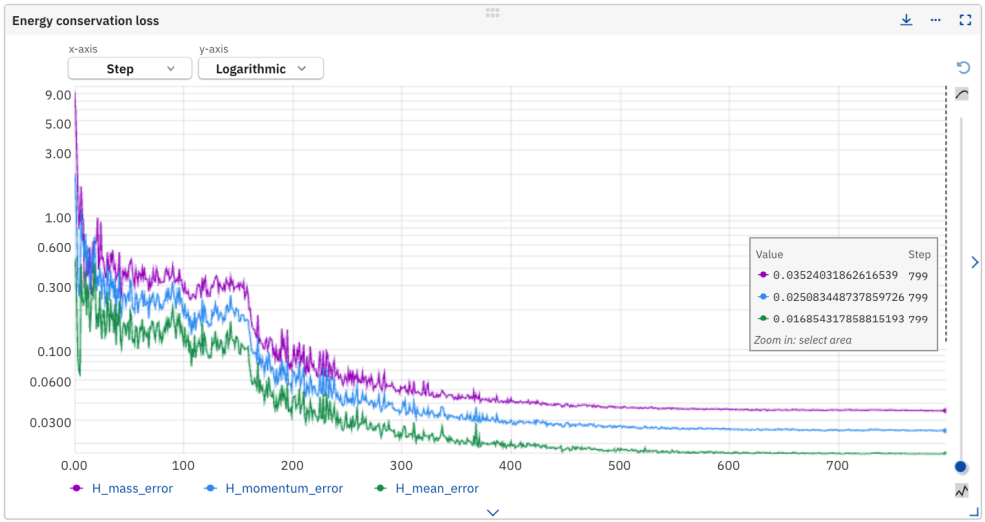


Figure 8.16: During the 800-epoch training period on the colliding solitary waves dataset with conservation regularization, all three measures of energy conservation loss decreased and stabilized. The values at the last epoch are reported in Table 8.3. The y-axis is in the logarithmic scale to make small changes more prominent.

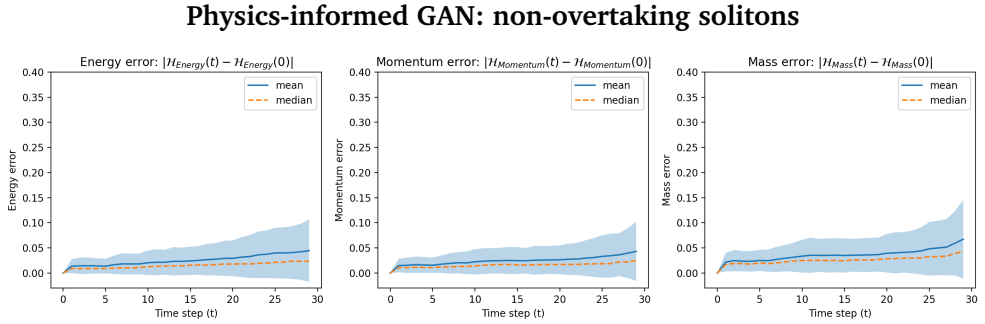


Figure 8.17: Statistics of 2000 generated non-overtaking solitons from the physics-informed model. The statistics consider the different conserved quantities with respect to every time step. The blue-shaded region represents one standard deviation from the mean. The same y-axis range as Figure 8.14 is used for easy comparison.

The energy error bump revealed in Figure 8.18 indicates that the GAN generator is insufficient in learning the dynamics of the colliding waves, even when being physics-informed. The mean error seen at time step 15 in the energy error

equals 0.15, identical to the error seen in the model trained without conservation regularization in Figure 8.14. The mean and median errors are also similar, indicating that the average sample experiences the same energy error at the time of the collision. These findings suggest that some limitation hinders the model from improving beyond this barrier and is likely due to insufficient network capacity.

The energy bump's dynamics in Figure 8.18 provides valuable insights into the colliding wave simulation. Initially, up to time-step 10, the model performs well in simulating the two colliding waves. However, as the waves move apart after the collision, starting from time step 11 and beyond, the model struggles to represent their behavior accurately. Since the system is highly nonlinear, the energy exchange and position of the waves during their collision are not intuitive. It requires an in-depth understanding of the system to model correctly, which the GAN model fails to capture fully.

Physics-informed GAN: overtaking solitons

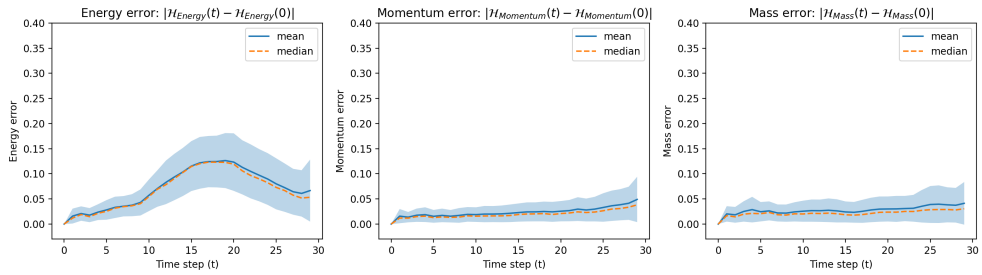


Figure 8.18: Statistics of 2000 generated overtaking solitons from the physics-informed model. The statistics consider the different conserved quantities with respect to every time step. The blue-shaded region represents one standard deviation from the mean. The same y-axis range as Figure 8.14 is used for easy comparison.

By measuring the conservation errors of the reconstructed samples, we can evaluate the autoencoder's ability to reconstruct the target samples. The findings demonstrate that the generator network can not reconstruct the samples perfectly. Figure 8.19 displays the conservation errors of reconstructed overtaking samples, which closely resemble the statistics from the physics-informed model in Figure 8.18. Even though the autoencoder was trained using target data, one would reasonably expect to see a substantially lower energy error, but this did not occur. The energy error plots of both models are nearly identical, revealing a shared error bump. This similarity suggests that the generator network has been trained to its potential limitations in both cases. Since the generator is solely responsible for generating the samples, it must contain all the necessary knowledge to create them. However, the observed energy errors indicate that the generator network lacks the ability to construct samples with greater accuracy and lower error.

The autoencoder and the physics-informed GAN achieved more than three times less conservation error for all conservation quantities than the regular GAN. These low conservation errors also revealed the limitations of the generator net-

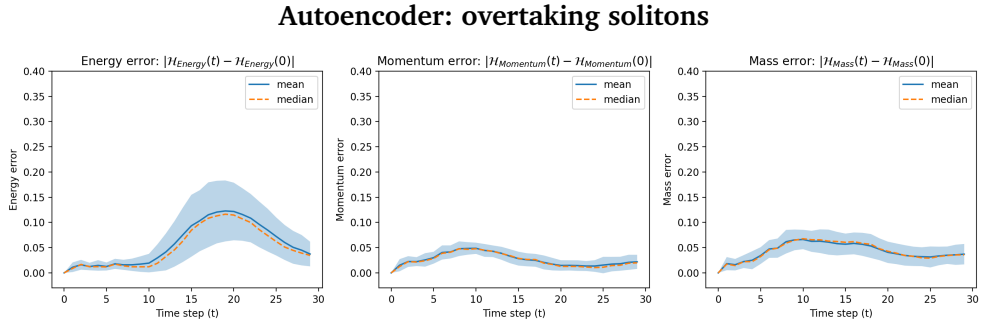


Figure 8.19: Statistics of 453 reconstructed overtaking solitons using the autoencoder. The statistics consider the different conserved quantities with respect to every time step. The blue-shaded region represents one standard deviation from the mean. The same y-axis range as Figure 8.18 is used for easy comparison.

work architecture, as both the autoencoder and physics-informed GAN obtained very similar sample statistics. Both models encountered a similar obstacle that prevented further improvements and given that they shared the same generator architecture, the generator capacity is considered insufficient. The aim of both models was to assess the generator's capacity and establish a benchmark for the GAN without conservation error knowledge to compare against, which they successfully achieved.

8.4 Model comparisons

Here we combine the results into a discussion about the relative performance differences. The main results considering each model about their conservation errors are listed together in Figure 8.20 for easy comparison.

There exist many differences to discuss, so the section is divided into two main parts: Sample complexity and Physics information, where the former compares the two different datasets used with additional comparisons of the overtaking scenarios, while the latter will discuss what performance difference between the regular and physics-informed GAN.

8.4.1 Sample complexity

Both models demonstrate they can produce many moderately realistic samples, as some samples included small visual flaws. The commonly used evaluation methods, such as sample comparison using PCA, t-SNE, SW distance, and mode drop investigation, all show excellent performance for both models. Since both models show similar performance, it is problematic to understand how the sample complexity affected the model performance or even to know if the samples are correctly modeled. None of these evaluation methods can properly assess their sample quality differences. Using more accurate measurements, such as a more

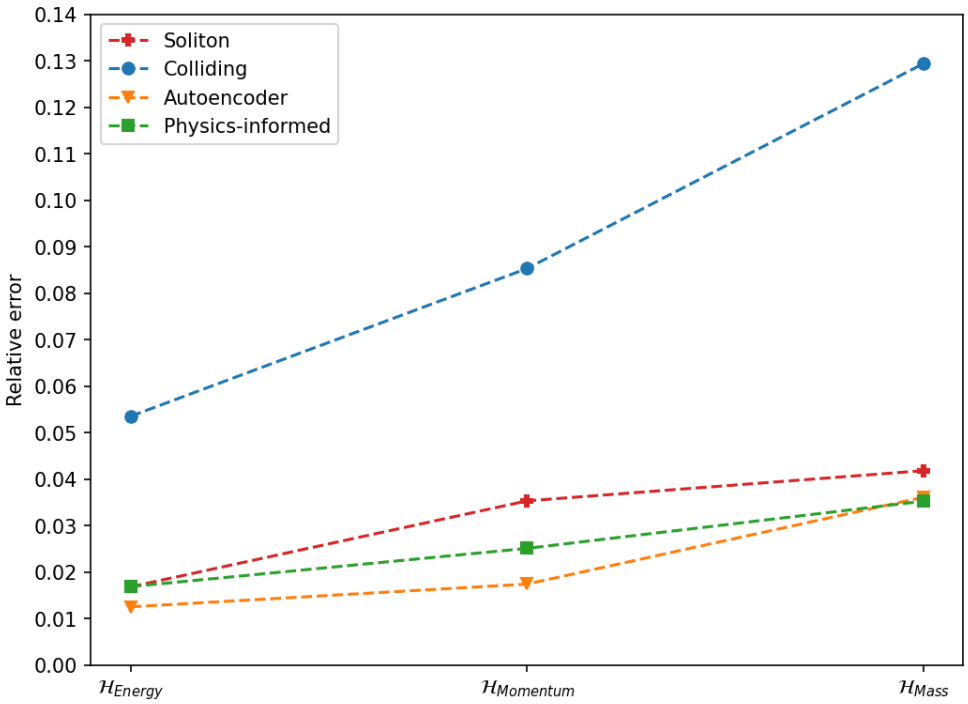


Figure 8.20: The combined plot of all experimental models introduced above and their respective conservation losses. Here Solion and Colliding denote the models trained in the soliton and colliding solitary waves dataset, respectively.

accurate Wasserstein distance estimation, would provide more insight, but not as clearly as measuring data-specific properties such as conservation laws. The error associated with conserved quantities (mass, momentum, and energy) all reveal considerable differences between the two datasets (visualized in Figure 8.20). The model trained to generate simple solitons follows the conservation quantities much more closely, while the colliding solitons violate them more. This is clearly visible from the statistics of the overtaking samples in Figure 8.14) and also shows that there is much space for improvement. The error difference in the conserved quantities also matches our expected model performances, as the collision of two waves should be much more difficult to model.

The total energy associated with the systems significantly differs between the two datasets. A system containing two waves may have up to almost three times as much energy as a single-wave system, which can influence the results. Figure 8.21 reveals a correlation in which a system's general energy conservation error increases with respect to its total energy. This correlation is also present in the soliton data but not less significant (visualized in Figure A.10 in the Appendix). The conservation errors for the model generating two waves may therefore have some error explained by the samples themself holding more energy.

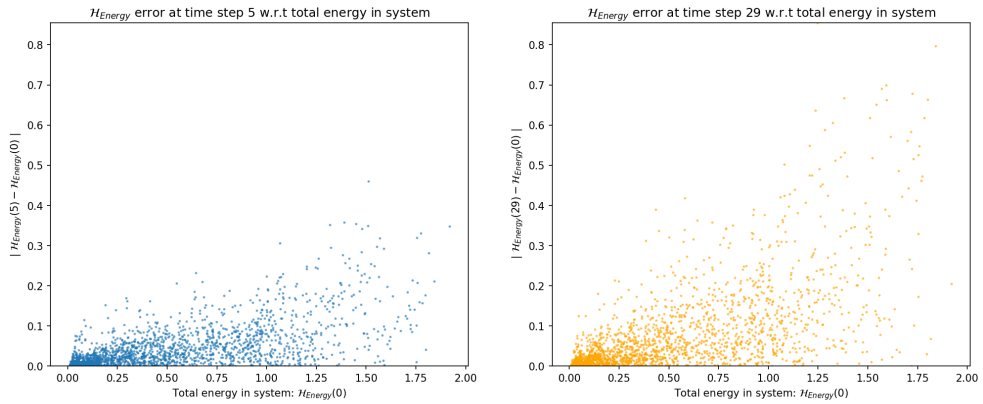


Figure 8.21: Scatter plot illustrating the relationship between the energy error ($\mathcal{H}_{\text{Energy}}$) of 2000 generated colliding solitons and the total energy in the system at two different time steps, time step 5 and the final time step (29). At the final stages of the simulation (right plot), the error is generally larger, but also for systems containing consisting of more energy. However, there are still many samples with low error, similar to the early phases.

Overtaking vs non-overtaking samples

Although the model trained on colliding waves performed worse than solitons, it exceeded expectations in certain parts of the simulation. The simulation of two waves shows similar properties to the evolution of a single wave, such as the shape of the wave and wave speed, which depend on the wave height. Both models share that the shapes of the waves appear realistic and move at the correct speeds. This holds for the colliding wave samples until the collision, as evidenced by the statistics of the non-overtaking waves (Figure 8.13), which remain relatively stable until they come into contact and interact.

Considering the modeling of waves that overtake each other, the model performs much worse. This is evident in the statistics on overtaking samples (Figure 8.14), as the errors are much more dynamic with respect to time. The large error bumps arise from difficulties with the wave's interaction and separation again after overtaking has occurred.

Testing the generator network capacity using the autoencoder reveals that there is still much potential for the GAN to perform better. It is not certain how much better performance is achievable, but some other combination of hyperparameters definitely exists to obtain slightly better results. Envisioning a more extensive performance gain would require modifications to the architecture. This is believed to be the case because the autoencoder results also revealed that the generator capacity is insufficient in modeling the colliding waves accurately. This was confirmed by neither the autoencoder nor the physics-informed model being able to model the colliding waves accurately as they obtained considerable energy conservation errors (by comparing Figure 8.18 and 8.19).

The generator architecture is also considered quite simple, comprising just

two GRU layers and three fully-connected layers. Since the colliding waves include several highly nonlinear interactions, the generator must replicate the same nonlinearities. As the autoencoder could not do so, the generator is believed to comprise too few nonlinear activation layers. Increasing the generator capacity would seem to fix this issue. Still, we have tried to address this problem by including extra GRU layers, more fully-connected layers, and nodes, but without achieving better results. Something about the COTGAN training procedure is also suspected to affect the model results, but we have yet to look thoroughly into this.

Another factor that could have increased the overtaking conservation error is the in-homogeneous data distribution; only approximately 12% of the target samples are considered good overtaking samples (Figure A.13). This challenges the model in learning their difficult interaction when only a few samples are considered when performing network updates, rendering its gradient update ineffective. It is not considered one of the leading causes, but something that should be investigated further.

8.4.2 Physics information

By comparing the conservation errors of the physics-informed GAN and regular GAN in Figure 8.20, it is evident that the physics-informed model adheres much more closely to the conservation laws. The conservation errors obtained by the physics-informed GAN are over three times lower and near the autoencoder, a model trained explicitly on target data. As both the autoencoder and physics-informed GAN yield similar results and sample statistics, it is believed that they have encountered the same performance barrier due to restricted generator capacity.

Both visually and using the commonly used evaluation methods, it isn't easy to accurately state the performance differences between the physics-informed and regular GAN. It is possible to see some visual improvements since the data includes smooth curves, but that is it. Knowing that the physics-informed model generates samples much more accurately, but where the improvements are not visible or easily observable, is thus a surprising result. For cases where more complex data is considered and not smooth curves, e.g., medical data, this is a huge problem. If it is not possible to be certain that the underlying constraints in the data are learned and respected, generated samples could be deemed misleading and provide false statistics. This proves the importance of proper evaluation metrics and how knowing about underlying constraints can help evaluate and enhance performance.

However, knowing or being able to define underlying constraints is very often not possible, e.g., working with image data. In these cases, one can only assume the model has learned the underlying constraints. Using the conservation errors, we also found that our model had insufficient capacity to model the scenarios perfectly, which would have gone unnoticed without the conservation knowledge. This also illustrates the importance of data knowledge and the uncertainty in model performance.

The potential performance of the regular GAN and what it fails to capture

The regular GAN's lack of consideration for underlying physics is apparent when comparing statistics of the conservation errors of the overtaking solitons, as shown in Figures 8.14 and 8.18. The physics-informed model accurately accounts for the mass and momentum conservation with significantly lower and more stable errors. In contrast, the regular GAN has at least twice the error, which fluctuates considerably throughout the simulation. While all models struggle with energy conservation, the physics-informed model and autoencoder recover somewhat after the most nonlinear interactions, as the conservation error decreases after most collisions finish. Conversely, the regular GAN accumulates errors without noticeable recovery, indicating that it does not understand the wave interactions.

However, it should be noted that even if the conservation error falls back to zero after the collision, a generated sample may not be entirely accurate. The new wave positions after the collision are non-trivial and do not affect the conservation checks. Therefore, adding conservation laws does not make the GAN water-tight against sample errors. As this is specific to this particular wave equation and not a general physical law, we have not evaluated whether the waves are correctly positioned during the entire simulation.

An important point to note regarding the non-informed GAN result is that we chose the model based on its performance on the SW distance and conservation errors. This means many other versions of the model with different architectures and hyperparameters may perform similarly to the one presented here when only utilizing standard evaluation metrics. The analyzed results are hence subject to selection bias and do not necessarily reflect the typical or expected performance of GAN models but rather the best model that could adhere to the conservation laws with minimal error through an implicit optimization process. Therefore, all of the reported values of the conservation errors are generally worse when evaluating the regular GAN's performance without access to the conservation errors. Figure A.11 contains the results of many worse-performing models encountered in the hyperparameter selection process. This more strongly signifies that GANs are not able to learn the underlying laws in the data correctly.

8.5 Other notable points

GAN training difficulties

One important point to note regarding the model results is that it took a significant time to find suitable architectures and hyperparameters to achieve desirable outcomes. A total of 553 hours ($\sim$23 days) was spent only training the different GANs. Although COTGAN was one of the easier models to train without encountering issues such as total mode collapse or non-convergence, it still required careful tuning. One of the major challenges was dealing with local minima in generator network capacity. Several attempts were made to increase the generator capacity,

but they resulted in poorer results. Experiments involved testing different generator architectures using the autoencoder for quick performance testing but were also unsuccessful. It was assumed that a more significant capacity generator would yield higher-quality samples, but all experiments in this direction failed. Therefore, we believe the architecture needs to be significantly larger or redesigned using different modules, such as convolutions. This is discussed further in future work.

Higher spatial and temporal resolution

We explored the feasibility of achieving high spatial and temporal resolutions in our simulations without encountering too severe training issues. We experimented with spatial resolutions of up to 180 and 60 time steps, but we found that these resolutions led to unrealistic-looking samples. Therefore, we decided on the lower resolution, which was easier to work with. Although higher spatial resolution would have resulted in smoother simulations, it would have significantly extended the hyperparameter search period without considerably affecting the findings. Higher resolutions would, however, allow us to perform a more in-depth sample analysis of the waves during the collision and modify the dataset to enable the waves to travel for an extended period. Investigating where the GAN fails in greater depth may be viewed as a topic for future research while also creating strategies to address the challenge of generating high-dimensional time series.

Approximate Sliced-Wasserstein distance as an evaluation method

Every generative model faces the challenge of lacking proper evaluation metrics to easily and reproducibly assess the results. Many evaluation methods use other neural networks, are non-deterministic, or rely solely on visual inspection. We applied a new metric that does not rely on these flaws, the fast approximated Sliced-Wasserstein distance and used it to monitor and evaluate our results. While it was effective in tracking the training process, it wasn't as precise when the generated samples closely resembled the target samples. The method had high variance even when using 2000 samples, making it inadequate for accurately assessing model performance. Figure A.9 in the Appendix shows the high variance obtained from different sample sets drawn from the same distribution of solitons. Increasing the sample size could address this issue, but using a more accurate Wasserstein distance estimation method like Sinkhorn divergence (section 2.3.2) or non-deterministic Sliced-Wasserstein (section 2.3.3) would provide higher accuracy for post-evaluation.

Additional indications of inaccuracies in the approximated Sliced-Wasserstein distance are presented in Figure 8.22. The figure demonstrates that, despite exhibiting similar SW performance, the three models behave very differently regarding energy conservation. While expecting the SW distance to detect these differences in conservation errors precisely may be unreasonable, more distinction in the SW

distance between the models would be beneficial. When more samples are available, more samples can be considered to estimate the SW distance more accurately. However, in cases where only limited samples are available, better techniques must be utilized. This thesis demonstrated that utilizing data knowledge, such as conservation laws, is one powerful technique that can lead to better model evaluation without requiring numerous samples.

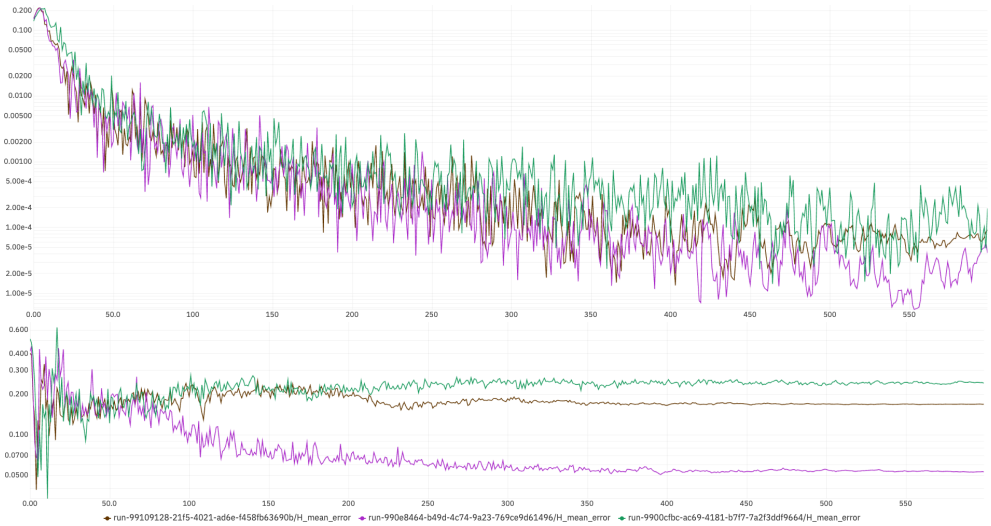


Figure 8.22: The monitored SW distance and energy conservation error of three non-informed models with slight hyperparameter differences. The top plot displays the SW distance, and the bottom plot shows the energy conservation error, with each model being represented by a different color - green, brown, and purple. The monitored SW distance shows similar values for all three models, whereas the energy conservation error varies greatly. This indicates that the SW distance is not sensitive enough to differentiate the models based on their loyalty to the conservation law.

Chapter 9

Conclusion and Future Work

This chapter provides a comprehensive summary of the results obtained and addresses the research questions posed in this thesis. Additionally, it offers insights into potential avenues for future research and expansion of the current work.

9.1 Conclusion

The problem of not knowing if GANs adhere to the same underlying laws as the real data is a significant bottleneck for interpretation. Data that does not obey the same laws may exhibit different statistics and be considered invalid for many practical applications. While some applications can inform the GAN about underlying constraints due to domain knowledge, this is not the case for most applications, as the underlying process might be too complicated, inaccessible, or unknown. In these situations, understanding if GANs can adhere to the underlying constraints is of great importance to improve awareness about their limitations and help aid the selection of reliable models for critical applications.

In this thesis, we investigated the capability of GANs to capture the implicit laws embedded in a physical system. Our experiments focused on a simulated environment of nonlinear propagating ocean waves described by the well-known KdV equation. Simulations following the KdV equation adhere to conservation laws, which are also mathematically defined and enable measuring the flaws in the generated samples. We demonstrate the capability of GANs by training them to produce new simulations on a dataset of examples and measure their adherence to the underlying laws.

Before conducting the main experiment, we tested four different GAN frameworks, namely RGAN, TimeGAN, COTGAN, and RTSGAN, on a simple dataset of sinusoidal waves. By assessing sample quality, diversity, and training behavior, we identified COTGAN as the best-performing model and selected it as the base model for the main experiments.

In addition to conducting various GAN experiments, we utilized a novel evaluation metric, the Fast approximation of the Sliced-Wasserstein distance (SW) proposed by Nadjahi [26], to monitor and evaluate the performance of the model—this

effort aimed to contribute to the ongoing exploration of effective evaluation metrics for generative models.

For the main experiment, we utilized the conservation errors and SW distance to assess the performance of different GANs, specifically a regular and physics-informed GAN. Considering the performance of the physics-informed GAN as the benchmark for comparing performances, we report statistical differences between the conservation errors associated with mass, momentum, and energy between the two models.

RQ1: How well do regular GANs conform to the underlying physical laws governing the data in comparison to physics-informed GANs?

The results exhibit remarkable similarities when comparing the regular GAN with the physics-informed GAN using standard evaluation methods like PCA, t-SNE, UMAP, and SW distance. Both models generate relatively smooth waves that visually appear to behave correctly. However, a more detailed analysis of the mass, momentum, and energy conservation error reveals that the physics-informed GAN captures the underlying laws with considerably greater adherence. The conservation error between the models is over three times higher for the regular GAN, which reflects the best-case scenario.

It is important to note that the reported result for the regular GAN is biased as it was implicitly optimized by observing the conservation errors during the tuning process. Consequently, these results do not reflect the performance one would expect from a regular GAN trained without access to the conservation errors. This observation highlights that an unbiased GAN would adhere less to the underlying physical laws. Given our findings, a regular GAN can not be expected to consistently conform to the underlying conservation laws in general settings, as our results reflect the best possible GAN scenario.

With its simplicity and smoothness, the simulation governed by the KdV equation is relatively simple compared to other real-world data. Real-world data often includes noise, higher-dimensional data, and more complex nonlinearities, which may not have the same visually appealing representation. In such scenarios, the performance of GANs is expected to be even less impressive, as the mediocre results observed here provide little evidence of adequate adherence to the underlying constraints.

In conclusion, regular GANs may exhibit inconsistency in conforming to the underlying physical laws that govern the data. Despite producing visually pleasing results, regular GANs tend to have significant conservation errors and face challenges in accurately adhering to the underlying constraints of complex systems.

RQ2: What are the limitations and challenges GANs face in adhering to complex physical laws in multivariate time series data?

Several challenges limit GANs in adhering correctly to the underlying laws, but the primary reason is believed to be that they are purely statistical. Only relying

on the statistical properties of the training data in combination with its stochastic generation yields a problematic task, and gradients used for updating the network do not provide sufficient information. Models converged to results that do not adequately adhere to conservation laws, as the delicate details in the training data are considered too tiny to capture through statistical learning.

A more direct challenge that significantly affects GAN's capability to reach results that adhere to the underlying laws is its unpredictable tuning process. Its sensitivity to small hyperparameters and architecture changes dramatically slows the convergence rate toward optimal design and hyperparameters. Despite monitoring the conservation errors and making reasonable modifications, GANs unpredictably generate poor results, even when utilizing explicit algorithms such as the Sinkhorn algorithm to provide accurate gradients for the generator.

The problem is hence not a direct cause of poor evaluation metrics but how GANs behave in the first place. Even using perfect evaluation metrics, such as the conservation errors used in our experiments, does not fix the issue of getting the model to optimal results without explicitly informing the model about them. Obtaining optimal GANs with sufficient architecture and good hyperparameters in applications without access to the underlying constraints is thus even less likely. Investigating what causes this unpredictable tuning behavior would be a case for future work.

Secondary challenges exist regardless if their unpredictable tuning nature is resolved, such as the lack of a suitable evaluation metric sensitive enough to detect violations of conservation laws, the time-consuming nature of hyperparameter tuning, and the uncertainty regarding the generator's capacity.

The lack of proper evaluation metrics makes it difficult to distinguish between versions of the model that perform similarly but conform differently to the underlying laws. Visual inspection and conventional statistical techniques, such as PCA, t-SNE, and UMAP, are insufficient for assessing these differences, necessitating the use and development of more sensitive evaluation methods.

Tuning GANs requires significant time and computational resources. The training duration scales with the sequence length and dimensionality, and applying GANs to complex datasets and practical problems increases the time spent on tuning. Employing hyperparameter search techniques, such as Bayesian optimization, becomes infeasible due to the long training periods and the lack of suitable evaluation metrics to guide the search. Finding optimal parameters where the GAN adheres to the conservation laws within a reasonable margin may be out of scope for some applications.

Determining whether the network has sufficient capacity to generate accurate samples is a significant problem. Insufficient network capacity may go unnoticed during the training process, and the lack of capacity can only be identified through the measurement of conservation errors. This implies that the GAN may never produce correct samples, raising concerns when assessing network capacity in applications where constraint error can not be directly measured.

The issue of sufficient capacity also affects other deep learning models, as evid-

enced by the autoencoder. The autoencoder is capable of reconstructing samples that seem precise and realistic without showing any evidence of energy conservation violations. However, this observation does not hold when directly measuring the constraints. This finding suggests that even when the model is trained with perfect knowledge, it may not fully obey physical laws. Therefore, assuming that GANs, which implicitly learn the target distribution, achieve better results is unreasonable.

RQ3: How effective is the approximated Sliced-Wasserstein distance in evaluating the performance of GANs on time series?

The Fast approximation of the Sliced-Wasserstein distance (SW) has demonstrated effectiveness for different purposes. Although it is not specifically tailored to account for the causal nature of time series data, it proves to be a valuable evaluation metric for the sinusoidal dataset. It effectively establishes a distinct and meaningful boundary between different model performances, and its computational efficiency makes it suitable for monitoring GAN training progress.

However, the metric lacks the sensitivity to differentiate model performance in the main experiment effectively. When generating colliding solitary waves, the models quickly converge to produce samples similar to the target distribution, rendering the SW distance too inaccurate for assessing their performance differences.

Since the metric is an approximation of the Sliced-Wasserstein distance, it is expected to have a sensitivity threshold, but determining the point at which the approximation becomes too inaccurate is difficult to reckon beforehand. The data dimensions and the number of samples considered also influence this threshold. Therefore, when working with small sample sets, it is advisable to have less reliance on the metric.

The sensitivity threshold is also affected by the dimensions of the data and the number of samples used for the measurement. Hence, when dealing with limited sample sets, placing less reliance on the metric is recommended.

In conclusion, the Fast approximation of the Sliced-Wasserstein distance is highly valuable for monitoring GAN training in the early stages, but it is not suitable for assessing performance differences between similarly performing models or when only small sample sets are available.

9.2 Future work

Given that the primary objective of this study was to demonstrate the limitations of GANs in capturing the underlying physical laws present in the data, it is important to note that our findings do not conclusively prove their inability. Therefore, it is essential to delve deeper into the assumptions and choices made within this thesis, conducting further investigations to gather additional evidence supporting either scenario. It is important to perform comprehensive testing with diverse

configurations and data types, as this will contribute to a more comprehensive understanding of the capabilities and limitations of GANs. Expanding the scope of our analysis to encompass a broader range of settings and modules presents an important road for future research. Some natural ideas for extending our analysis include examining the performance of higher capacity networks, exploring the use of different recurrent modules, and extending the dataset to higher spatiotemporal resolutions.

Enhancing Network Capacity

The findings of this thesis indicate that the model's ability to generate samples adhering to the laws of physics was limited, raising the question of whether insufficient generator capacity played a significant role in this limitation. Despite our efforts, we were unable to increase the generator's capacity to the desired level without experiencing significantly poorer training results. It is intuitive to believe that a higher-capacity generator, equipped with more nonlinear operations, would better capture the simulated system than a lower-capacity model. Therefore, future research focused on acquiring a Generative Adveserial Network with sufficient capacity and performing similar experiments would yield important results. Such work would explore whether GANs encounter difficulties in learning good representations when strict capacity constraints are in place and potentially reveal that our GAN performed poorly due to convergence to a local optimum. Understanding whether insufficient capacity is a major barrier significantly affecting adherence to underlying relationships would be important.

Given the challenges encountered in augmenting the capacity of the current architectural design, a redesign becomes necessary. Alternative recurrent modules and design methodologies should be considered. Noteworthy starting points for exploration include the progressive network scaling method proposed by Karras *et al.* [96] and the utilization of weight normalization to stabilize training, as investigated by Xiang and Li [97].

Exploring alternative recurrent modules

Throughout this thesis, we have mainly utilized one specific recurrent module, namely the Gated Recurrent Unit. It's popularity and computational efficiency primarily drove this choice compared to other modules like LSTMs. Since the recurrent module plays a crucial role in establishing connections between time steps and predicting the next step in the sequence within our architecture, it would be worthwhile to investigate different recurrent modules. Modules such as LSTM, dilated (causal) convolutions, or transformers offer distinct behaviors and may offer additional capabilities and other outcomes. Dilated convolutions, designed to respect causality in convolutions, have already been successfully utilized in GANs for speech enhancement [98], and transformers have been applied to time series data as well [78, 86].

Conducting similar experiments to those in this thesis will help determine if the choice of recurrent modules in the GAN architecture is a limiting factor in achieving better results or if other aspects of the framework need improvement.

Utilizing higher spatiotemporal resolution

Expanding the simulation to a higher spatiotemporal resolution would enable a closer examination of how the GAN understands wave interactions, providing more insights into the underlying issues in the generation. Although we briefly considered increasing the spatial resolution in this thesis, we ultimately abandoned it due to the increased complexity of tuning and training. However, there are existing approaches to address challenges with long sequences [99] and high dimensionality [96]. While investigating higher-resolution data could be interesting and valuable for modeling long sequences, it is unlikely to yield fundamentally different results from those presented here. Nevertheless, it would contribute to a better understanding of the underlying problem. Advancements in this area would benefit applications where strict adherence to the underlying constraints is less critical.

Bibliography

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Networks*, Number: arXiv:1406.2661 arXiv:1406.2661 [cs, stat], Jun. 2014. [Online]. Available: http://arxiv.org/abs/1406.2661 (visited on 25/05/2022).

[2] J. Ho, X. Chen, A. Srinivas, Y. Duan and P. Abbeel, 'Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design,' en, in *Proceedings of the 36th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, May 2019, pp. 2722–2730. [Online]. Available: https://proceedings.mlr.press/v97/ho19a.html (visited on 24/05/2023).

[3] D. P. Kingma and M. Welling, *Auto-Encoding Variational Bayes*, arXiv:1312.6114 [cs, stat], Dec. 2022. DOI: 10.48550/arXiv.1312.6114. [Online]. Available: http://arxiv.org/abs/1312.6114 (visited on 24/05/2023).

[4] Y. Qiu, Z. Niu, B. Song, T. Ma, A. Al-Dhelaan and M. Al-Dhelaan, 'A Novel Generative Model for Face Privacy Protection in Video Surveillance with Utility Maintenance,' en, *Applied Sciences*, vol. 12, no. 14, p. 6962, Jan. 2022, Number: 14 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: 10.3390/app12146962. [Online]. Available: https://www.mdpi.com/2076-3417/12/14/6962 (visited on 05/06/2023).

[5] Y. Qu, S. Yu, W. Zhou and Y. Tian, 'GAN-Driven Personalized Spatial-Temporal Private Data Sharing in Cyber-Physical Social Systems,' *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2576–2586, Oct. 2020, Conference Name: IEEE Transactions on Network Science and Engineering, ISSN: 2327-4697. DOI: 10.1109/TNSE.2020.3001061.

[6] B. Xin, W. Yang, Y. Geng, S. Chen, S. Wang and L. Huang, 'Private FL-GAN: Differential Privacy Synthetic Data Generation Based on Federated Learning,' in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, ISSN: 2379-190X, May 2020, pp. 2927–2931. DOI: 10.1109/ICASSP40776.2020.9054559.

[7] W. Sirichotedumrong and H. Kiya, 'A GAN-Based Image Transformation Scheme for Privacy-Preserving Deep Neural Networks,' in *2020 28th European Signal Processing Conference (EUSIPCO)*, ISSN: 2076-1465, Jan. 2021, pp. 745–749. DOI: 10.23919/Eusipco47968.2020.9287532.

[8] E. Choi, S. Biswal, B. Malin, J. Duke, W. F. Stewart and J. Sun, *Generating Multi-label Discrete Patient Records using Generative Adversarial Networks*, arXiv:1703.06490 [cs], Jan. 2018. DOI: 10.48550/arXiv.1703.06490. [Online]. Available: http://arxiv.org/abs/1703.06490 (visited on 31/05/2023).

[9] R. Sharma, A. Guleria and R. Singla, 'An overview of flow-based anomaly detection,' *International Journal of Communication Networks and Distributed Systems*, vol. 21, no. 2, pp. 220–240, Jan. 2018, Publisher: Inderscience Publishers, ISSN: 1754-3916. DOI: 10.1504/IJCNDS.2018.094221. [Online]. Available: https://www.inderscienceonline.com/doi/abs/10.1504/IJCNDS.2018.094221 (visited on 05/06/2023).

[10] X. Xia, X. Pan, N. Li, X. He, L. Ma, X. Zhang and N. Ding, 'GAN-based anomaly detection: A review,' en, *Neurocomputing*, vol. 493, pp. 497–535, Jul. 2022, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2021.12.093. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231221019482 (visited on 31/05/2023).

[11] D. Li, D. Chen, B. Jin, L. Shi, J. Goh and S.-K. Ng, 'MAD-GAN: Multivariate Anomaly Detection for Time Series Data with Generative Adversarial Networks,' en, in *Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series*, I. V. Tetko, V. Kůrková, P. Karpov and F. Theis, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2019, pp. 703–716, ISBN: 978-3-030-30490-4. DOI: 10.1007/978-3-030-30490-4_56.

[12] W. Jiang, Y. Hong, B. Zhou, X. He and C. Cheng, 'A GAN-Based Anomaly Detection Approach for Imbalanced Industrial Time Series,' *IEEE Access*, vol. 7, pp. 143 608–143 619, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2944689.

[13] V. Böhm, F. Lanusse and U. Seljak, *Uncertainty Quantification with Generative Models*, arXiv:1910.10046 [astro-ph, stat], Oct. 2019. [Online]. Available: http://arxiv.org/abs/1910.10046 (visited on 05/06/2023).

[14] A. Daw, M. Maruf and A. Karpatne, 'PID-GAN: A GAN Framework based on a Physics-informed Discriminator for Uncertainty Quantification with Physics,' en, in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, Virtual Event Singapore: ACM, Aug. 2021, pp. 237–247, ISBN: 978-1-4503-8332-5. DOI: 10.1145/3447548.3467449. [Online]. Available: https://dl.acm.org/doi/10.1145/3447548.3467449 (visited on 31/05/2023).

[15] V. Edupuganti, M. Mardani, S. Vasanawala and J. Pauly, 'Uncertainty Quantification in Deep MRI Reconstruction,' *IEEE Transactions on Medical Imaging*, vol. 40, no. 1, pp. 239–250, Jan. 2021, Conference Name: IEEE Transactions on Medical Imaging, ISSN: 1558-254X. DOI: 10.1109/TMI.2020.3025065.

[16] V. L. S. Silva, C. E. Heaney and C. C. Pain, *A GAN-based Reduced Order Model for Prediction, Data Assimilation and Uncertainty Quantification*, arXiv:2105.13859 [cs, stat], Oct. 2022. DOI: 10.48550/arXiv.2105.13859. [Online]. Available: http://arxiv.org/abs/2105.13859 (visited on 31/05/2023).

[17] D. Kiyasseh, G. A. Tadesse, L. N. T. Nhan, L. Van Tan, L. Thwaites, T. Zhu and D. Clifton, 'PlethAugment: GAN-Based PPG Augmentation for Medical Diagnosis in Low-Resource Settings,' *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 11, pp. 3226–3235, Nov. 2020, Conference Name: IEEE Journal of Biomedical and Health Informatics, ISSN: 2168-2208. DOI: 10.1109/JBHI.2020.2979608.

[18] C. Han, H. Hayashi, L. Rundo, R. Araki, W. Shimoda, S. Muramatsu, Y. Furukawa, G. Mauri and H. Nakayama, 'GAN-based synthetic brain MR image generation,' in *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, ISSN: 1945-8452, Apr. 2018, pp. 734–738. DOI: 10.1109/ISBI.2018.8363678.

[19] P. Shende, M. Pawar and S. Kakde, 'A Brief Review on: MRI Images Reconstruction using GAN,' in *2019 International Conference on Communication and Signal Processing (ICCSP)*, Apr. 2019, pp. 0139–0142. DOI: 10.1109/ICCSP.2019.8698083.

[20] K. G. Hartmann, R. T. Schirrmeister and T. Ball, *EEG-GAN: Generative adversarial networks for electroencephalograhic (EEG) brain signals*, arXiv:1806.01875 [cs, eess, q-bio, stat], Jun. 2018. [Online]. Available: http://arxiv.org/abs/1806.01875 (visited on 28/09/2022).

[21] J. Fu, S. Li, Y. Jiang, K.-Y. Lin, C. Qian, C. C. Loy, W. Wu and Z. Liu, *StyleGAN-Human: A Data-Centric Odyssey of Human Generation*, arXiv:2204.11823 [cs], Apr. 2022. [Online]. Available: http://arxiv.org/abs/2204.11823 (visited on 16/05/2023).

[22] X. Pan, A. Tewari, T. Leimkühler, L. Liu, M. Abhimitra and C. Theobalt, 'Drag Your GAN: Interactive Point-based Manipulation on the Generative Image Manifold,' 2023. [Online]. Available: https://vcai.mpi-inf.mpg.de/projects/DragGAN/ (visited on 31/05/2023).

[23] Y. Zeng, J.-L. Wu and H. Xiao, 'Enforcing Imprecise Constraints on Generative Adversarial Networks for Emulating Physical Systems,' en, *Communications in Computational Physics*, vol. 30, no. 3, pp. 635–665, Jun. 2021, Number: 3 Publisher: Global Science Press, ISSN: 1815-2406. [Online]. Available: https://resolver.caltech.edu/CaltechAUTHORS:20210910-173446842 (visited on 04/05/2023).

[24] P. Stinis, T. Hagge, A. M. Tartakovsky and E. Yeung, 'Enforcing constraints for interpolation and extrapolation in Generative Adversarial Networks,' *Journal of Computational Physics*, vol. 397, p. 108 844, Nov. 2019, arXiv:1803.08182 [cs, stat], ISSN: 00219991. DOI: 10.1016/j.jcp.2019.07.042. [Online]. Available: http://arxiv.org/abs/1803.08182 (visited on 15/05/2023).

[25] A. Ali and H. Kalisch, 'On the Formulation of Mass, Momentum and Energy Conservation in the KdV Equation,' en, *Acta Applicandae Mathematicae*, vol. 133, no. 1, pp. 113–131, Oct. 2014, ISSN: 1572-9036. DOI: 10.1007/s10440-013-9861-0. [Online]. Available: https://doi.org/10.1007/s10440-013-9861-0 (visited on 16/04/2023).

[26] K. Nadjahi, 'Sliced-Wasserstein distance for large-scale machine learning: Theory, methodology and extensions,' en, p. 173, Jan. 2022.

[27] M. Paskin, *1. Introduction to Probability Theory*, en. [Online]. Available: https://ai.stanford.edu/~paskin/gm-short-course/lec1.pdf.

[28] I. Sason, 'Divergence Measures: Mathematical Foundations and Applications in Information-Theoretic and Statistical Problems,' en, *Entropy*, vol. 24, no. 5, p. 712, May 2022, Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 1099-4300. DOI: 10.3390/e24050712. [Online]. Available: https://www.mdpi.com/1099-4300/24/5/712 (visited on 16/05/2023).

[29] C. Villani, *Optimal Transport* (Grundlehren der mathematischen Wissenschaften), M. Berger, B. Eckmann, P. de la Harpe, F. Hirzebruch, N. Hitchin, L. Hörmander, A. Kupiainen, G. Lebeau, M. Ratner, D. Serre, Y. G. Sinai, N. J. A. Sloane, A. M. Vershik and M. Waldschmidt, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 338, ISBN: 978-3-540-71049-3 978-3-540-71050-9. DOI: 10.1007/978-3-540-71050-9. [Online]. Available: http://link.springer.com/10.1007/978-3-540-71050-9 (visited on 20/11/2022).

[30] G. Peyré and M. Cuturi, *Computational Optimal Transport*, arXiv:1803.00567 [stat], Mar. 2020. [Online]. Available: http://arxiv.org/abs/1803.00567 (visited on 08/03/2023).

[31] L. V. Kantorovich, 'Mathematical Methods of Organizing and Planning Production,' *Management Science*, vol. 6, no. 4, pp. 366–422, Jul. 1960, Publisher: INFORMS, ISSN: 0025-1909. DOI: 10.1287/mnsc.6.4.366. [Online]. Available: https://pubsonline.informs.org/doi/10.1287/mnsc.6.4.366 (visited on 27/03/2023).

[32] P. Clement and W. Desch, 'An elementary proof of the triangle inequality for the Wasserstein metric,' en, *Proceedings of the American Mathematical Society*, vol. 136, no. 1, pp. 333–339, Sep. 2007, ISSN: 0002-9939, 1088-6826. DOI: 10.1090/S0002-9939-07-09020-X. [Online]. Available: https://www.ams.org/proc/2008-136-01/S0002-9939-07-09020-X/ (visited on 09/03/2023).

[33] C. R. Givens and R. M. Shortt, 'A class of Wasserstein metrics for probability distributions,' *Michigan Mathematical Journal*, vol. 31, no. 2, pp. 231–240, 1984, ISSN: 0026-2285. DOI: 10.1307/mmj/1029003026. [Online]. Available: https://mathscinet.ams.org/mathscinet-getitem?mr=752258 (visited on 10/03/2023).

[34] M. Cuturi, 'Sinkhorn Distances: Lightspeed Computation of Optimal Transport,' in *Advances in Neural Information Processing Systems*, vol. 26, Curran Associates, Inc., 2013. [Online]. Available: https://papers.nips.cc/paper_files/paper/2013/hash/af21d0c97db2e27e13572cbf59eb343d-Abstract.html (visited on 07/05/2023).

[35] J. Rabin, G. Peyré, J. Delon and M. Bernot, 'Wasserstein Barycenter and Its Application to Texture Mixing,' en, in *Scale Space and Variational Methods in Computer Vision*, A. M. Bruckstein, B. M. ter Haar Romeny, A. M. Bronstein and M. M. Bronstein, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2012, pp. 435–446, ISBN: 978-3-642-24785-9. DOI: 10.1007/978-3-642-24785-9_37.

[36] G. Reeves, 'Conditional central limit theorems for Gaussian projections,' in *2017 IEEE International Symposium on Information Theory (ISIT)*, ISSN: 2157-8117, Jun. 2017, pp. 3045–3049. DOI: 10.1109/ISIT.2017.8007089.

[37] E. Kohmann, 'Using GANs for Time Series Generation: A Preliminary Project,' en, Dec. 2022.

[38] M. Sugiyama, T. Suzuki and T. Kanamori, *Density Ratio Estimation in Machine Learning*, en, 1st ed. Cambridge University Press, Feb. 2012, ISBN: 978-0-521-19017-6 978-1-139-03561-3. DOI: 10.1017/CBO9781139035613. [Online]. Available: https://www.cambridge.org/core/product/identifier/9781139035613/type/book (visited on 20/09/2022).

[39] M. Sugiyama, T. Suzuki and T. Kanamori, 'Density Ratio Estimation: A Comprehensive Review,' en, p. 22,

[40] S. Liu, A. Takeda, T. Suzuki and K. Fukumizu, 'Trimmed Density Ratio Estimation,' in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/hash/ea204361fe7f024b130143eb3e189a18-Abstract.html (visited on 20/09/2022).

[41] S. Mohamed and B. Lakshminarayanan, *Learning in Implicit Generative Models*, arXiv:1610.03483 [cs, stat], Feb. 2017. [Online]. Available: http://arxiv.org/abs/1610.03483 (visited on 19/09/2022).

[42] L. Tiao, *Density Ratio Estimation for KL Divergence Minimization between Implicit Distributions*, en-us, Aug. 2018. [Online]. Available: https://tiao.io/post/density-ratio-estimation-for-kl-divergence-minimization-between-implicit-distributions/ (visited on 20/09/2022).

[43] N. Kodali, J. Abernethy, J. Hays and Z. Kira, *On Convergence and Stability of GANs*, arXiv:1705.07215 [cs], Dec. 2017. [Online]. Available: http://arxiv.org/abs/1705.07215 (visited on 12/09/2022).

[44] D. Saxena and J. Cao, *Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions*, arXiv:2005.00065 [cs, eess, stat], Oct. 2020. DOI: 10.48550/arXiv.2005.00065. [Online]. Available: http://arxiv.org/abs/2005.00065 (visited on 12/09/2022).

[45] M. Arjovsky, S. Chintala and L. Bottou, *Wasserstein GAN*, arXiv:1701.07875 [cs, stat], Dec. 2017. DOI: 10.48550/arXiv.1701.07875. [Online]. Available: http://arxiv.org/abs/1701.07875 (visited on 12/09/2022).

[46] J. Thickstun, 'Kantorovich-Rubinstein Duality,' en,

[47] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen, *Improved Techniques for Training GANs*, arXiv:1606.03498 [cs], Jun. 2016. [Online]. Available: http://arxiv.org/abs/1606.03498 (visited on 29/11/2022).

[48] C. K. Sønderby, J. Caballero, L. Theis, W. Shi and F. Huszár, *Amortised MAP Inference for Image Super-resolution*, arXiv:1610.04490 [cs, stat], Feb. 2017. [Online]. Available: http://arxiv.org/abs/1610.04490 (visited on 29/11/2022).

[49] L. Adolphs, H. Daneshmand, A. Lucchi and T. Hofmann, *Local Saddle Point Optimization: A Curvature Exploitation Approach*, arXiv:1805.05751 [cs, math, stat], Feb. 2019. [Online]. Available: http://arxiv.org/abs/1805.05751 (visited on 12/09/2022).

[50] Z. Lin, A. Khetan, G. Fanti and S. Oh, 'PacGAN: The Power of Two Samples in Generative Adversarial Networks,' *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 324–335, May 2020, ISSN: 2641-8770. DOI: 10.1109/JSAIT.2020.2983071. [Online]. Available: https://ieeexplore.ieee.org/document/9046238/ (visited on 15/06/2022).

[51] K. Liu, W. Tang, F. Zhou and G. Qiu, *Spectral Regularization for Combating Mode Collapse in GANs*, arXiv:1908.10999 [cs, stat], Oct. 2019. [Online]. Available: http://arxiv.org/abs/1908.10999 (visited on 12/09/2022).

[52] R. Durall, A. Chatzimichailidis, P. Labus and J. Keuper, *Combating Mode Collapse in GAN training: An Empirical Analysis using Hessian Eigenvalues*, arXiv:2012.09673 [cs], Dec. 2020. [Online]. Available: http://arxiv.org/abs/2012.09673 (visited on 12/09/2022).

[53] Q. Hoang, T. D. Nguyen, T. Le and D. Phung, 'MGAN: TRAINING GENERATIVE ADVERSARIAL NETS WITH MULTIPLE GENERATORS,' en, p. 24, 2018.

[54] M. Mirza and S. Osindero, *Conditional Generative Adversarial Nets*, arXiv:1411.1784 [cs, stat], Nov. 2014. DOI: 10.48550/arXiv.1411.1784. [Online]. Available: http://arxiv.org/abs/1411.1784 (visited on 13/09/2022).

[55] L. Metz, B. Poole, D. Pfau and J. Sohl-Dickstein, *Unrolled Generative Adversarial Networks*, arXiv:1611.02163 [cs, stat], May 2017. DOI: 10.48550/arXiv.1611.02163. [Online]. Available: http://arxiv.org/abs/1611.02163 (visited on 12/09/2022).

[56] O. Mogren, *C-RNN-GAN: Continuous recurrent neural networks with adversarial training*, arXiv:1611.09904 [cs], Nov. 2016. [Online]. Available: http://arxiv.org/abs/1611.09904 (visited on 22/09/2022).

[57] C. Esteban, S. L. Hyland and G. Rätsch, *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*, arXiv:1706.02633 [cs, stat], Dec. 2017. DOI: 10.48550/arXiv.1706.02633. [Online]. Available: http://arxiv.org/abs/1706.02633 (visited on 27/09/2022).

[58] S. Harada, H. Hayashi and S. Uchida, 'Biosignal Generation and Latent Variable Analysis With Recurrent Generative Adversarial Networks,' *IEEE Access*, vol. 7, pp. 144 292–144 302, 2019, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2934928.

[59] C. Zhang, S. R. Kuppannagari, R. Kannan and V. K. Prasanna, 'Generative Adversarial Network for Synthetic Time Series Data Generation in Smart Grids,' in *2018 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, Oct. 2018, pp. 1–6. DOI: 10.1109/SmartGridComm.2018.8587464.

[60] J. Yoon, D. Jarrett and M. van der Schaar, 'Time-series Generative Adversarial Networks,' in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/c9efe5f26cd17ba6216bbe2a7d26d490-Abstract.html (visited on 27/09/2022).

[61] T. Xu, L. K. Wenliang, M. Munn and B. Acciaio, 'COT-GAN: Generating Sequential Data via Causal Optimal Transport,' in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 8798–8809. [Online]. Available: https://papers.nips.cc/paper/2020/hash/641d77dd5271fca28764612a028d9c8e-Abstract.html (visited on 13/01/2023).

[62] J. B. Veraguas, M. Beiglböck, Y. Lin and A. Zalashko, *Causal transport in discrete time and applications*, arXiv:1606.04062 [math], Mar. 2017. DOI: 10.48550/arXiv.1606.04062. [Online]. Available: http://arxiv.org/abs/1606.04062 (visited on 19/02/2023).

[63] H. Pei, K. Ren, Y. Yang, C. Liu, T. Qin and D. Li, 'Towards Generating Real-World Time Series Data,' in *2021 IEEE International Conference on Data Mining (ICDM)*, Auckland, New Zealand: IEEE, Dec. 2021, pp. 469–478, ISBN: 978-1-66542-398-4. DOI: 10.1109/ICDM51629.2021.00058. [Online]. Available: https://ieeexplore.ieee.org/document/9679006/ (visited on 15/06/2022).

[64] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. Courville, *Improved Training of Wasserstein GANs*, arXiv:1704.00028 [cs, stat] version: 3, Dec. 2017. DOI: 10.48550/arXiv.1704.00028. [Online]. Available: http://arxiv.org/abs/1704.00028 (visited on 09/12/2022).

[65] F. Zhu, F. Ye, Y. Fu, Q. Liu and B. Shen, 'Electrocardiogram generation with a bidirectional LSTM-CNN generative adversarial network,' en, *Scientific Reports*, vol. 9, no. 1, p. 6734, May 2019, Number: 1 Publisher: Nature Publishing Group, ISSN: 2045-2322. DOI: 10.1038/s41598-019-42516-z. [Online]. Available: https://www.nature.com/articles/s41598-019-42516-z (visited on 28/09/2022).

[66] E. Brophy, M. De Vos, G. Boylan and T. Ward, 'Multivariate Generative Adversarial Networks and Their Loss Functions for Synthesis of Multichannel ECGs,' *IEEE Access*, vol. 9, pp. 158 936–158 945, 2021, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3130421.

[67] G. R. Khattak, S. Vallecorsa, F. Carminati and G. M. Khan, 'High Energy Physics Calorimeter Detector Simulation Using Generative Adversarial Networks With Domain Related Constraints,' *IEEE Access*, vol. 9, pp. 108 899–108 911, 2021, Conference Name: IEEE Access, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3101946.

[68] T. Che, Y. Li, A. P. Jacob, Y. Bengio and W. Li, *Mode Regularized Generative Adversarial Networks*, arXiv:1612.02136 [cs], Mar. 2017. [Online]. Available: http://arxiv.org/abs/1612.02136 (visited on 24/08/2022).

[69] J. Pan, J. Dong, Y. Liu, J. Zhang, J. Ren, J. Tang, Y.-W. Tai and M.-H. Yang, 'Physics-Based Generative Adversarial Models for Image Restoration and Beyond,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 7, pp. 2449–2462, Jul. 2021, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.2969348.

[70] D. Tretiak, A. T. Mohan and D. Livescu, *Physics-Constrained Generative Adversarial Networks for 3D Turbulence*, arXiv:2212.00217 [physics], Nov. 2022. DOI: 10.48550/arXiv.2212.00217. [Online]. Available: http://arxiv.org/abs/2212.00217 (visited on 04/05/2023).

[71] Y. Xie, E. Franz, M. Chu and N. Thuerey, 'tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow,' *ACM Transactions on Graphics*, vol. 36, no. 4, pp. 1–14, Aug. 2017, arXiv:1801.09710 [cs], ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3072959.3073643. [Online]. Available: http://arxiv.org/abs/1801.09710 (visited on 15/05/2023).

[72] Q. Zheng, L. Zeng, Z. Cao and G. E. Karniadakis, *Physics-informed semantic inpainting: Application to geostatistical modeling*, arXiv:1909.09459 [physics, stat], Dec. 2019. [Online]. Available: http://arxiv.org/abs/1909.09459 (visited on 15/05/2023).

[73] A. Subramaniam, M. L. Wong, R. D. Borker, S. Nimmagadda and S. K. Lele, *Turbulence Enrichment using Physics-informed Generative Adversarial Networks*, arXiv:2003.01907 [physics], Mar. 2020. [Online]. Available: http://arxiv.org/abs/2003.01907 (visited on 15/05/2023).

[74] C. Chen, G. Zheng, H. Wei and Z. Li, 'Physics-informed Generative Adversarial Networks for Sequence Generation with Limited Data,' en, 2020.

[75] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf and A. Smola, 'A Kernel Two-Sample Test,' *Journal of Machine Learning Research*, vol. 13, no. 25, pp. 723–773, 2012, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v13/gretton12a.html (visited on 22/05/2023).

[76] G. Al-Naymat, S. Chawla and J. Taheri, *SparseDTW: A Novel Approach to Speed up Dynamic Time Warping*, arXiv:1201.2969 [cs], Jan. 2012. [Online]. Available: http://arxiv.org/abs/1201.2969 (visited on 22/05/2023).

[77] H. Arnout, J. Bronner and T. Runkler, 'Evaluation of Generative Adversarial Networks for Time Series Data,' in *2021 International Joint Conference on Neural Networks (IJCNN)*, ISSN: 2161-4407, Jul. 2021, pp. 1–7. DOI: 10.1109/IJCNN52387.2021.9534373.

[78] P. Srinivasan and W. J. Knottenbelt, *Time-series Transformer Generative Adversarial Networks*, arXiv:2205.11164 [cs], May 2022. [Online]. Available: http://arxiv.org/abs/2205.11164 (visited on 18/08/2022).

[79] S. Asre and A. Anwar, 'Synthetic Energy Data Generation Using Time Variant Generative Adversarial Network,' en, *Electronics*, vol. 11, no. 3, p. 355, Jan. 2022, Number: 3 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-9292. DOI: 10.3390/electronics11030355. [Online]. Available: https://www.mdpi.com/2079-9292/11/3/355 (visited on 28/09/2022).

[80] M. Lee, D. Tae, J. H. Choi, H.-Y. Jung and J. Seok, 'Improved recurrent generative adversarial networks with regularization techniques and a controllable framework,' en, *Information Sciences*, vol. 538, pp. 428–443, Oct. 2020, ISSN: 0020-0255. DOI: 10.1016/j.ins.2020.05.116. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520305417 (visited on 01/10/2022).

[81] H. Ni, L. Szpruch, M. Wiese, S. Liao and B. Xiao, *Conditional Sig-Wasserstein GANs for Time Series Generation*, arXiv:2006.05421 [cs, stat], Jun. 2020. [Online]. Available: http://arxiv.org/abs/2006.05421 (visited on 05/10/2022).

[82] M. Wiese, R. Knobloch, R. Korn and P. Kretschmer, 'Quant GANs: Deep Generation of Financial Time Series,' *Quantitative Finance*, vol. 20, no. 9, pp. 1419–1440, Sep. 2020, arXiv:1907.06673 [cs, q-fin, stat], ISSN: 1469-7688, 1469-7696. DOI: 10.1080/14697688.2020.1730426. [Online]. Available: http://arxiv.org/abs/1907.06673 (visited on 11/11/2022).

[83] M. Dogariu, L.-D. Ştefan, B. A. Boteanu, C. Lamba and B. Ionescu, 'Towards Realistic Financial Time Series Generation via Generative Adversarial Learning,' in *2021 29th European Signal Processing Conference (EUSIPCO)*, ISSN: 2076-1465, Aug. 2021, pp. 1341–1345. DOI: 10.23919/EUSIPCO54536.2021.9616176.

[84] D. Snow, *MTSS-GAN: Multivariate Time Series Simulation Generative Adversarial Networks*, en, SSRN Scholarly Paper, Rochester, NY, Jun. 2020. DOI: 10.2139/ssrn.3616557. [Online]. Available: https://papers.ssrn.com/abstract=3616557 (visited on 28/09/2022).

[85] J. Hellermann and S. Lessmann, 'Leveraging Image-based Generative Adversarial Networks for Time Series Generation,' en, *undefined*, 2021. (visited on 27/09/2022).

[86] X. Li, V. Metsis, H. Wang and A. H. H. Ngu, *TTS-GAN: A Transformer-based Time-Series Generative Adversarial Network*, arXiv:2202.02691 [cs], Jun. 2022. DOI: 10.48550/arXiv.2202.02691. [Online]. Available: http://arxiv.org/abs/2202.02691 (visited on 21/09/2022).

[87] L. McInnes, J. Healy and J. Melville, 'UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction,' en, Feb. 2018. DOI: 10.48550/arXiv.1802.03426. [Online]. Available: https://arxiv.org/abs/1802.03426v3 (visited on 06/12/2022).

[88] K. Pearson, 'LIII. On lines and planes of closest fit to systems of points in space,' Nov. 1901. DOI: 10.1080/14786440109462720. [Online]. Available: https://zenodo.org/record/1430636 (visited on 08/12/2022).

[89] L. v. d. Maaten and G. Hinton, 'Visualizing Data using t-SNE,' *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008, ISSN: 1533-7928. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html (visited on 20/11/2022).

[90] K. Brauer, 'The Korteweg-de Vries Equation: History, exact Solutions, and graphical Representation,' en, Jun. 2000.

[91] A. Jabbar, X. Li and B. Omar, 'A Survey on Generative Adversarial Networks: Variants, Applications, and Training,' *ACM Computing Surveys*, vol. 54, no. 8, 157:1–157:49, Oct. 2021, ISSN: 0360-0300. DOI: 10.1145/3463475. [Online]. Available: https://dl.acm.org/doi/10.1145/3463475 (visited on 23/05/2023).

[92] V. Dumont, X. Ju and J. Mueller, 'Hyperparameter Optimization of Generative Adversarial Network Models for High-Energy Physics Simulations,' Tech. Rep., Oct. 2022, arXiv:2208.07715 [hep-ex, physics:physics]. DOI: 10.21203/rs.3.rs-2181360/v1. [Online]. Available: http://arxiv.org/abs/2208.07715 (visited on 01/02/2023).

[93] S. Eidnes, B. Owren and T. Ringholm, 'Adaptive energy preserving methods for partial differential equations,' en, *Advances in Computational Mathematics*, vol. 44, no. 3, pp. 815–839, Jun. 2018, ISSN: 1572-9044. DOI: 10.1007/s10444-017-9562-8. [Online]. Available: https://doi.org/10.1007/s10444-017-9562-8 (visited on 04/06/2023).

[94] J. W. Pratt and J. D. Gibbons, 'Kolmogorov-Smirnov Two-Sample Tests,' en, in *Concepts of Nonparametric Theory*, ser. Springer Series in Statistics, J. W. Pratt and J. D. Gibbons, Eds., New York, NY: Springer, 1981, pp. 318–344, ISBN: 978-1-4612-5931-2. DOI: 10.1007/978-1-4612-5931-2_7. [Online]. Available: https://doi.org/10.1007/978-1-4612-5931-2_7 (visited on 22/04/2023).

[95] R. A. Fisher, *Statistical methods for research workers*, eng. Edinburgh, Oliver and Boyd, 1938, ISBN: 978-0-05-002170-5. [Online]. Available: http://archive.org/details/statisticalmethoe7fish (visited on 04/06/2023).

[96] T. Karras, T. Aila, S. Laine and J. Lehtinen, *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, arXiv:1710.10196 [cs, stat], Feb. 2018. DOI: 10.48550/arXiv.1710.10196. [Online]. Available: http://arxiv.org/abs/1710.10196 (visited on 30/05/2023).

[97] S. Xiang and H. Li, *On the Effects of Batch and Weight Normalization in Generative Adversarial Networks*, arXiv:1704.03971 [cs, stat], Dec. 2017. DOI: 10.48550/arXiv.1704.03971. [Online]. Available: http://arxiv.org/abs/1704.03971 (visited on 31/05/2023).

[98] S. Ye, X. Hu and X. Xu, *Tdcgan: Temporal Dilated Convolutional Generative Adversarial Network for End-to-end Speech Enhancement*, arXiv:2008.07787 [eess], Sep. 2020. DOI: 10.48550/arXiv.2008.07787. [Online]. Available: http://arxiv.org/abs/2008.07787 (visited on 30/05/2023).

[99] Z. Lin, A. Jain, C. Wang, G. Fanti and V. Sekar, 'Using GANs for Sharing Networked Time Series Data: Challenges, Initial Promise, and Open Questions,' in *Proceedings of the ACM Internet Measurement Conference*, arXiv:1909.13403 [cs, stat], Oct. 2020, pp. 464–483. DOI: 10.1145/3419394.3423643. [Online]. Available: http://arxiv.org/abs/1909.13403 (visited on 22/09/2022).

Appendix A

Additional Material

A.1 Experiment: Framework testing, more results

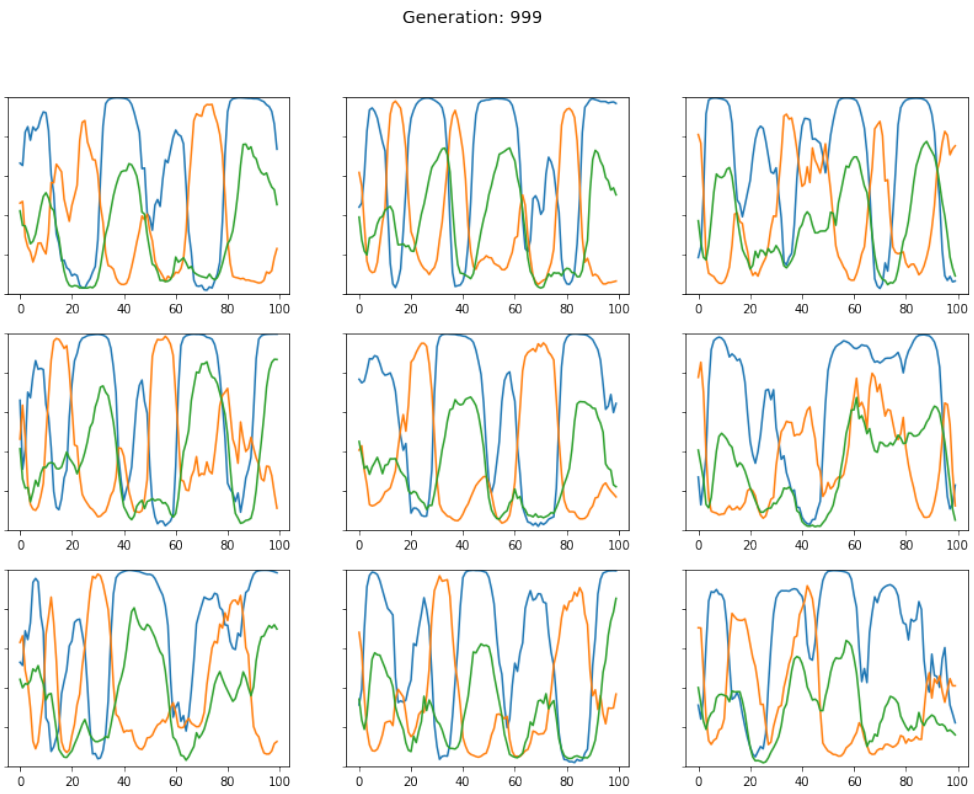


Figure A.1: Additional examples of the best performing RGAN model on sinusoidal data.

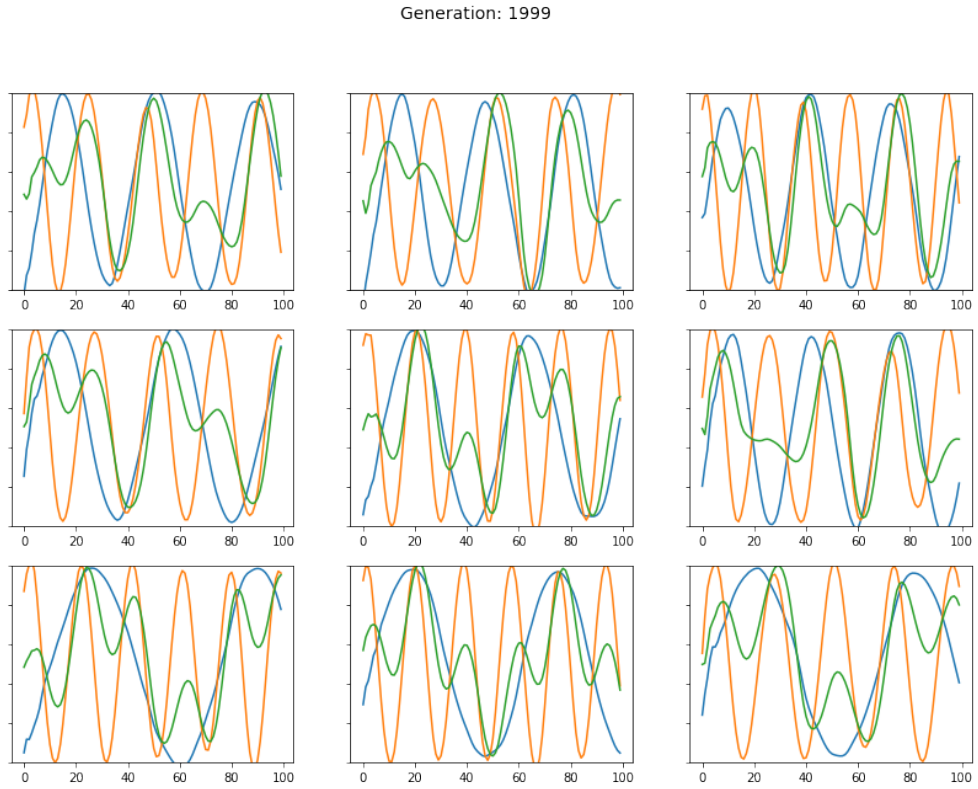


Figure A.2: Additional examples of the best performing TimeGAN model on sinusoidal data.

A.2 Experiment: Learning of underlying physical laws, more results

A.2.1 Solitons

A.2.2 Colliding solitary waves

A.2.3 Autoencoder and conservation regularization

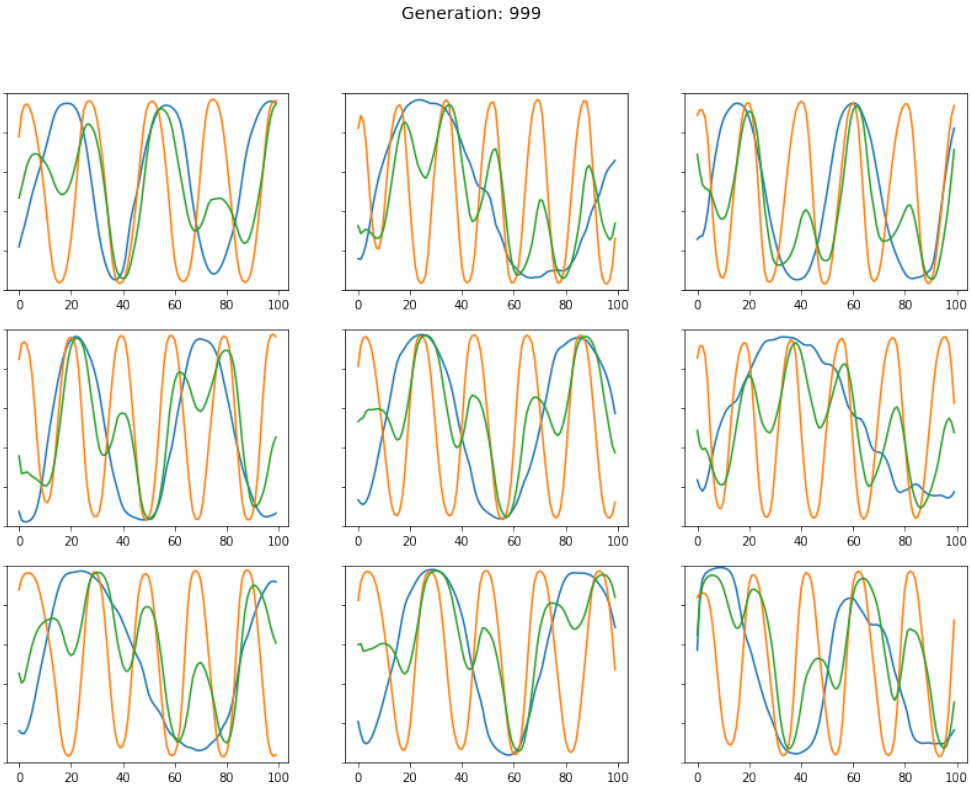


Figure A.3: Additional examples of the best performing RTSGAN model on sinusoidal data.

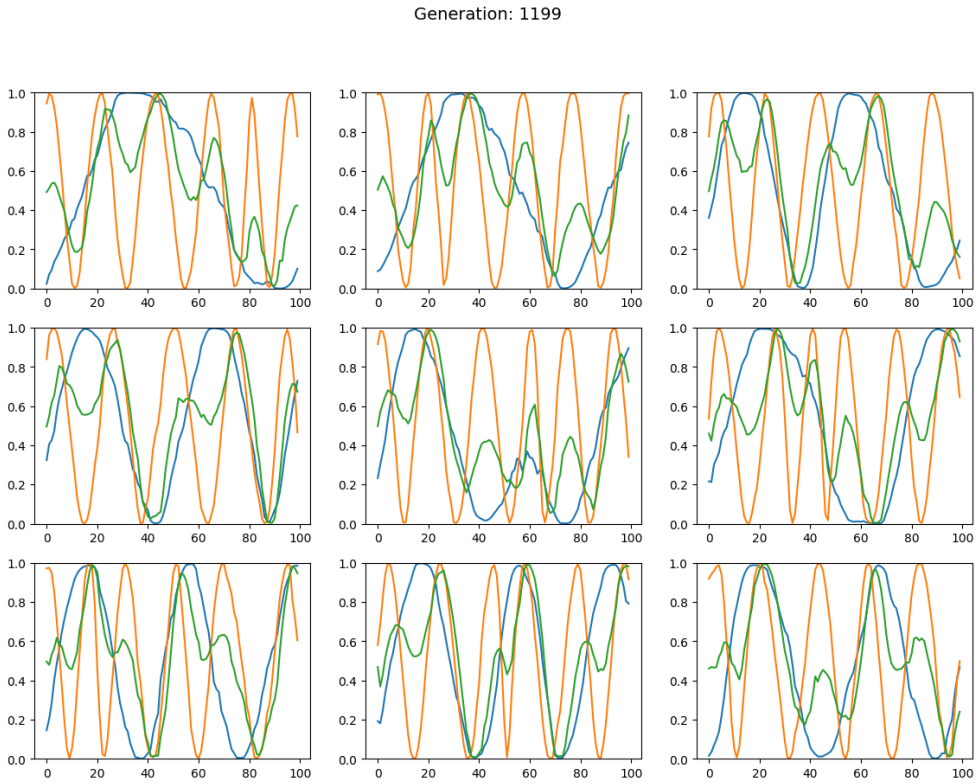
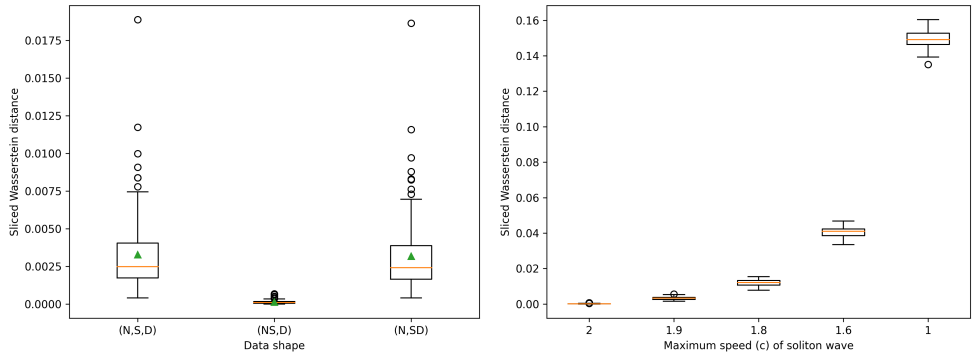


Figure A.4: Additional examples of the best performing COT-GAN model on sinusoidal data.



(a) SW distance results using three different**(b)** The sensitivity of having (N, S, D)-shapes data shapes. Two adjacent letters indicate adata when minor adjustments to the dataset dimension equal to their product.

Figure A.5: SW distance statistics of different data collections containing 1000 randomly generated samples using equation 7.2 with one of the datasets having different soliton height upper ranges. The boxplots are based on 100 different runs.

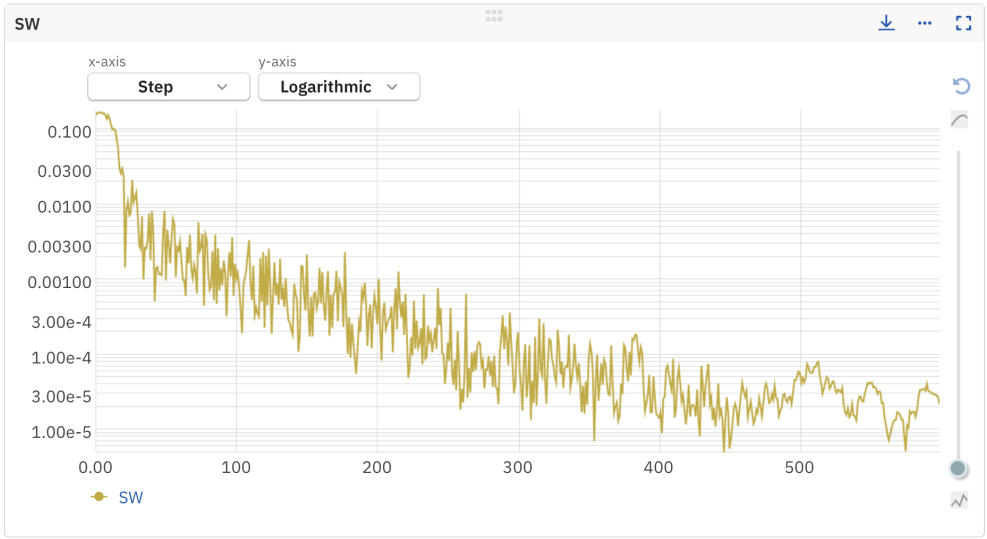


Figure A.6: The SW distance monitored over the training period for the soliton dataset.

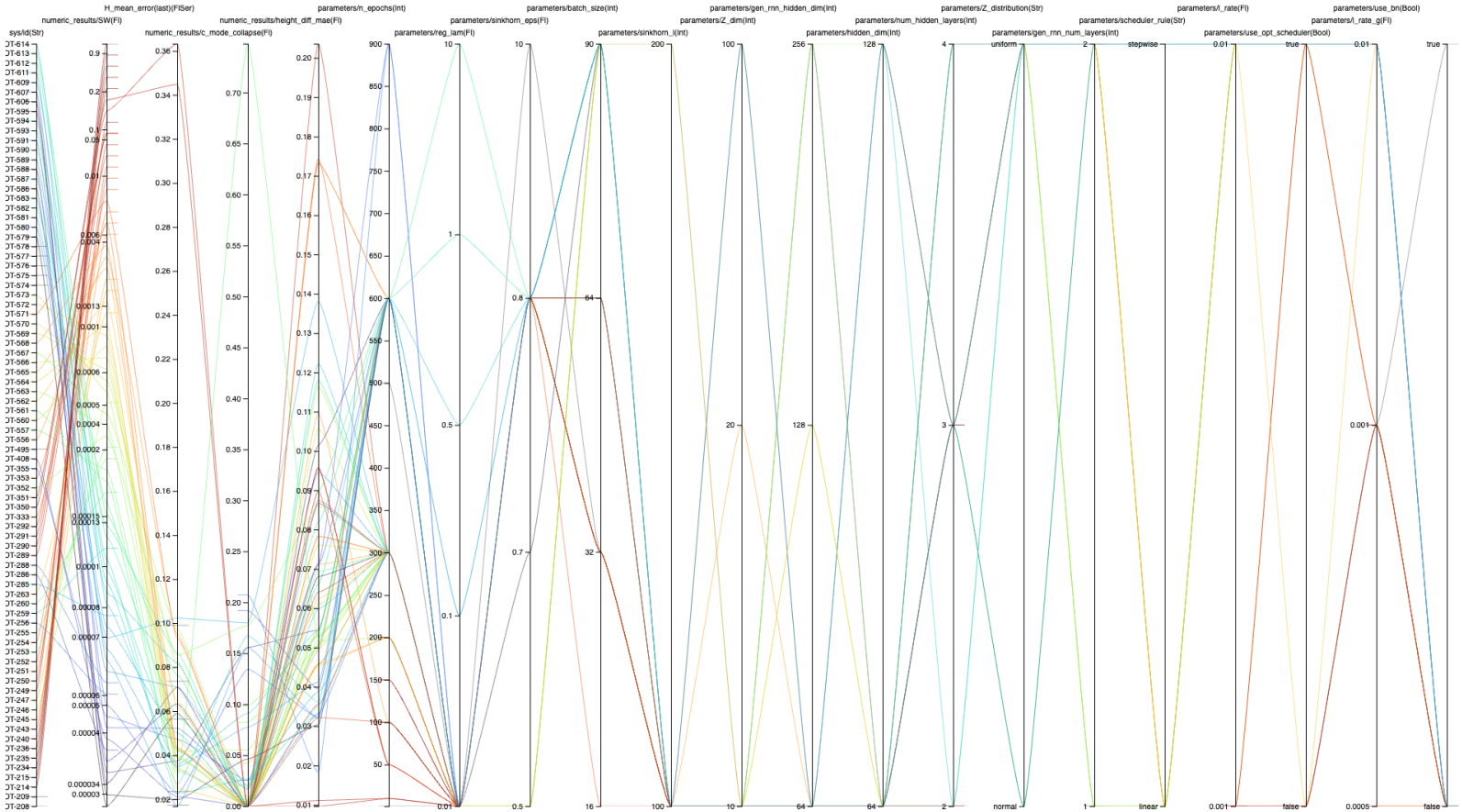


Figure A.7: A HiPlot that illustrates the various hyperparameters and architecture configurations that were experimented on with 80 different models. The evaluation metrics used to assess the models' performances on the Solitons dataset are also included in the image, with the first column indicating different runs (models) and the next two columns representing the SW and $\mathcal{H}_{\text{Energy}}$. The colored lines signify the hyperparameters related to each model and are shaded according to their SW performance, using a color spectrum that ranges from red, which represents poor performance, to violet, which represents good performance. The improvements made by finding better and better hyperparameters can be easily seen as the far-left column shows the most recent runs, with the newest model being on the top.

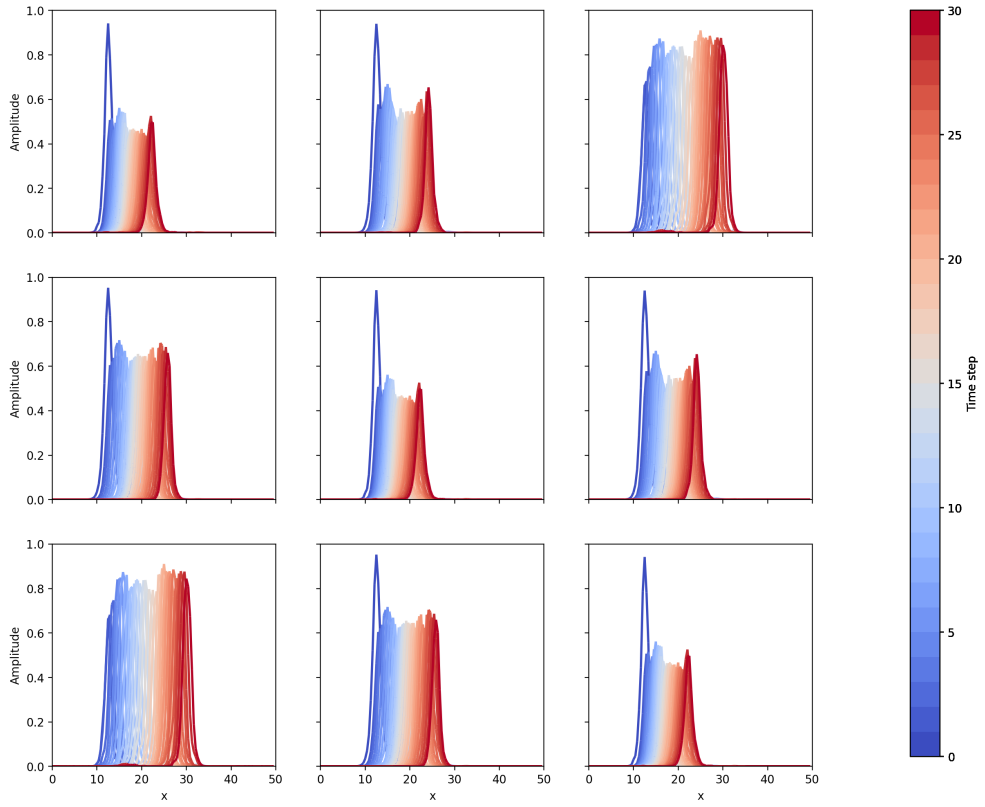


Figure A.8: A set of anomalies based on which samples contributed the most to the conservation error.

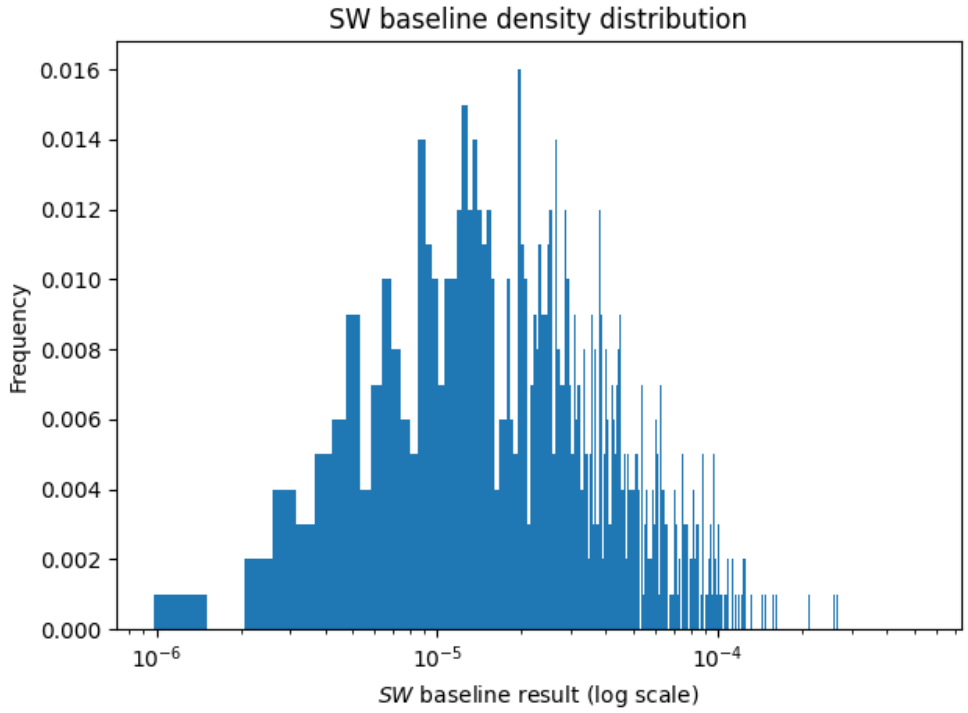


Figure A.9: Different results are obtained for the *SW* baseline when considering many different sample sets from the target distribution. This plot resembles the density distribution for 1000 different *SW* baseline calculations on the Soliton dataset, where sample sets contain 2000 samples.

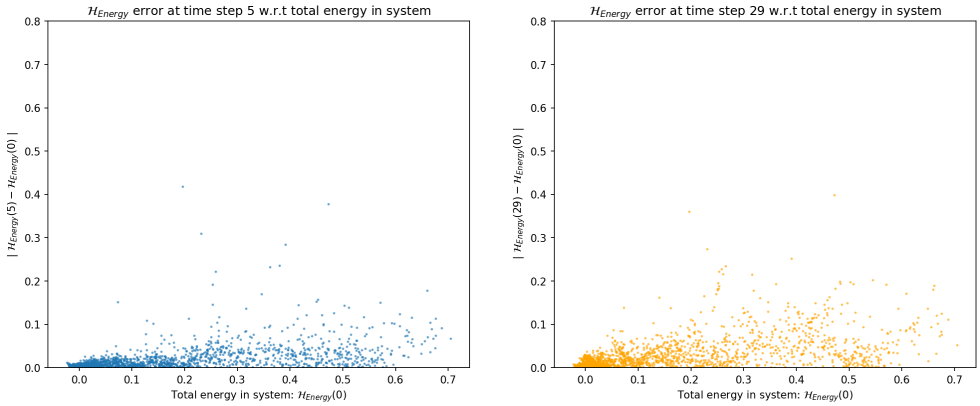


Figure A.10: Scatter plot illustrating the relationship between the energy error ($\mathcal{H}_{\text{Energy}}$) of 2000 colliding solitons and the total energy in the system at two different time steps, time step 5 and the final time step (29). At the final stages of the simulation (right plot), the error is generally larger, but also for systems containing consisting of more energy. However, there are still many samples with low error, similar to the early phases.

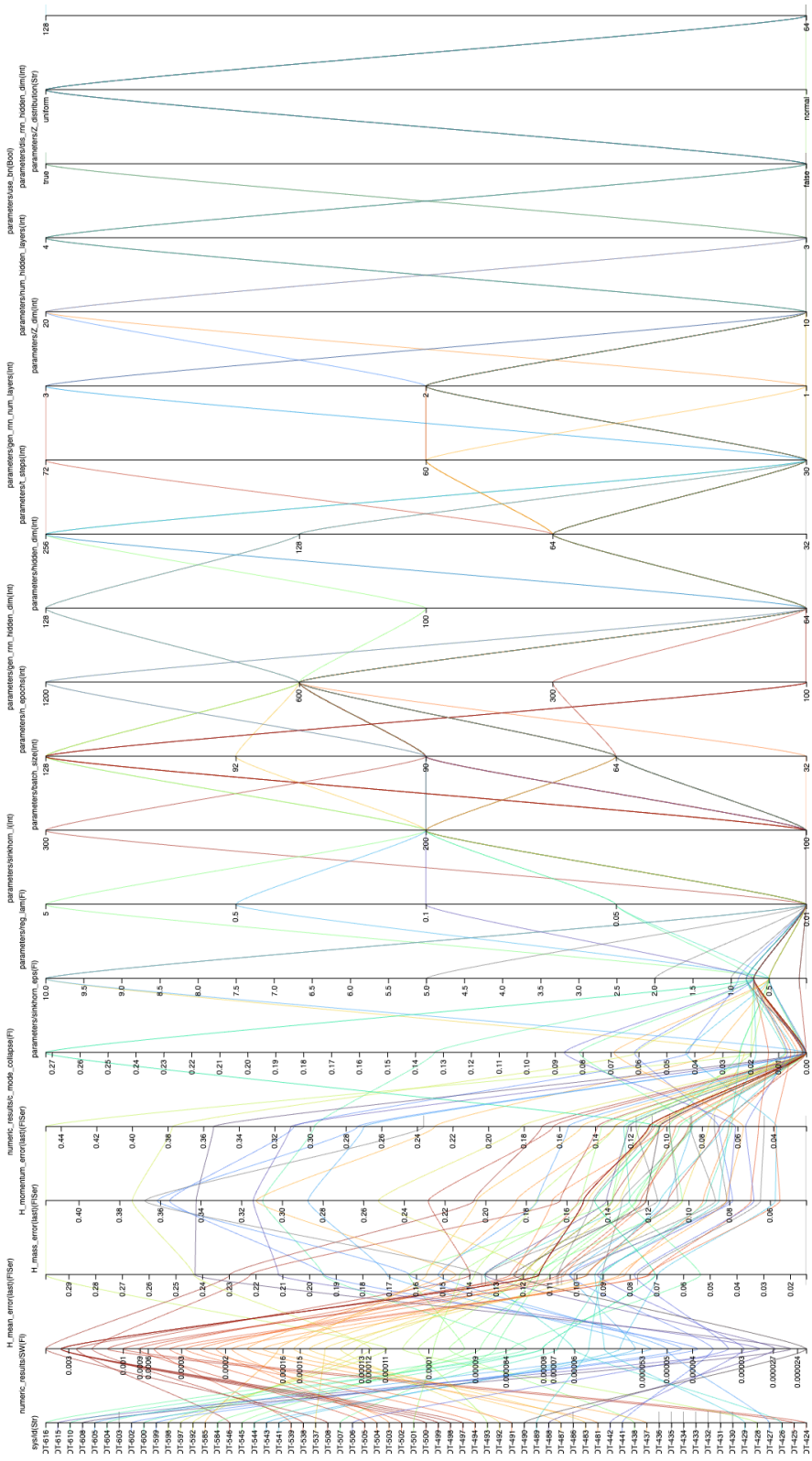


Figure A.11: A HiPlot illustrates the various hyperparameters and architecture configurations experimented on with 63 different models. The evaluation metrics used to assess the models' performances on the Colliding solitary waves dataset are also included in the image, with the first column indicating different runs (models) and the next two columns representing the SW and $\mathcal{H}_{\text{Energy}}$ metrics. The colored lines signify the hyperparameters related to each model. They are shaded according to their SW performance, using a color spectrum ranging from red, representing poor performance, to violet, representing good performance.

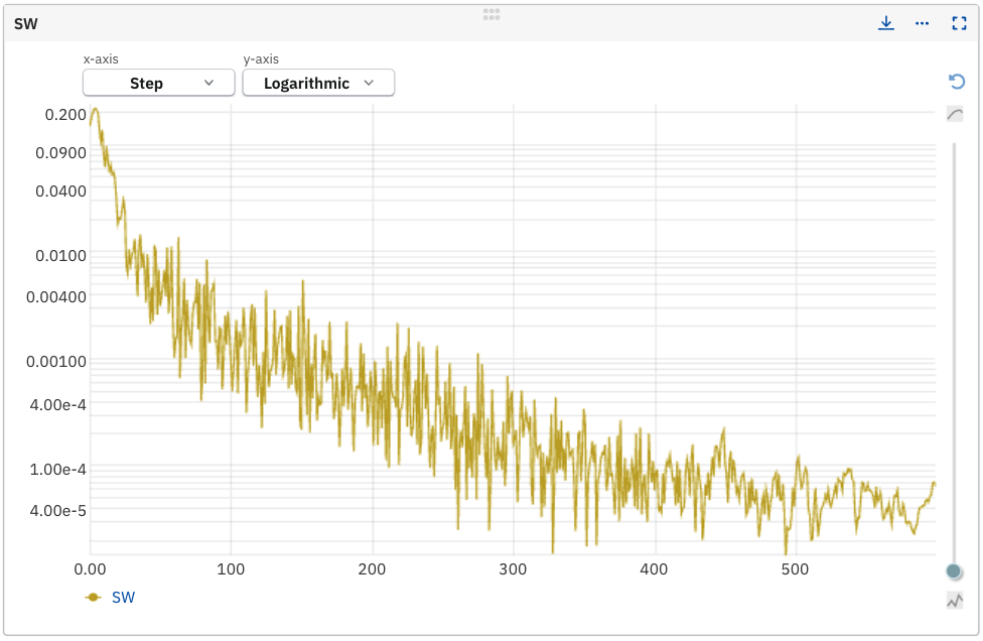


Figure A.12: The SW distance monitored over the training period for the colliding solitary waves dataset.

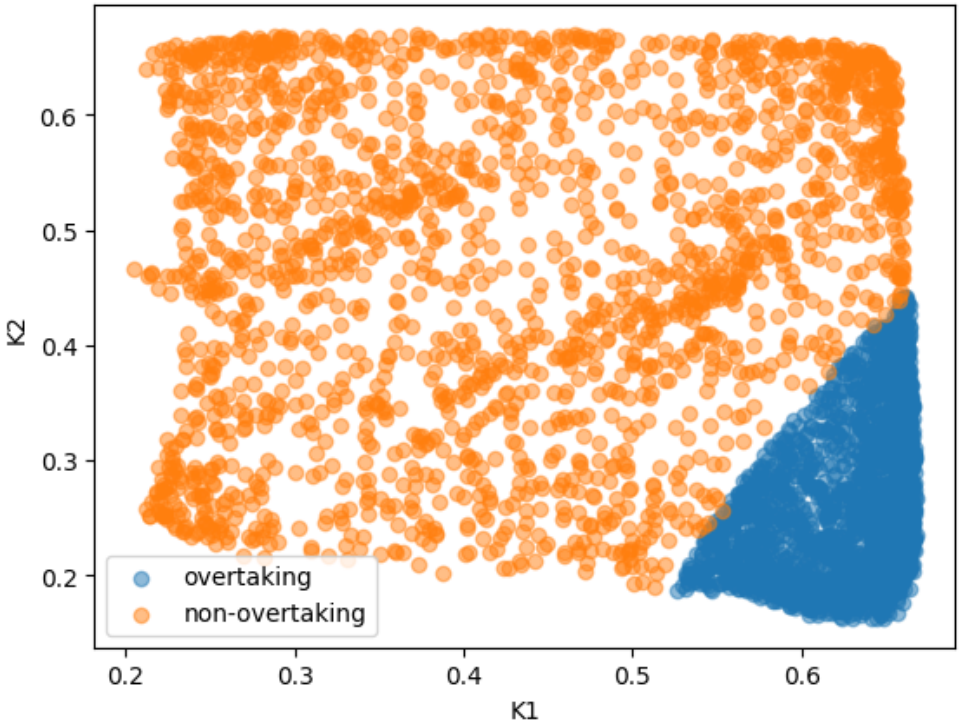
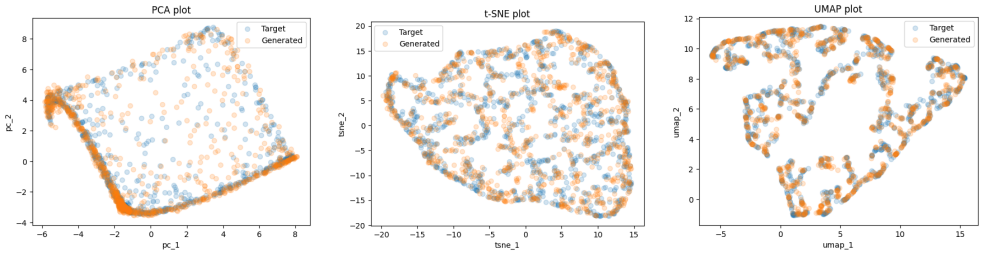


Figure A.13: An overview of different initial conditions which result in the left-soliton overtaking the other. The axis represents the height of the left and right soliton, K_1 and K_2, respectively. The combination which leads to the overtaking is colored in blue, which represents approximately 12% of the samples.



(a) PCA **(b)** t-SNE **(c)** UMAP

Figure A.14: Visualization of the target and generated samples from the physics-informed model. High-dimensional target (blue) and generated (orange) samples are visualized on a 2-dimensional plane using different methods. Each plot suggests the generated samples are fairly close to the target samples.

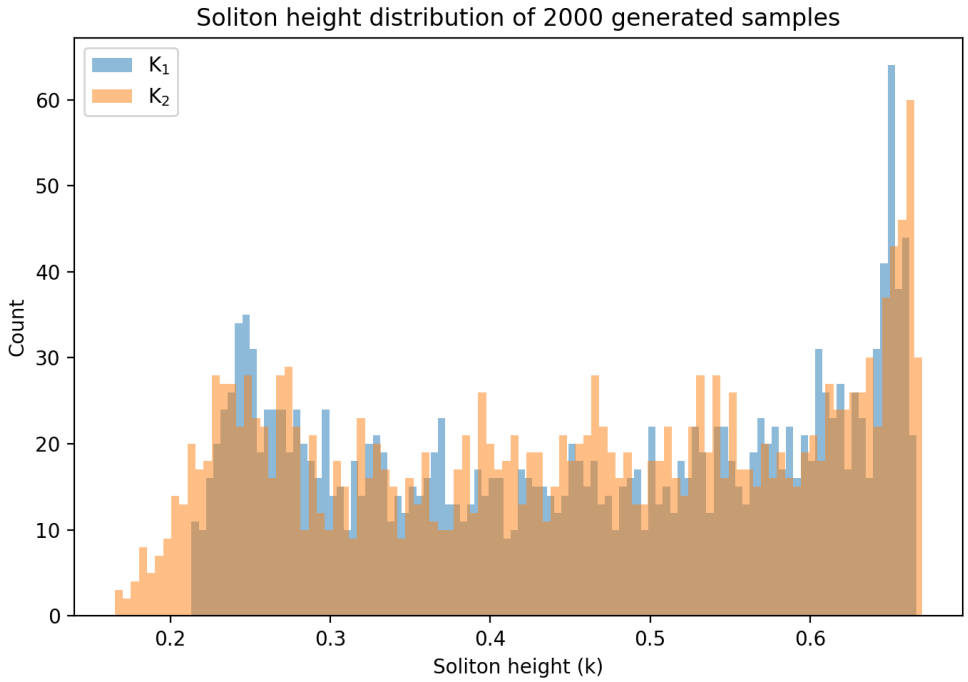


Figure A.15: A visual plot of the density distribution of 2000 generated soliton heights for the fully-trained model. The KS test associated with the two distributions (K_1, K_2) received an average p-value of 0.0001, indicating that the generated sample does not entirely generate the modes uniformly.

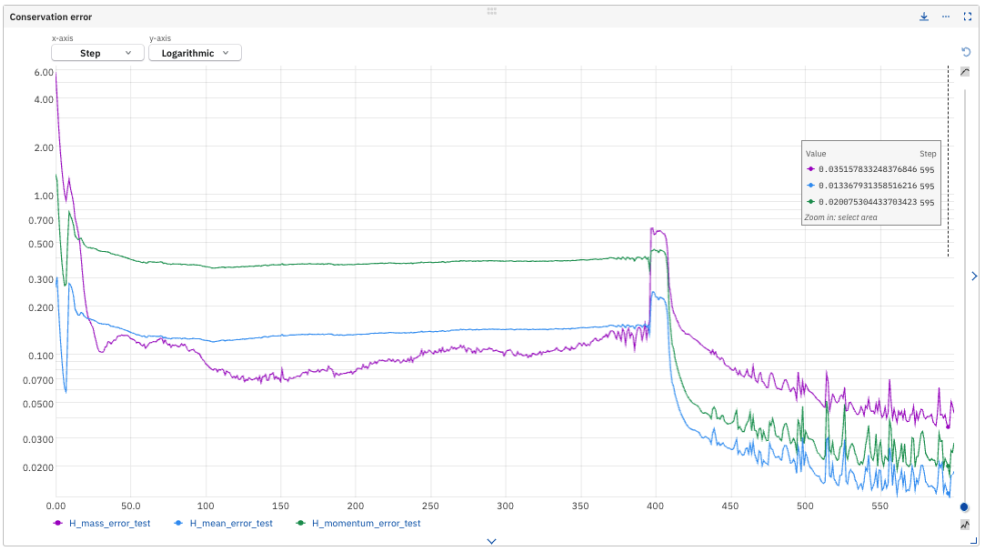


Figure A.16: During the 600-epoch training period of the autoencoder, all three measures of energy conservation loss decreased and stabilized. The values at the last epoch are reported in Table 8.3. The y-axis is in the logarithmic scale to make small changes more prominent.

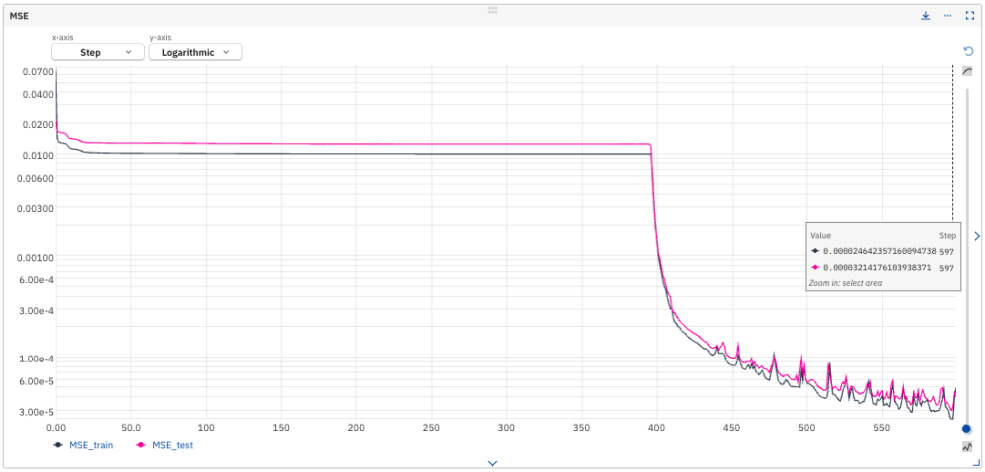
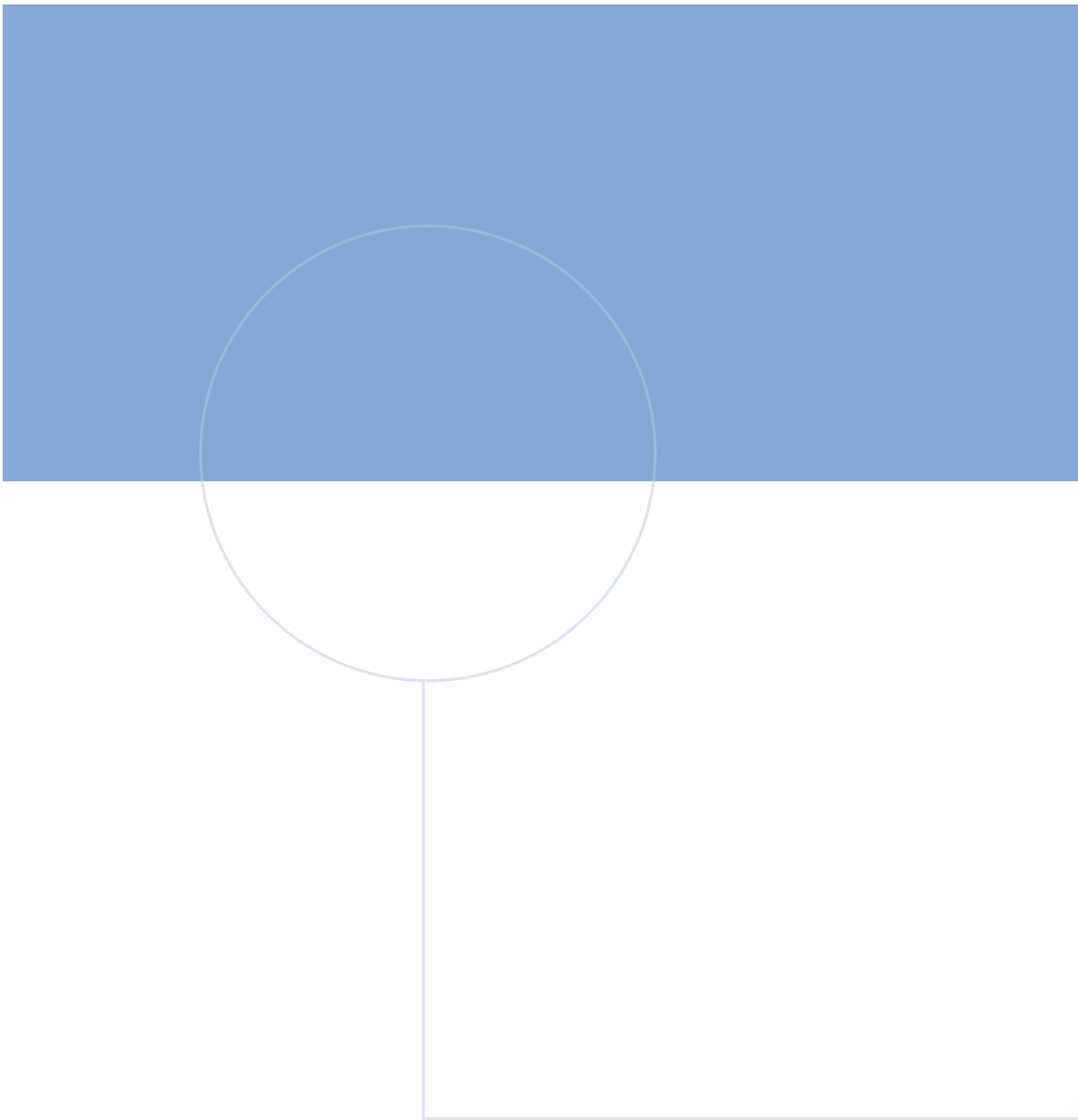


Figure A.17: The mean squared errors of both the train and validation dataset over the training period of 600 epochs. The errors plateaued for a while without any noticeable improvements until they eventually continued to improve.



NTNU

Norwegian University of
Science and Technology