University of Cape Town

FTX5052W

Minor Dissertation

An Application of Generative Adversarial Networks to One-Dimensional Value-at-Risk

---

Author:
Rachel Swallow

Student Number:
SWLRAC001

December 12, 2023

Abstract

A generative adversarial network (GAN) is an implicit generative model made up of two neural networks. This minor dissertation applies GANs to recover target statistical distributions. GANs have a distinctive training architecture designed to create examples that reproduce target data samples. These models have been applied successfully in high-dimensional domains such as natural image generation and processing. Much less research has been reported on applications with low dimensional distributions, where properties of GANs may be better identified and understood. One such area in finance is the use of GANs for estimating value-at-risk (VaR). Through this financial application, this dissertation introduces readers to the concepts and practical implementations of GAN variants to generate one-dimensional portfolio returns over a single period. Large portions of the discussions should be accessible to anyone who has an entry-level statistics course. It is aimed at data science or finance students looking to better their understanding of GANs and the potential of these models for other financial applications. Five GAN loss variants are introduced and three of these models are practically implemented to estimate VaR. The GAN estimates are compared to more traditional VaR estimation techniques and all models are backtested. Most GAN models trained in this dissertation are able to capture key features of each of the distributions, however these models do not outperform historical VaR estimates.

Acknowledgements

I would like to thank the following individuals and organisations who have played a significant role in the completion of this dissertation. First and foremost, thank you to my supervisor, Dr Obeid Mahomed, for their constructive feedback and thoughtful suggestions throughout the entire research process. Their expertise and attention to detail have been a highlight of this experience. Special thanks go to the AIFMRM faculty for providing me with the necessary resources, coursework and for encouraging me to push myself academically. Thank you to my friends and classmates who made the process so much fun. Lastly, I would like to thank my family for their support, multiple proofreads and encouragement throughout this academic journey.

This dissertation would not have been possible without the collective contributions of these individuals and institutions. I am truly grateful for their support.

Plagiarism Declaration

1. Plagiarism is the use of ideas, material and other intellectual property of another's work and to present it as my own.
2. I agree that plagiarism is a punishable offence because it constitutes theft.
3. Accordingly, all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism.
4. I also understand that direct translations are plagiarism.
5. I declare that the work contained in this dissertation, except otherwise stated, is my original work.

Signed by candidate

Rachel Swallow

12/12/2023

Date

Contents

List of Figures

List of Tables

1 Introduction

In the landmark paper by Ian Goodfellow et al [6] the Generative Adversarial Network (GAN) framework is presented. Since then, hundreds of GAN variants have been introduced, an overview and classification has been provided in [22], and we have seen successful applications in a variety of fields. The majority of research and success of GANs has been in the realm of natural images. GANs have been used to generate high-quality images [11], in image-to-image translation [9], overlaying special effects like realistic ageing [1], and the creation of high-quality animated characters [10], amongst others.

The GAN framework can be viewed as a two-player game made up of a discriminator and a generator, both of which are parameterised as neural networks. The generator's goal is to learn to transform samples from the noise, or latent, distribution to the chosen target distribution so that the discriminator cannot distinguish whether the sample belongs to the target or the generated distribution. The GAN framework can also be viewed as minimising a statistical divergence between the target and generated distribution. In this dissertation, GANs with different objective functions have been selected for comparison. The GANs presented in [6], the Wasserstein GAN (WGAN) in [2], the WGAN with gradient penalty presented in [8] and the Maximum-Mean Discrepancy (MMD) GAN in [14], have been investigated.

GANs can be trained to recover target statistical distributions and they have the proposed advantage that once trained, they can generate unlimited samples from that distribution. This, in theory, means GANs can provide a much richer dataset than empirical methods can produce. As much of the investigation into the ability of GANs to produce realistic synthetic data has been in higher dimensions - such as natural images - the question arises whether GANs can learn to sample from much lower dimensions. In this dissertation, we aim to investigate the ability of a GAN to generate samples from one-dimensional (1D) target distributions. These generated samples can enhance sample sets used for calculating historical value-at-risk (VaR).

VaR was introduced in the late 1980s. Since then it has become commonplace in financial institutions, aided by the publication of the Basel II accord in 2004. Basel II made VaR an important component of the regulatory framework of the Basel Committee of Banking Supervision (BCBS). VaR is the maximum loss that, with a certain confidence level, will not be exceeded over a defined period. It is a quantile risk metric so, for it to be reliable, the estimation of the profit and loss distribution must be accurate. In popular classical methods, VaR is calculated using parametric statistical techniques or by using historical returns. These methods have limitations as they may not accurately capture the true distribution of returns and thus do not provide an accurate VaR estimation. Historical VaR is often used in industry as it

is interpretable and requires minimal assumptions. The sample size available for its calculation is often limited which can affect the accuracy of the forecast.

This dissertation aims to provide a thorough theoretical understanding of GANs, how they are trained and how they perform in a simple 1D setting. We focus on the distribution of returns over a single period and will not consider 1D time series generation of returns as seen in work presented in [27], [24] and [23]. This will add to the small amount of research done in using GANs to enhance VaR - which to our knowledge consists of two blog posts [12] [18] and one master's thesis [4]. The two blog posts provide a very high-level overview of the topic but do not backtest their models. Fiechtner's master's thesis [4] uses two GAN variants, the WGAN GAN and the Non-saturating (NS) GAN, to enhance empirical return samples for historical VaR. Both time series generation and the generation of stationary distributions of empirical returns were covered. The results were promising and in-line with classical VaR models. This dissertation adds to Fiechtner's work on stationary distribution generation, by testing an additional GAN, the MMD GAN, on parametric target distributions.

This dissertation trained three GAN loss variants, namely the WGAN, the MMD GAN and the NS GAN, to generate target parametric distributions with different training sample sets. The distributions of interest were a normal distribution and two different Gaussian mixture models. The generated samples were assessed with regard to the distributional characteristics of the sample as well as the sample VaR when compared to traditional historical VaR techniques. The effect of the enhanced sample on the VaR was in line with historical VaR models - however, some models were unstable during training and provided unsatisfactory results. These findings suggest that GAN models can reproduce key statistical features of parametric distribution, however, not all models were able to capture these adequately and results were often unstable. The use of GANs for VaR is in line with historical VaR techniques however the additional effort needed to train a GAN model seems wasted when there is no associated out-performance.

The outline of this dissertation is as follows - in Section 2, the GAN framework will be introduced and the various GAN variants explored. In Section 3, 1D distributions produced by the trained GAN models will be investigated with Section 4 moving into the VaR framework and how these 1D distributions may be used to enhance historical VaR models.

2 Generative Adversarial Networks

In this section, we introduce implicit generative modelling as context for GANs. Additionally, the basics of the GAN framework are explained and several GAN objective functions are presented in Section 2.2. The original GANs [6], namely the NS GAN and the min-max GAN (MM GAN), and three variants, the WGAN [2], WGAN GP [8] and the MMD GAN [14], are presented to showcase the differences in GAN variants. Finally, the details of GAN training and the problems associated with it are covered in Section 2.3.

2.1 GANs as Implicit Generative Models

Generative models lie at the heart of machine learning and statistics. They describe probabilistic distributions of data and form part of the unsupervised learning category in the machine learning taxonomy. These models can be broken down into two sub-categories - explicit and implicit generative models (IGMs). Explicit generative models aim to model the probability density of the underlying data. These generative models have been widely studied and are commonplace in statistics. A simple example of an explicit generative model would be modelling the outcome of a coin toss via a Bernoulli distribution.

In many applications, however, we are interested in sampling from the distribution instead of measuring the density. This is where IGMs are appropriate. IGMs aim to model transformations between a known source distribution and a target distribution - this aims to simulate a sampling process without making any assumptions about the density of the underlying data. This is comparable to the inverse cumulative distribution function (CDF) transform method which is used in Monte Carlo [20]. This method generates data samples by using random latent variables and an inverse CDF to generate synthetic data samples.

To gain more intuition on this concept, if we take the same example of a coin being tossed there are multiple sources of randomness that will influence the outcome (i.e., the environment, the angle of throw, etc.). The physical world takes these sources of randomness and maps them to an outcome by following the laws of physics. In a similar vein, or by abstract analogy, the IGM simulates the sampling by using a transform to map randomness to the target distribution. We can go on to define the output of this IGM.

Definition 2.1 (IGM Output)**.** Given a finite sample from a target distribution $\{x_i\}_{i=1}^n$, where $x_i \in X$ and $x_i \sim P_x$, an IGM will train a generator G_θ, which is parameterised by θ, to transform noise samples, $\{z_i\}_{i=1}^n$, from a base distribution P_z to the model distribution Q so that Q approximates P_x. The IGM's output is

defined by

$$\hat{x} = G_\theta(z).$$

IGMs do not explicitly represent the distribution of the data, only the transformation, therefore these models cannot answer other queries such as what is the probability of a particular observation.

The architecture of IGM is as follows: the noise vector z is sampled from a fixed distribution P_z. Most applications choose P_z to be a known parametric distribution, usually a Gaussian or a Uniform distribution. This vector is then passed as input to a deterministic generator network G_θ, which produces an output sample $\hat{x} = G_\theta(z)$. Schematically, this process is shown in Figure 1.

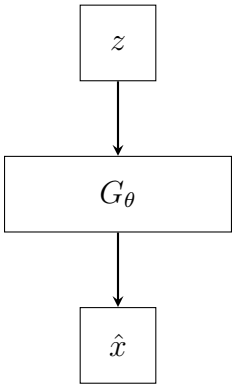


Figure 1: A simplistic schematic diagram of an IGM. The process diagram demonstrates how the latent vector z is transformed by the generative model G_θ to $\hat{x}$.

One type of IGM is a GAN. The framework originated in the landmark paper by Goodfellow et al. in [6] and is the focus of this dissertation. In essence, a GAN is made up of two adversarial components, a generator, and a discriminator. The generator learns a generative model through the guidance of the discriminative model whose output indicates how real or fake the data sample is. A generic GAN schematic is shown in Figure 2.

In the original formulation of the problem in [6], these components work against each other and have competing objectives, hence the term 'adversarial'. This formulation can be viewed as a two-player game where the generator learns to transform the input z into a sample that's distributed according to Q which approximates the target distribution P_x so that the discriminator cannot tell which distribution the samples belong to.

Following the original GAN paper, there has been a vast amount of work published on modifying and improving the GAN framework which has resulted in hundreds of

GAN variants. The proposed GAN taxonomy, presented in [22], divides GANs into two main categories, namely architecture variants and loss variants. Architecture variants have variations in the network architecture, the latent space, or are application specific whilst loss variants have different loss functions to be optimised or have additional penalisation added to the loss function.

For this dissertation, popular loss variants are reviewed and a selection are trained to produce 1D synthetic data. The classic NS GAN [6], the MM GAN [5], the WGAN [2], the WGAN GP [8] and the MMD GAN [14] are discussed in Section 2.2.

2.2 GAN Objective Functions

GAN objective functions are criteria that allow us to find the neural network parameters, for both the generator and the discriminator, such that the transformed latent data $\hat{x}$ converges to x in distribution. GANs are usually trained using simultaneous or alternating variations of gradient descent and their convergence is dependent on their objective function. Research presented in [16] shows how common GAN variants converge and under what conditions. An appropriate criterion for an objective function would be a measure of how well the generated distribution Q approximates the target distribution P_x. Game-theoretic approaches, presented in the original GAN paper [6], have been shown to approximate distance minimisation between P_x and Q, whilst the GAN objectives presented in Sections 2.2.3 to 2.2.5 minimise the distance between P_x and Q directly.

GAN loss variants have different objective functions but the same overarching framework. The generic GAN framework is depicted in Figure 2.

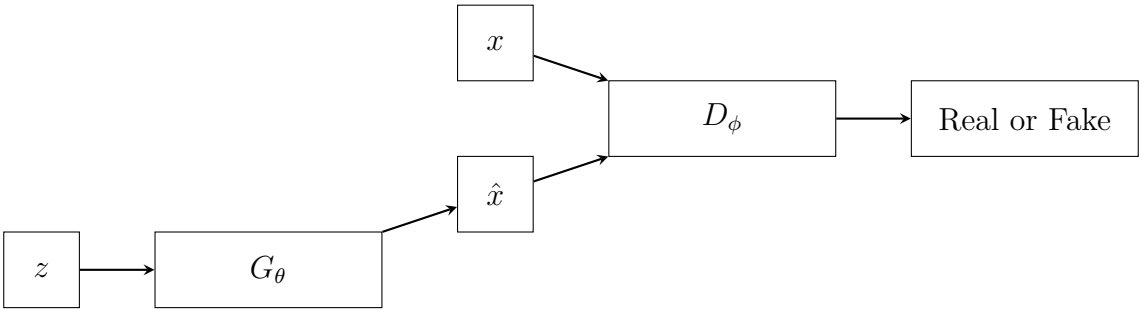


Figure 2: A simplistic schematic diagram of a generic GAN. The process diagram demonstrates how the latent vector z is transformed by the generator model G_θ to $\hat{x}$ and how the discriminator model D_ϕ outputs a prediction or metric for how real or fake the sample is.

For some additional context, the essence of most machine learning algorithms is

to build an optimisation model and learn the parameters in the objective function from the given data. Optimisation is the problem of finding a set of parameters for an objective function that results in a maximum or minimum function evaluation. An objective function is either a loss function (which is minimised) or its opposite (in specific domains, it can be called a reward function, a profit function, a utility function, etc.), in which case it has to be maximised. In most cases, by adding a negative to a reward function, the corresponding loss function can be found.

2.2.1 Min-Max Loss

In the classical GAN framework, the generator transforms a noise sample z into $\hat{x}$ using $G_\theta(z)$. The discriminator D_ϕ, parameterised by ϕ, is a binary classifier that tries to distinguish between generated samples $\hat{x}$ and the real data samples x. This process is shown in Figure 2. The MM GAN neural nets are adversaries. The point of convergence for this GAN is when the discriminator can no longer distinguish between samples from the target distribution and samples from the generated distribution. Since the discriminator is a binary classifier, this is the point where it would assign equal probability that the sample came from the real distribution than from the approximated distribution.

The discriminator loss function is obtained from the binary-cross-entropy (BCE) formula[1] and is derived as follows:

$$L^{(D)}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{1}$$

For a real sample x, the true data label y is set to one and the predicted label $\hat{y}$ is the discriminator output, namely $\hat{y} = D_\phi(x)$. The loss function in Equation 1 reduces to:

$$L^{(D)}(D_\phi(x), 1) = -\log(D_\phi(x))$$

For a sample from the generator, $\hat{x}$, the true data label y is set to zero and the predicted label $\hat{y}$ is set to $\hat{y} = D_\phi(\hat{x})$. The loss function in Equation 1 reduces to:

$$L^{(D)}(D_\phi(G_\theta(z)), 0) = -\log(1 - D_\phi(G_\theta(z)))$$

Reverting back to the idea that the generator and discriminator are adversaries, the discriminator has this competing objective to the generator, which is indicated by the opposite sign. When training the generator, only samples from the generator

[1] All logarithms are natural

are fed into the discriminator for feedback so the objective of the generator reduces to

$$L^{(G)}(D_\phi(G_\theta(z)), 0) = \log(1 - D_\phi(G_\theta(z))). \tag{2}$$

The above loss functions capture the difference between the actual and predicted values for a single output whereas cost functions aggregate the difference for the entire training dataset. The cost functions for the MM GAN generator and discriminator are defined as

$$J^{(D)}(D_\phi, G_\theta(z)) = -E_{P_x}[\log(D_\phi(x))] - E_{P_z}[\log(1 - D_\phi(G_\theta(z)))]$$

$$J^{(G)}(D_\phi, G_\theta(z)) = E_{P_z}[\log(1 - D_\phi(G_\theta(z)))].$$

When combining the two component cost functions, the GAN training objective function is derived. The GAN optimisation problem is defined as

$$\min_\theta \max_\phi V(G_\theta, D_\phi) = E_{P_x}[\log(D_\phi(x))] + E_{P_z}[\log(1 - D_\phi(G_\theta(z)))]. \tag{3}$$

In practice, individual cost functions are defined for both the generator and the discriminator. If both the generator and the discriminator have sufficient capacity, the GAN converges where $G_\theta(z)$ is being drawn from the same distribution as x, so that $D(x) = \frac{1}{2}$ for all x (i.e., the discriminator assigns equal probability that the sample comes from the real distribution than from the approximated distribution).

The original paper [6] completed theoretical investigations into Equation 3. They show that the min-max formulation of the game can be reformulated as minimising the scaled and shifted Jensen-Shannon divergence between the real and generated probability distributions.

2.2.2 Non-Saturating Loss

The NS GAN is the second GAN introduced in the original GAN paper by Ian Goodfellow et al [6]. This GAN is strategically equivalent to the game underlying the MM GAN, this was shown in work presented in [4], as the Nash equilibrium, and hence solution set, is the same. The reason this GAN was introduced was to solve a weakness in MM GAN training, which is termed the vanishing gradient problem. This is a problem in the early stages of training where the generator performs poorly since the discriminator can easily distinguish between real and generated samples, with high confidence. This means that gradients are calculated at points where the sigmoid function in the discriminator loss, due to the nature of the binary classifier,

is very flat so the gradients are very small and will have negligible effect when used to update the discriminator in gradient descent. To remedy this problem, the authors suggest changing the cost function of the generator from minimising the log probability of the discriminator being correct to maximising the log probability of the discriminator being wrong. This means labelling samples from the generator as real (i.e., $y = 1$) in its cost function. The updated cost function for the generator is:

$$J^{(G)}(G_\theta, D_\phi)) = E_{P_z}[logD_\phi(G_\theta(z))],$$

while the cost function of the discriminator remains the same:

$$J^{(D)}(G_\theta, D_\phi)) = -E_{P_x}[logD_\phi(x)] - E_{P_z}[log(1 - D_\phi(G_\theta(z)))].$$

This game is heuristically motivated and ensures that each player has a strong gradient when they are 'losing' the game, as shown in [5].

2.2.3 Wasserstein Loss

NS and MM GANs are notoriously difficult to train and are often very unstable. As mentioned in subsection 2.2.1, when the discriminator is trained to optimality, the MM GAN generator tries to minimise the Jensen-Shannon divergence between the generated and target distributions. This divergence metric has been identified as a potential reason for instability during training as it is constant for non-overlapping distributions, meaning it provides no useful feedback during training. One way this problem was addressed was by using Integral Probability Metrics (IPMs). An IPM is a way to measure the distance between two probability distributions by the largest difference in their expectations over a certain class of witness functions. In this context, a witness function is a function that is used to construct a discrepancy measure between two probability distributions. The concept of witness functions was introduced by Arthur Gretton and co-authors in paper [7].

Definition 2.2 (IPM)**.** For a class of functions $\mathcal{F}$, with $X \sim P_x$ and $\hat{X} \sim Q$, the IPM is defined as

$$d_\mathcal{F}(P, Q) = \sup_{f \in \mathcal{F}}[E_{P_x} f(X) - E_Q f(\hat{X})],$$

where sup is the supremum or upper bound. The witness function f is a function that achieves the maximum difference between the expectations under the two distributions. The particular witness function class determines the probability metric.

A popular IPM used in GAN training is the Wasserstein distance. Replacing the discriminator loss function of the NS GAN with Wasserstein loss results in the

WGAN GAN variant. The WGAN was introduced by Martin Arjovsky et al in paper [2]. Due to the adjustment to the discriminator loss function, the authors refer to the discriminator of the WGAN as a critic.

The objective function of the WGAN has better theoretical properties than the original MM and NS GANs. The theoretical property of interest is the continuity of the divergence metric. In [2], the original WGAN paper, the authors illustrate that when compared to Jensen-Shannon divergence and Kullback–Leibler (KL) divergence, only the Wasserstein metric provides a smooth measure of distance between two non-overlapping distributions. This smoothness is helpful for a stable learning process using gradient descent, which is the mechanism used for parameter estimation in GANs.

In WGAN, BCE loss is replaced by the Wasserstein loss metric. This metric uses the Wasserstein distance, which measures the distance between the real and generated distributions. It is, in the continuous domain, the percentage of 'mass' that has to be transported from point $\hat{x}$ to point x to make $\hat{x}$ follow the distribution of x using the optimal transport plan. In other words, it can be interpreted as the minimum energy cost of making the generated distribution equal to the real distribution.

The Wasserstein distance, not yet in IPM form, in the continuous probability domain is

$$W(P_x, Q) = \inf_{\gamma \sim \Pi(P_x, Q)} E_{(\hat{x},x)\sim\gamma} \|\hat{x} - x\| \tag{4}$$

In Equation 4, $\Pi(P_x, Q)$ is the set of all possible joint probability distributions between the real and generated probability distributions (P_x and Q). The $\gamma \in \Pi(P_x, Q)$, is a transport plan to make P_x equal Q in the continuous probability space. The percentage of mass that should be transported from point $\hat{x}$ to x to make Q equal to P_x is $\gamma(\hat{x}, x)$. If we think of $\hat{x}$ as the starting point and x as the destination, the total amount of dirt moved is $\gamma(\hat{x}, x)$ and the distance is $\|\hat{x} - x\|$. The cost of the transport plan is therefore $\gamma(\hat{x}, x)\|\hat{x} - x\|$. The expected average cost across all $(\hat{x}, x)$ pairs is $\sum_{\hat{x},x} \gamma(\hat{x}, x)\|\hat{x} - x\|$, which is equal to $E_{(\hat{x},x)\sim\gamma}\|\hat{x} - x\|$. The greatest lower bound, or infimum, of the costs is taken as the final Wasserstein distance.

The metric given in Equation 4 cannot directly be applied in a GAN setting as it requires minimisation over a set of joint probability measures. This is hard to parameterise by neural networks. To be able to interpret this distance as an IPM, the authors of paper [2] proposed a transformation of Equation 4 based on the Kantorovich-Rubinstein Duality to

$$W(P_x, Q) = \sup_{\|f\|_L \leq K} [E_{P_x} D_\phi(x) - E_Q D_\phi(\hat{x})], \tag{5}$$

with $\|f\|_L := \frac{|f(x_1)-f(x_2)|}{\|x_1-x_2\|}$ denoting the Lipschitz constant of a function f. This, intuitively, is the highest difference, on average, of any function f between real and generated inputs as long as the function is a K-Lipschitz function. We find this K-Lipschitz function, $f := D_\phi$, by parametrizing it through a deep neural network. Equation 5 can be interpreted as an IPM, as per Definition 2.2, constructed from a witness class of K-Lipschitz functions.

Practical implementation of the WGAN, when compared to the NS GAN, involves replacing the label y from either 0 or 1, for real or fake, with -1 or 1. The sigmoid activation is also removed which means the discriminator's output ranges from $(-\infty, \infty)$ instead of being bound between 0 and 1. The discriminator is trained to maximise the difference in scores between real and fake data samples with real samples scoring highly. The Wasserstein loss is the negative product of the sample label y and the discriminator output $\hat{y}$ and is defined as

$$L^{(D)}(y, \hat{y}) = -y\hat{y},$$

which reduces to

$$L^{(D)}(y, \hat{y}) = D_\phi(x) - D_\phi G_\theta(z). \tag{6}$$

The optimisation for the WGAN discriminator is achieved by minimising cost function

$$J^{(D)}(G_\theta, D_\phi) = -E_{P_x}[D_\phi(x)] - E_{P_z}[D_\phi(G_\theta(z))].$$

The generator is trained by comparing generated data $D_\phi(G(z)$ to label $y = 1$ and optimisation is occurs by minimising:

$$J^{(G)}(G_\theta, D_\phi) = -E_{P_z}[D_\phi(G_\theta(z))]. \tag{7}$$

The WGAN optimisation problem, relating back to Equation 5, can therefore be defined as

$$\min_\theta \max_{||\phi||_L<1} V(G_\theta, D_\phi) = E_{P_x}[D_\phi(x)] - E_{P_z}[D_\phi(G_\theta(z))]. \tag{8}$$

Since the discriminator can produce outputs in the range $(-\infty, \infty)$, to ensure that the loss function is continuous and differentiable and that it doesn't grow too much during training, a K-Lipschitz continuity is enforced. This is the condition in Equation 8 on the maximisation of ϕ. The K-Lipschitz continuity between two points, x_1 and x_2, is defined as:

$$\left| \frac{D_\phi(x_1) - D_\phi(x_2)}{x_1 - x_2} \right| \leq K.$$

This limits the function slope to some constant K along its trajectory. In the original WGAN paper, K-Lipschitz continuity is enforced using weight-clipping. This simple enforcement means that after every gradient update, the weights of the discriminator are clamped to a small window, such as $[-0.01, 0.01]$. The discriminator's overall gradient in the WGAN paper is thus limited by separately limiting, or clipping, each of the weights in the discriminator - this means it obtains its lower and upper bounds to preserve the Lipschitz continuity. This was noted in paper [2] as a 'terrible' solution. Work presented in a paper by Ishaan Gulrajani [8] showed that model performance is very sensitive to the weight-clipping hyperparameter and that weight-clipping ignores higher moments of target data distributions. In response to this, WGAN GP was introduced in paper [8] as an alternative.

2.2.4 Wasserstein Loss with Gradient Penalty

The WGAN GP variant uses gradient penalty instead of weight-clipping to encourage the K-Lipschitz constraint. The cost function of the original WGAN discriminator, $J^{(D)}_{WGAN}$, is modified to include a gradient penalty. The gradient penalty is calculated using points that are linearly interpolated between random samples from P_x and Q, with interpolation constant ϵ. Interpolated points c are calculated by

$$c(x, G_\phi(z)) = \epsilon x + (1 - \epsilon) G_\phi(z).$$

The gradient penalty keeps the L^2-norm of the discriminator's gradient close to one and the gradient penalty term λ is a hyperparameter choice for GAN training.

The WGAN GP discriminator cost function is

$$J^{(D)}(G_\theta, D_\phi) = J^{(D)}_{WGAN} + \lambda E[(\|\nabla D_\phi(c)\|_2 - 1)^2].$$

The symbol used to represent the gradient is ∇ and $\|(.)\|_2$ is the L^2-norm. The generator cost function remains the same as in Equation 7.

2.2.5 Maximum Mean Discrepancy Loss

Another IPM used in GAN variants is maximum mean discrepancy (MMD). The MMD GAN is a loss variant that uses kernel mean embeddings (KME) to signal the proximity between the probability density function of the generated distribution Q and the target distribution P_x. The MMD GAN was first introduced in 2017 by Chun-Liang Li in [14] and further investigated in 2018 by Arthur Gretton and team in [3].

Like the WGAN, the proposed benefit of the MMD GAN over the NS and MM GANs is that it is able to provide useful gradients for learning, even if the generated and target distributions are non-overlapping. This means that the training is more stable. In Li's work in [14], it was shown that the WGAN is a special case of the MMD GAN under certain conditions. In Gretton's work in [3], the MMD GAN has the proposed advantage that it can train with smaller mini-batch sizes.

The MMD is a metric on the space of probability distributions and is defined in terms of function spaces that witness the difference in distributions. In Gretton's work in [7], the definition of a MMD for an arbitrary function space is defined as follows:

Definition 2.3 (MMD)**.** Let $(\mathbb{X}, d)$ be a metric space and let $\mathcal{F}$ be a class of functions f: $\mathbb{X} \to$ R, with x, $\hat{x}$, P_x and Q defined as above. Gretton et al define the MMD as

$$\text{MMD}(P_x, Q; \mathcal{F}) := \sup_{f \in \mathcal{F}} (E_{P_x}[f(x)] - E_Q[f(\hat{x})]).$$

In statistical literature, this is an IPM as defined in 2.2.

For the MMD GAN, the witness class of functions $\mathcal{F}$ is a unit ball in a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, with a positive definite kernel $k(x, x')$. A Hilbert space is a mathematical concept that is used to describe a space of functions that can be added and multiplied in a way that preserves some key properties, such as the norm and the inner product. A RKHS is a Hilbert space of functions that has a kernel function that satisfies certain properties, such as positive definiteness. The reproducing property of an RKHS states that for any function f in the RKHS $\mathcal{H}$ and any point x in the underlying domain, the value of f at x can be recovered by evaluating the inner product of f with the kernel function evaluated at x. More formally:

$$f(x) = \langle f, k(x, \cdot) \rangle_{\mathcal{H}}.$$

Here, $k(x, \cdot)$ denotes the kernel function evaluated at x with its first argument fixed, and the angle brackets denote the inner product in the RKHS. This allows one to compute the value of a function at any point in the domain using only the inner product between the function and the kernel function, which can be much more efficient than directly evaluating the function. The reproducing property can also be used to define various operations in the RKHS, such as differentiation and integration.

The RKHS allows MMD GANs to use mean embeddings to signal proximity between distributions. Mean embedding is a technique used to transform a probability distribution into a fixed dimensional vector in a RKHS. We can define the mean embedding of the probability measure P_x as the element $\mu_{P_x} \in \mathcal{H}$ such that

$$E_{P_x} f(X) = \langle f, \mu_{P_x} \rangle_{\mathcal{H}}$$

with $\mu_{P_x} = E_{P_x}[k(\cdot, X)]$.

Given two distributions P_x and Q, and letting $k(x, \hat{x}) = \langle \varphi(x), \varphi(\hat{x}) \rangle_{\mathcal{H}}$, the MMD is defined as a IPM, following Definition 2.2 and 2.3 with the class of witness functions $\mathcal{F}$ being the unit ball in $\mathcal{H}$

$$\text{MMD}(P_x, Q; \mathcal{H}) = \sup_{f \in \mathcal{H}, ||f||_{\mathcal{H}} \leq 1} E_{P_x} f(X) - E_Q f(\hat{X}). \tag{9}$$

In the MMD GAN framework, empirical KMEs are used as we only have data samples and not distributions. The empirical witness function $\hat{f}$ that attains the supremum, shown in paper [7], is

$$\hat{f}(\cdot) = \frac{1}{m} \sum_{i=1}^{m} k(x_i, \cdot) - \frac{1}{n} \sum_{i=1}^{n} k(\hat{x}_i, \cdot)$$

with samples $X = \{x_i\}_{i=1}^m$ drawn from P_x, and $\hat{X} = \{\hat{x}\}_{j=1}^n$ from Q.

When using a characteristic kernel to create embeddings of the distributions, each distribution can be uniquely represented in $\mathcal{H}$ so that all of the statistical features are preserved. A kernel is characteristic if the mean embedding is injective which means that for the function f there exists a function f^{-1} such that $f^{-1}(f(x)) = x$. When using a characteristic kernel in MMD, the MMD is only equal to zero when P_x is equal to Q. A popular characteristic kernel used is the Gaussian kernel. Li's work in paper [3] uses the Gaussian kernel, which is part of the Radial Basis Function (RBF) class, to form part of the discriminator loss function. Gretton's work in paper [14] uses the rational quadratic kernel as the authors note the Gaussian kernel, and its derivatives, decay exponentially which can cause problems in gradient-based learning in high dimensions. Mathematically, the Gaussian kernel is defined as

$$k(x, \hat{x}) = exp\left(-\frac{||x - \hat{x}||^2}{2\sigma^2}\right).$$

with $\sigma > 0$ as the bandwidth parameter. Li, in paper [14], uses a linear combination of Gaussian kernels, with five different bandwidth values, in their application, i.e.,

$$K = \sum_{i=1}^{5} k_{\sigma_i}$$

with $\sigma_i \in \{1, 2, 4, 8, 16\}$. This is the strategy employed in this dissertation. Wan et al in their work in [21], propose this is done to prevent saturation when the distance $||x - \hat{x}||$ is either too large or too small compared to the bandwidth parameter, which may cause diminishing gradients during training.

A key strategy in recent work on MMD GANs is that the MMD is not computed directly on the samples, rather the samples first pass through a mapping function φ, which is generally a convolutional network. Intuitively we can think of this as the MMD with kernel k on features $\varphi(x)$. This relates to Li's work in [14] which encourages the adversarially learned kernel to be injective by approximating f using an autoencoder. An auto-encoder consists of two neural nets, one encoder D_{ϕ_e} (which maps the sample to a feature space) and one decoder, D_{ϕ_d}, that reconstructs the sample back to the original space. The encoder D_{ϕ_e} can be thought of as the mapping function φ. The decoder is trained to regularise f so that is it injective, namely $D_{\phi_d} \approx f^{-1}$.

The optimisation problem for the MMD GAN is:

$$\min_{\theta} \max_{\phi} \hat{\text{MMD}}(D_{\phi_e}(x), D_{\phi_e}(G_\theta(z)) - \lambda E_{y \in X \cup G_\theta(z)} ||y - D_{\phi_d}(D_{\phi_e}(y))||^2$$

with $\lambda E_{y \in X \cup G_\theta(z)} ||y - D_{\phi_d}(D_{\phi_e}(y))||^2$ being the penalty assigned to the discriminator for the reconstruction error of any sample from the real or generated distributions. The reconstruction error is the difference between the original sample and one that has been fed through the auto-encoder network. This encourages the discriminator function to be injective.

2.3 GAN Training

Training GANs is notoriously difficult and finding a good model often requires trying out various hyperparameter configurations by brute force. This subsection will outline how GANs are trained in practice and highlights the common problems that can occur during training.

2.3.1 Algorithmic Implementation

The neural networks in GANs are trained using mini-batch gradient descent (MBGD). MBGD is an iterative optimisation algorithm that is used to find the minimum of a differentiable function. For MBGD, the training dataset is split into small subsets (mini-batches). In each iteration, a mini-batch is passed through the neural network to compute the value of the cost function. The gradients of the cost function with respect to the parameters of the network are then calculated and used to update the parameters of the network accordingly. Once all the mini-batches have been fed through the network, one epoch has passed. Usually, multiple epochs are used to train a neural network.

MBGD is a way to minimise an objective function by updating the parameters of the network in the opposite direction of the gradient, ∇, of the cost function. The

learning rate α determines the size of the steps we take to reach a local minimum and n is a hyperparameter that denotes the size of a single mini-batch.

The formula of MBGD, with mini-batch size n and iteration i, that updates the weights w is:

$$w_{i+1} = w_i - \alpha \nabla_{w_i} J(D(x^{i:i+n}), y^{i:i+n})$$

Tables 1 and 2 give a summary of the cost functions of the GAN variants that were discussed in subsection 2.2.

Table 1: Discriminator cost functions. Each cost function has been modified so that optimisation occurs by minimising the cost.

GAN Type	Discriminator Cost $J^{(D)}$
MM	$-E_{P_x}[logD_\phi(x)] - E_{P_z}[log(1 - D_\phi(G_\theta(z)))]$
NS	$-E_{P_x}[logD_\phi(x)] - E_{P_z}[log(1 - D_\phi(G_\theta(z)))]$
WGAN	$-E_{P_x}[D_\phi(x)] + E_{P_z}[D_\phi(G_\theta(z))]$
WGAN GP	$J^D_{WGAN} + \lambda E_{P_z}[(\|\nabla_{\tilde{x}} D(\tilde{x})\|_2 - 1)^2]$
MMD	$-[\mathrm{MMD}^2(P_x, Q) + \lambda E_{y \in X \cup G_\theta(z)} \|y - D_{\phi_d}(D_{\phi_e}(y))\|^2]$

Table 2: Generator cost functions. Each cost function has been modified so that optimisation occurs by minimising the cost.

GAN Type	Generator Cost $J^{(G)}$
MM	$E_{P_z}[log(1 - D_\phi(G_\theta(z)))]$
NS	$-E_{P_z}[logD_\phi(G_\theta(z))]$
WGAN	$-E_{P_z}[D_\phi(G_\theta(z))]$
WGAN GP	$-E_{P_z}[D_\phi(G_\theta(z))]$
MMD	$\mathrm{MMD}^2(P_x, Q)$

In each cost function, the expected values are approximated by their empirical counterparts based on mini-batches. In general:

$$E(x) = \sum_{i=1}^{n} \frac{x_i}{n}.$$

Algorithm 1 summarises the training procedure for NS GAN, MM GAN, WGAN, WGAN GP and MMD GAN.

Algorithm 1 MBGD training implementation for selected GAN variants (Adapted from [6], [2], [8], [14]). hyperparameters are the learning rate α, the mini-batch size n, the number of discriminator updates per generator update k, the clipping parameter c (for WGAN only).

for number of epochs **do**

for steps 1 to k **do**

1. Sample a mini-batch of real data $\{x_i\}_{i=1}^{n}$ from P_x.
2. Sample a mini-batch of latent data $\{z_j\}_{j=1}^{n}$ from P_z.
3. Update the discriminator by gradient descent: NS GAN and MM GAN:

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} J_{MM}^{(D)}$$

WGAN:

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} J_{WGAN}^{(D)}$$

4. Clip Gradients (only for WGAN)

$$\phi \leftarrow \text{clip}(\phi, -c, c)$$

WGAN GP

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} J_{WGANGP}^{(D)}$$

MMD GAN

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} J_{MMD}^{(D)}$$

end for

1. Sample a mini-batch of latent data $\{z_j\}_{j=1}^{n}$ from P_z.
2. Update the generator by gradient descent: MM GAN

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_{MM}^{(G)}$$

NS GAN

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_{NS}^{(G)}$$

WGAN and WGAN GP

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_{WGAN}^{(G)}$$

MMD GAN

3. Sample a mini-batch of noise $\{x_i\}_{j=1}^{n}$ from P_x.

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} J_{MMD}^{(G)}$$

end for

2.3.2 Overview of Common Problems

Although GANs are capable of producing realistic synthetic samples, training GANs is notoriously difficult. A brief summary of the main difficulties is highlighted below.

Partial Mode Collapse: This problem occurs when the GAN learns to map several inputs in the latent space to the same output. This means that the generator ends up producing samples with limited diversity. This can occur when the discriminator is able to produce realistic output for a few modes during training as it is incentivized by the generator cost function to restrict the output to these modes instead of improving the quality of samples produced from other modes.

The choice of discriminator cost function has been put forward as a heuristic solution to improve partial mode collapse, although Ian Goodfellow in [5] has disputed if the choice in cost function would be the root cause of this problem.

Vanishing Gradients: This problem prevents beneficial training. It occurs when the gradient updates for the generator become very small, and hence are not useful in training. The original MM GAN implementation in subsection 2.2.1 struggles with this problem, this is why the NS GAN in subsection 2.2.2 is implemented in practice.

Non-Convergence: Local convergence of objective functions is mostly only guaranteed under certain conditions, this is investigated in [16]. If these are not met, which is often the case in the GAN setting, then oscillations may occur in training and the equilibrium may not be reached. Work has been done in [16] to discuss conditions under which some of the most popular GANs converge locally.

Evaluation of Generative Models: Evaluation of the generated data from GANs can be challenging. Suggested metrics for evaluation often are application and domain dependent. A large volume of research has been dedicated to investigating evaluation metrics for natural images, such as in [25] where popular evaluation metrics are assessed on image datasets for interpretability, however for applications in lower dimensions these metrics are less appropriate.

Hyperparameter Tuning: Hyperparameter choices and tuning play a huge role in the model fit. Through experimentation, it was seen that even slight variations in choices had a huge effect on outputs. Grid-search methods are recommended for future iterations of the research. Brute force methods are time-consuming and inefficient.

3 Learning One-Dimensional Distributions with GANs

In this section, we train the different GAN models to sample from a chosen parametric distribution. The NS GAN, the MMD GAN and the WGAN are trained to produce three parametric distributions, namely the normal distribution and two Gaussian Mixture Models (GMMs). These generated 1D distributions will be used in Section 4 to supplement and potentially enhance historical VaR models. In subsection 3.1 the candidate GAN architecture choices are presented. In subsection 3.2 the training procedure implemented is outlined. In subsection 3.3 the result metrics are described and in 3.4 the results are presented and discussed.

3.1 GAN Hyperparameter Choices

Due to the multiple hyperparameter choices, and their impact on training stability, the tuning procedure was lengthy. The range of choices is presented below.

3.1.1 GAN Loss Variant

Three GANs were evaluated for this dissertation, namely the NS GAN, the WGAN, and the MMD GAN. The MM GAN was not practically implemented as the NS GAN, presented in the same paper, is the recommended variant with very few papers choosing to implement the MM GAN. Various network architectures for each discriminator and generator were compared. The number of hidden layers and the number of neurons in each layer were varied, this is detailed in 3.1.3.

3.1.2 Latent Variable Distribution and Batch Size

The model fit is sensitive to both the latent distribution as well as the batch size of the latent vector $\boldsymbol{z}$. In paper [19], the authors analyse different parametric distributions for the GANs' latent space and propose a way to obtain suitable non-parametric latent distributions. In practice, the latent space is usually sampled from a Gaussian or Uniform distribution. The batch size of the latent space, and its impact on GAN performance, is rarely discussed and is somewhat arbitrarily chosen. There is, however, a preference to use latent variables with higher dimensions.

During training, to produce a sample, the generator was fed latent vector $\boldsymbol{z}$, with a batch size of n. This corresponded to the number of neurons in the input layer of the generator. For this dissertation $n = \{1, 10, 20\}$ was tested. For the latent variable distribution, a standard normal distribution was chosen and compared to a uniform distribution.

3.1.3 Network Architecture

Both the generator and discriminator were chosen to be standard fully-connected feed-forward networks. Both the number of layers as well as the number of hidden units were varied to find the best model fit. Networks with two, three and four hidden layers were considered. A huge variety of unit numbers were tested, leveraging prior research done in [4] and [26]. Tables 3 and 4 summarise the network architectures for the more successful models. The number of neurons in the input layer of the generator corresponds to the batch size n of the latent vector. Figure 3 gives an example of the network architecture for the encoder D_e.

Table 3: Generator architecture hyperparameters

Hyperparameter	G^1	G^2	G^3
Number of layers	6	5	3
Number of neurons	(n,128,248,496,992,1)	(n,128,128,128,1)	(n,7,13,7,1)
Activation Function	ReLU	LeakyReLU	ELU
Reference Paper	NA	[4]	[26]

Table 4: Discriminator architecture hyperparameters

Hyperparameter	D^1	D^2	D_e	D_d
Number of layers	5	5	3	3
Number of neurons	(1,128,128,128,1)	(1,7,13,7,1)	(1,11,29)	(29,11,1)
Activation Function	LeakyReLU	LeakyReLU	ELU	ELU
Reference Paper	[4]	NA	[26]	[26]

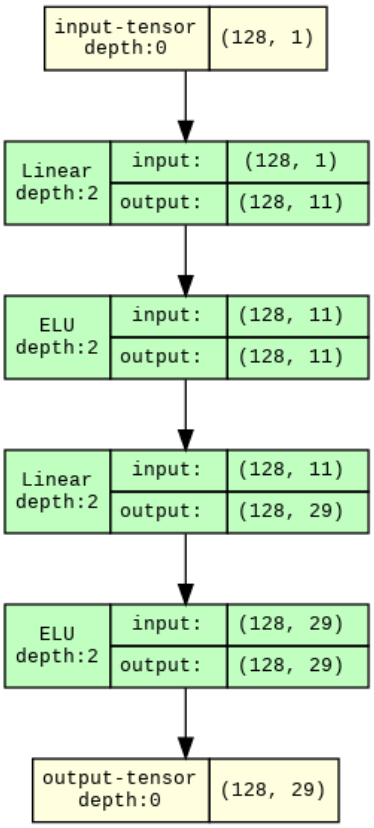


Figure 3: Encoder network architecture with a batch-size of 128.

3.1.4 Activation Functions

The Rectified Linear Unit (ReLU), Leaky ReLU, suggested in [4], and the Exponential Linear Unit (ELU) activation functions were tested as suggested in [26].

Definition 3.1 (ReLU)**.** For $x \in (-\infty, \infty)$, the ReLU function is defined as:

$$\text{ReLU}(x) = \max(0, x).$$

Definition 3.2 (LeakyReLU)**.** For $x \in (-\infty, \infty)$, the LeakyReLU function is defined as:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha x & \text{otherwise.} \end{cases}$$

with $\alpha \in [0, \infty)$ as a hyperparameter.

Definition 3.3 (ELU)**.** For $x \in (-\infty, \infty)$, the ELU activation function is defined as:

$$\text{ELU}(x) = \begin{cases} x & \text{if } x > 0, \\ \alpha(\exp(x) - 1) & \text{otherwise.} \end{cases}$$

with $\alpha \in [0, \infty)$ as a hyperparameter.

3.1.5 Optimiser

The ADAM optimiser was used for both the generator and the discriminator networks. The learning rate was experimented with during training and varied between 0.01 and 0.0001. RMSProp was another suggestion in the work presented by authors in [2] for WGAN and WGAN GP implementations.

3.1.6 Generator and Discriminator Update Ratio

A static ratio between the number of gradient descent iterations of the discriminator and the generator is recommended. The number of discriminator updates per generator update was fixed at five as suggested in [2]. In general, this means that the discriminator training step always has 5 times more updates than the generator training step. This was varied in the initial stages however the recommended ratio gave satisfactory results.

This ratio stems from ideas put forward in the original paper [6], authors had the idea that one should balance the optimisation of these two networks in the min-max game and that when the discriminator is near optimality, the ratio between probability densities is more accurate, generating better gradients to update the generator.

3.2 Training Implementation

The practical implementation of training the NS GAN, WGAN and MMD GAN to produce the specified 1D parametric distribution is detailed for each GAN below. All code for this dissertation is in Python. The machine learning framework used is PyTorch [17].

For training, a $n \times k$ training matrix $\mathcal{D}$, made up of s samples from P_x, is created. The mini-batch size n and discriminator to generator update ratio k are hyperparameters. Other hyperparameters for GAN training are the learning rate α, the weight clipping parameter c (WGAN), the gradient penalty term λ (MMD GAN) and the Gaussian bandwidth parameter σ (MMD GAN). P_x is the target parametric distribution (i.e., Normal or GMM).

Table 5: Training hyperparameters selected from research

Hyperparameter	**Value**	**Reference**
α	0.001	[13]
σ	$\sigma_i \in \{1, 2, 4, 8, 16\}$	[14]
λ	8	[14]
k	5	[2]
Iterations	10,000	[13]
c	$[-0.01, 0.01]$	[2]

A summary of hyperparameter choices and resources for where they were suggested is included in Table 5.

Algorithm 2 Training implementation for generating parametric 1D distributions for the NS GAN.

for number of epochs **do**

for steps i=0 to k **do**

1. $\boldsymbol{x} = \mathcal{D}_{i:}$
2. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
3. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
4. Calculate the discriminator cost for the mini-batch: $J_{NS}^{(D)}$.
5. Update the weights of the discriminator using ADAM optimisation.

$$\phi \leftarrow \text{Adam}(\nabla_\phi J_{NS}^{(D)}, \phi, \alpha)$$

end for

1. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
2. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
3. $\hat{\boldsymbol{y}} = D_\phi(\hat{\boldsymbol{x}})$
4. Calculate generator cost $J_{NS}^{(G)}$ for the mini-batch.
5. Update the weights of the generator using ADAM.

$$\theta \leftarrow \text{Adam}(\nabla_\theta J_{NS}^{(G)}, \theta, \alpha)$$

end for

Algorithm 3 Training implementation for generating parametric 1D distributions for the WGAN.

for number of epochs **do**

for steps i=0 to k **do**

1. $\boldsymbol{x} = \mathcal{D}_{i:}$
2. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
3. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
4. Calculate the discriminator cost for the mini-batch: $J_{WGAN}^{(D)}$.
5. Update the weights of the discriminator using ADAM optimisation.

$$\phi \leftarrow \text{Adam}(\nabla_\phi J_{WGAN}^{(D)}, \phi, \alpha,)$$

6. Clip discriminator weights $\phi \leftarrow \text{clip}(\phi, -c, c)$

end for

1. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
2. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
3. $\hat{\boldsymbol{y}} = D_\phi(\hat{\boldsymbol{x}})$
4. Calculate generator cost $J_{WGAN}^{(G)}$ for the mini-batch.
5. Update the weights of the generator using ADAM.

$$\theta \leftarrow \text{Adam}(\nabla_\theta J_{WGAN}^{(G)}, \theta, \alpha)$$

end for

Algorithm 4 Training implementation for generating parametric 1D distributions for the MMD GAN.

for number of epochs **do**

for steps i=0 to k **do**

1. $\boldsymbol{x} = \mathcal{D}_{i:}$
2. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
3. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
4. Encode: $\boldsymbol{x_e} = D_{\phi_e}(\boldsymbol{x})$, $\hat{\boldsymbol{x}}_{\boldsymbol{e}} = D_{\phi_e}(\hat{\boldsymbol{x}})$
5. Decode: $\boldsymbol{x_d} = D_{\phi_d}(D_{\phi_e}(\boldsymbol{x}))$, $\hat{\boldsymbol{x}}_{\boldsymbol{d}} = D_{\phi_d}(D_{\phi_e}(\hat{\boldsymbol{x}}))$
6. Calculate the discriminator cost for the mini-batch: $J_{MMD}^{(D)}$.
7. Update the weights of the discriminator using ADAM optimisation.

$$\phi \leftarrow \text{Adam}(\nabla_\phi J_{MMD}^{(D)}, \phi, \alpha)$$

end for

1. $\boldsymbol{z} = \left\{z_j\right\}_{j=1}^{n}$ from P_z.
2. $\hat{\boldsymbol{x}} = G_\theta(\boldsymbol{z})$
3. Encode: $\boldsymbol{x_e} = D_{\phi_e}(\boldsymbol{x})$
4. Decode: $\hat{\boldsymbol{x}}_{\boldsymbol{e}} = D_{\phi_e}(\hat{\boldsymbol{x}})$
5. Calculate generator cost $J_{MMD}^{(G)}$ for the mini-batch.
6. Update the weights of the generator using ADAM.

$$\theta \leftarrow \text{Adam}(\nabla_\theta J_{MMD}^{(G)}, \theta, \alpha)$$

end for

3.3 Result Metrics

To assess the distributional similarity of the target and generated data samples, we can use statistical significance tests that can quantify the likelihood that the samples have the same distribution. Seeing that we want our testing to be valid for all target distributions, and not just Gaussian distributions, non-parametric significance tests are used. These tests are free from distributional assumptions. The null hypothesis of the selected tests is that both samples were drawn from a population with the same distribution. These tests return a p-value, which can be interpreted as the probability of observing the two data samples given the base assumption that the two samples were drawn from a population with the same distribution. The p-value can be interpreted in the context of a chosen significance level α. If the p-value is high, then we cannot reject the null hypothesis in favour of the alternative. In this dissertation, we choose to test the null hypothesis, that the two samples were

drawn from the same distribution, at a 95% confidence level. This means that if the p-value from a non-parametric test is less than 5%, we will reject the null hypothesis in favour of the alternative, which is that the samples were drawn from different distributions.

The non-parametric statistical significance tests chosen are the Kolmogorov-Smirnov (KS) two-sample test and the Cramer-von Mises two-sample test. The Wasserstein distance, discussed in subsection 2.2.3, as well as the distributional moment (mean, variance, skew, kurtosis) scores of the generated and target distributions are also assessed. A brief overview of the result metrics chosen to assess distributional similarity is outlined in this subsection.

3.3.1 Kolmogorov-Smirnov Two-Sample Test

The KS two-sample test is probably the most popular non-parametric test to compare distributions. The idea of the KS two-sample test is to compare the empirical cumulative distribution functions (ECDFs) of the two samples, which are step functions that approximate the cumulative density functions based on the sample data. A two-sided test is performed with the null hypothesis being that the two samples are drawn from the same distribution. The KS test statistic is the maximum absolute difference between the two ECDFs. The main advantage of the KS test is its sensitivity to the shape of a distribution because it can detect differences everywhere along the scale. KS tests have the disadvantage that they are more sensitive to deviations near the centre of the distribution than at the tails. The test statistic d is calculated as:

$$d = \max(F_{P_x}(x) - F_Q(x)),$$

where $F_{P_x}(.)$ and $F_Q(.)$ denote the cumulative distribution functions for random variables that are distributed according to P_x and Q, respectively.

3.3.2 Cramer-von Mises Two-Sample Test

The Cramer-von Mises (CVM) two-sample test is similar to the KS two-sample test as it also compares the ECDFs of the two samples, however, the CVM two-sample test compares the two distributions along the whole domain, by integration.

Given the two empirical cumulative density functions F_{P_x} and F_Q from two sample sets $\{x_1.., x_N\}$ and $\{\hat{x_1}.., \hat{x_M}\}$ the test statistic is calculated as:

$$d = \frac{NM}{(N+M)} \int_{-\infty}^{\infty} [F_{P_x}(x) - F_Q(x)]^2 \, dH_{N+M}(x),$$

where $H_{N+M}(x)$ is the empirical distribution of the two samples together.

3.3.3 Wasserstein Distance

The Wasserstein distance is used to assess distributional similarity. The SciPy function is used for computation. A detailed discussion on this metric can be found in subsection 2.2.3. Smaller values indicate better distributional similarity. Since there is no widely defined definition of a 'good' Wasserstein distance, this metric will be used as a relative score.

3.3.4 Distribution Moments

The statistical moments describe the properties of a distribution. The moments assessed are the mean, variance, skew, and kurtosis. These moments will be the basis of comparison between the generated distribution Q and the target distribution P_x.

The idea for a distribution moment score is based on work done in paper [23] on deep hedging where the authors use statistical moments to assess GAN equity option market simulators. In financial applications, higher-order moments are of interest as they define the nature of the extreme values in a sample-set or distribution. The empirical mean, variance, skew, and kurtosis scores are computed for the generated datasets and compared to the moments of the target distribution. We define the distributional scores as:

Definition 3.4 (Distributional Moment Scores)**.** The empirical distributional moment scores for the target distribution P_x and the generated sample set $\hat{x}_j$ for $j \in 1, ...M$ from the generated distribution Q

$$
\begin{aligned}
|\operatorname{mean}(P_x) - \operatorname{mean}([\hat{x}_1, ...\hat{x}_M])|,\\
|\operatorname{variance}(P_x) - \operatorname{variance}([\hat{x}_1, ...\hat{x}_M])|,\\
|\operatorname{skew}(P_x) - \operatorname{skew}([\hat{x}_1, ...\hat{x}_M])|,\\
|\operatorname{kurtosis}(P_x) - \operatorname{kurtosis}([\hat{x}_1, ...\hat{x}_M])|,
\end{aligned}
$$

3.4 Evaluation of Results

GAN models were trained on three distributions and with a range of sample sizes. Since multiple choices exist for a GAN model's hyperparameters, as partly showcased in subsection 3.1, only the results from the best-performing models are assessed in this section.

All trials began with a given target distribution P_x and a training sample size s. The training sample s was drawn by random sampling from P_x and used to train the GAN. Once trained, the GAN generator is used to produce a synthetic sample $\hat{x}$ of size 10,000 from the GAN's learned distribution Q. The distribution is assessed against a sample drawn from P_x of size 10,000.

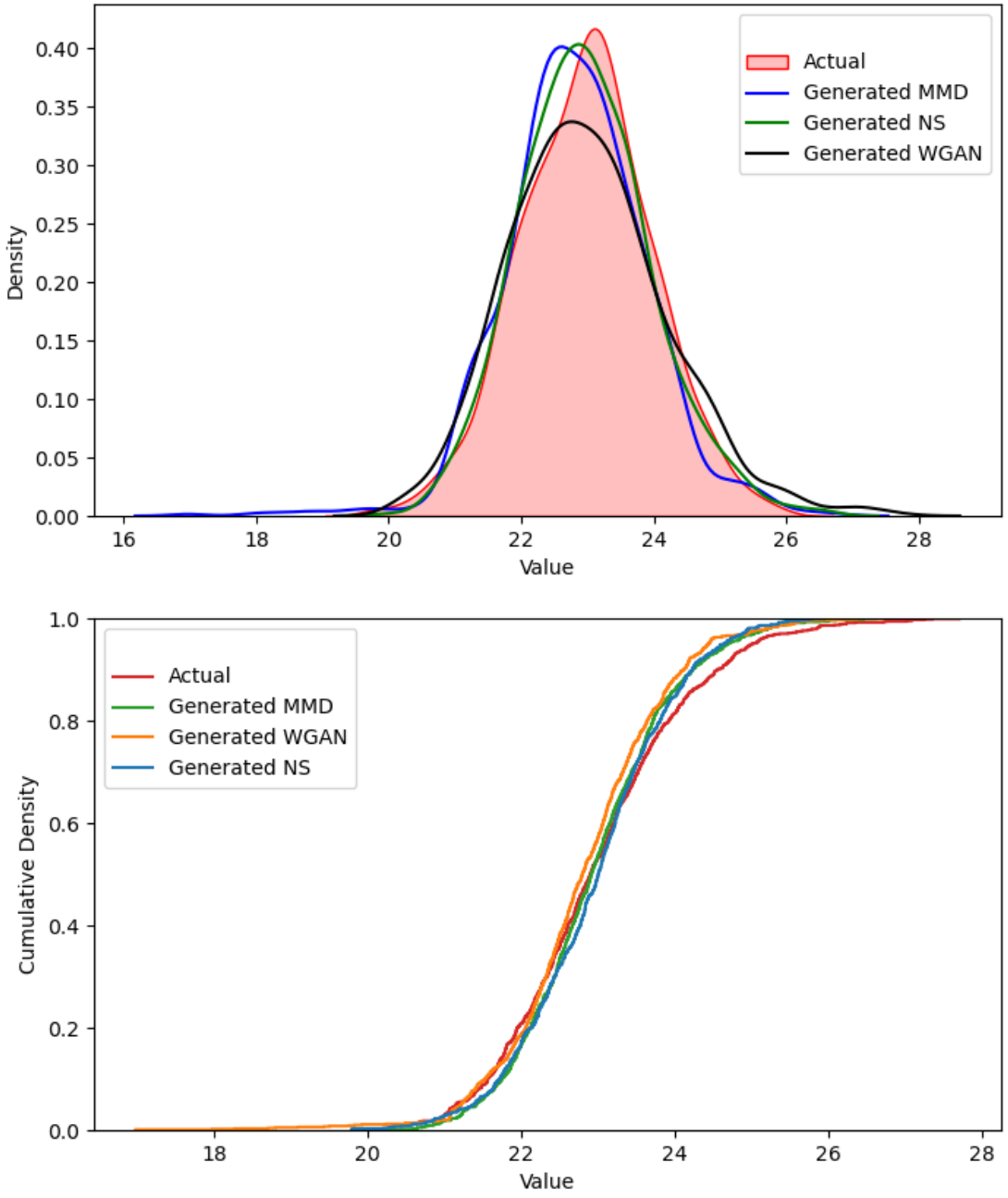
3.4.1 Normal Distribution Results

The first parametric distribution tested is a normal distribution. This distribution was chosen as portfolio return distributions are often modelled using this assumption. The target distribution is $X \sim N(23, 1)$. This was chosen in line with work done in paper [26]. The best models were found by varying the different hyperparameters for each GAN variant. The best GAN model for each variant was selected based on the training set results. Table 6 describes the hyperparameter choices for each variant.

Hyperparameter	NS	WGAN	MMD
Latent Distribution	$N(0, 1)$	$N(0, 1)$	$N(0, 1)$
Latent Space Dimension	20	20	1
Generator Architecture	G^2	G^2	G^1
Discriminator Architecture	D^1	D^1	D^1
Sample Size	1280	320	640

Table 6: The NS, MMD, and WGAN GAN model hyperparameters for the best-tuned models (based on a combination of result metrics) for producing a target normal distribution $N(23, 1)$.

Figure 4: KDE and ECDF showing the MMD, WGAN and NS GAN results for the target normal distribution.



GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.093	0.309	0.460	0.261
NS	640	0.002	0.077	0.446	0.318
NS	1280	0.012	0.025	0.384	0.325
MMD	320	0.274	0.221	0.150	0.066
MMD	640	0.168	0.417	0.478	0.202
MMD	1280	0.059	0.160	0.314	0.256
WGAN	320	0.163	0.680	0.421	0.279
WGAN	640	0.168	0.417	0.478	0.202
WGAN	1280	0.230	0.392	0.384	0.195

Table 7: The distribution moment scores results, as defined in 3.4, for the best performing MMD, WGAN and NS GANs for a range of sample sizes s with a latent distribution of $N(0, 1)$.

GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.443	0.132	0.123	0.031
NS	640	0.182	0.270	0.161	0.062
NS	1280	0.307	0.408	0.087	0.011
MMD	320	0.162	0.128	0.172	0.038
MMD	640	0.060	0.008	0.160	0.033
MMD	1280	0.049	0.322	0.031	0.011
WGAN	320	0.058	0.502	0.077	0.020
WGAN	640	0.061	0.828	0.035	0.012
WGAN	1280	0.148	0.337	0.015	0.052

Table 8: The distribution moment scores, as defined in 3.4, for the best performing MMD, WGAN and NS GANs for a range of sample sizes s with a latent distribution of $U(-1, 1)$.

Table 7 and 8 show the distribution moment scores for each of the GAN models. For GANs trained on a latent normal distribution, the NS GAN was the best at capturing the mean and variance of the distribution, this can be seen by the small distribution moment scores. For the skew and kurtosis, all GAN models performed relatively similarly. These moments seemed to be more difficult for the GAN models to capture accurately. Interestingly, GAN models trained on a latent uniform distribution were better able to capture the kurtosis and skew as they had much lower scores on average when compared to the GAN model results in Table 8. Figure 4 uses a kernel density function to approximate the histograms of the generated data with a continuous function, using kernel density estimation (KDE). From the both the

KDE and ECDF, it seems that the MMD and WGANs have slightly larger variances, which is confirmed in Table 7, while the mean seems similar across groups. The issue with relying on the KDE as a visual representation of model fit is that it is a black box method and might mask relevant features of the data, however the ECDF represents all the generated data points.

The size of the training dataset seems to have a marginal effect on the ability of the GAN to capture the distribution moments. There does not seem to be a clear, reliable correlation between training dataset size and lower distribution moment scores. This can be seen in Table 7, for both the MMD and NS GANs, an increase in sample size corresponded to a decrease in distribution moment scores, however the WGAN's ability to capture the mean decreases as sample size increases. The training sample size did, however, have a large effect on the GAN model training time.

GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.070	0.063	0.362	0.082	0.148
NS	640	0.058	0.001	1.391	0.003	0.129
NS	1280	0.049	0.004	0.756	0.011	0.091
MMD	320	0.096	0.004	1.170	0.0009	0.274
MMD	640	0.066	0.006	0.846	0.004	0.163
MMD	1280	0.140	0.00002	13.574	0.00003	0.122
WGAN	320	0.090	0.00001	0.673	0.00002	0.272
WGAN	640	0.085	0.0002	1.347	0.0004	0.204
WGAN	1280	0.133	0.0009	9.810	0.001	0.150

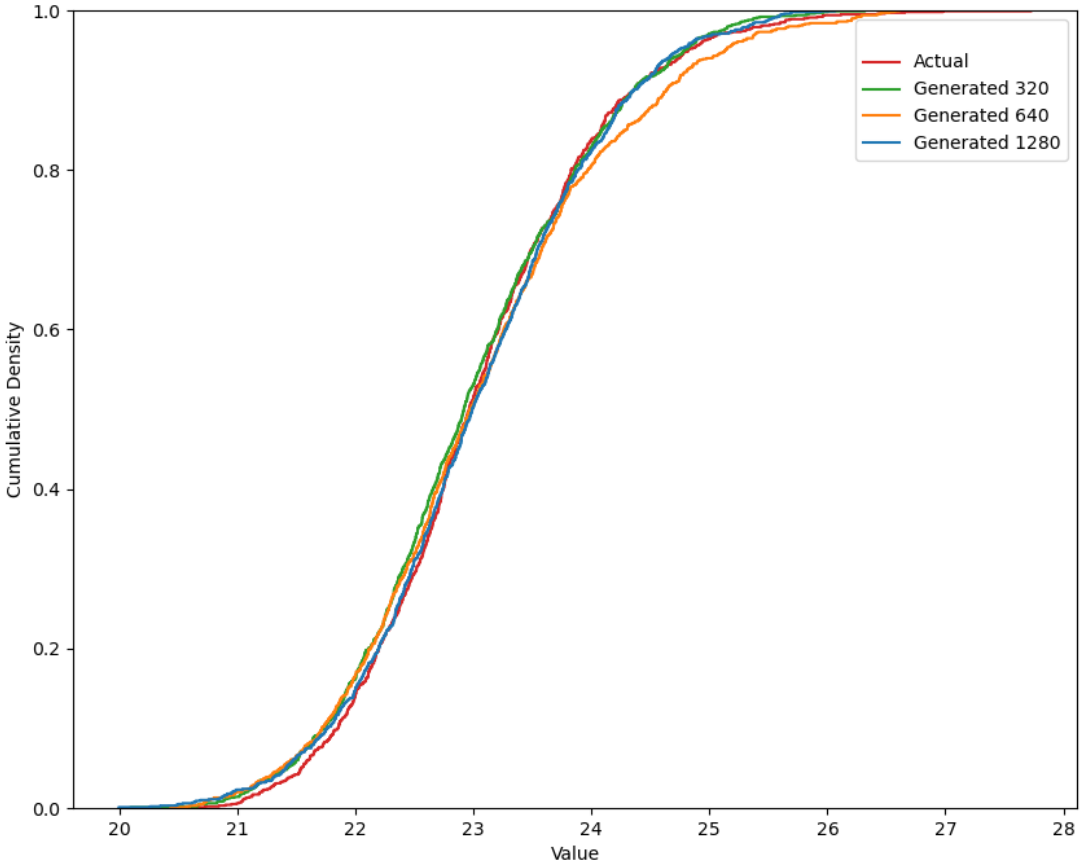
Table 9: Result table showing the Wasserstein distance, CVM test statistic and p-value, and the KS test statistic and p-value for the selected GAN models trained on a latent normal distribution with varying training set sample sizes.

GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.0941	0.006	0.897	0.004	0.380
NS	640	0.1189	2.72e-8	3.203	2.66e-8	0.2186
NS	1280	0.1686	6.60e-12	13.597	3.376e-10	0.3051
MMD	320	0.1271	0.0005	1.989	0.0001	0.2421
MMD	640	0.0781	0.0007	1.1683	0.0009	0.1253
MMD	1280	0.0699	8.0385e-6	1.947	0.00001	0.1311
WGAN	320	0.0811	0.0284	0.7088	0.0120	0.2253
WGAN	640	0.0998	5.719e-6	1.9805	1.4057e-6	0.2887
WGAN	1280	0.07083	5.8359e-6	1.8317	3.049e-6	0.1522

Table 10: Result table showing the Wasserstein distance, CVM test statistic and p-value, and the KS test statistic and p-value for the selected GAN models trained on a latent uniform distribution with varying training set sample sizes.

Tables 9 and 10 present the Wasserstein distance, the test statistic values and the p-values for the KS and CVM two-sample tests. The NS GAN models trained on a latent normal distribution were able to capture the target distribution the best, according to the Wasserstein distance. The non-parametric two-sample tests were very sensitive, with only the NS GAN in Table 9, trained on the smallest training set, not being able to reject both null hypothesises. This was counter intuitive as the NS GAN trained on a larger training sample set scored better on both the Wasserstein distance score as well as the distribution moment scores. Figure 5, however, shows that the NS GAN trained on a 320 point sample set follows the ECDF of the target distribution the best, which confirms the CVM and KS test results. The results for both two sample tests in Table 10 were disappointing, even though the Wasserstein distances for both Table 9 and 10 are very similar.

Figure 5: The ECDF showing the NS GAN results, trained on different training set sizes, for the target normal distribution.



3.4.2 Two Component Gaussian Mixture Model Results

A GMM can achieve great flexibility with only a few components. This is useful when modelling financial returns as desired distributional properties can be captured by mixing component Gaussians.

Definition 3.5 (GMM)**.** Given a finite set of probability density functions (PDFs) of general normal random variables $f_1(x; \mu_1, \sigma_1), ..., f_n(x; \mu_n, \sigma_n)$ and weights $w_1, ..., w_n$ such that $w_i \geq 0$ and $\sum w_i = 1$, the mixture distribution can be represented by writing the density, f, as a sum:

$$f(x) = \sum_{i=1}^{n} w_i f_i(x; \mu_i, \sigma_i).$$

This type of mixture, being a finite sum, is called a finite mixture. This is considered a more complicated distribution to replicate, from a GAN perspective, as it is made up of component Gaussian distributions with different weightings. Two GAN mixtures are chosen for investigation. The first mixture chosen has two distinct modes and is not meant to resemble a return distribution - it is chosen to see if the GAN models can capture more complicated target distributions. Target mixture one is:

$$f_1(x) = 0.5f(x; 1, 0.2) + 0.5f(x; 2, 0.2).$$

The best GAN model for each variant was selected based on the training set results. Table 11 describes the hyperparameter choices for each variant.

Hyperparameter	NS	WGAN	MMD
Latent Distribution	$N(0,1)$	$U(-1,1)$	$N(0,1)$
Latent Space Dimension	10	20	20
Generator Architecture	G^2	G^2	G^2
Discriminator Architecture	D^1	D^1	D^1
Sample Size	1280	640	320

Table 11: The NS, MMD, and WGAN GAN model hyperparameters for the best-tuned models (based a combination of result metrics) for producing a target GMM $0.5f(x; 1, 0.2) + 0.5f(x; 2, 0.2)$.

GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.074	0.001	0.018	0.017
NS	640	0.002	0.052	0.035	0.028
NS	1280	0.011	0.003	0.041	0.012
MMD	320	0.003	0.006	0.085	0.186
MMD	640	0.005	0.017	0.108	0.096
MMD	1280	0.013	0.014	0.038	0.063
WGAN	320	0.031	0.016	0.012	0.046
WGAN	640	0.104	0.009	0.056	0.053
WGAN	1280	0.005	0.007	0.003	0.002

Table 12: The distribution moment scores, as defined in 3.4, for the best performing MMD, WGAN and NS GANs, trained to produce a GMM of $0.5f(1, 0.2) + 0.5f(2, 0.2)$, for a range of sample sizes s with a latent distribution of $N(0, 1)$.

GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.123	0.046	0.075	0.083
NS	640	0.037	0.105	0.056	0.267
NS	1280	0.003	0.094	0.007	0.098
MMD	320	0.098	0.025	0.151	0.235
MMD	640	0.058	0.017	0.145	0.003
MMD	1280	0.039	0.010	0.063	0.040
WGAN	320	0.408	0.287	0.113	0.534
WGAN	640	0.478	0.313	0.182	0.618
WGAN	1280	0.046	0.004	0.028	0.014

Table 13: The distribution moment scores, as defined in 3.4, for the best performing MMD, WGAN and NS GANs, trained to produce a GMM of $0.5f(1, 0.2) + 0.5f(2, 0.2)$, for a range of sample sizes s with a latent distribution of $U(-1, 1)$.

The GMM distribution scores are presented in Table 12 and 13. GANs trained on a normal latent distribution performed slightly better, in terms of distribution moment scores, than GANs trained on a uniform latent distribution. As in the target normal results, the GMM results don't seem to have a reliable correlation between training sample set size and lower distribution moment scores, although the best performing models for each variant, besides the MMD GAN, were trained on the largest training set size. When looking at Table 12 alone, the distribution moment scores were low across all variants and models performed similarly across the different training set sizes. Figure 6 shows that the three GAN variants struggled to accurately capture the right tail of the GMM, both in the KDE and the ECDF.

GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.101	0.002	0.883	0.004	0.063
NS	640	0.064	0.010	0.891	0.004	0.050
NS	1280	0.031	0.164	0.260	0.175	0.016
MMD	320	0.054	0.285	0.146	0.401	0.044
MMD	640	0.048	0.092	3.213	2.35e-7	0.043
MMD	1280	0.025	0.001	1.382	0.0003	0.025
WGAN	320	0.063	0.148	0.321	0.117	0.038
WGAN	640	0.110	2.8e-7	2.889	1.32e-7	0.100
WGAN	1280	0.018	0.7921	0.058	0.821	0.010

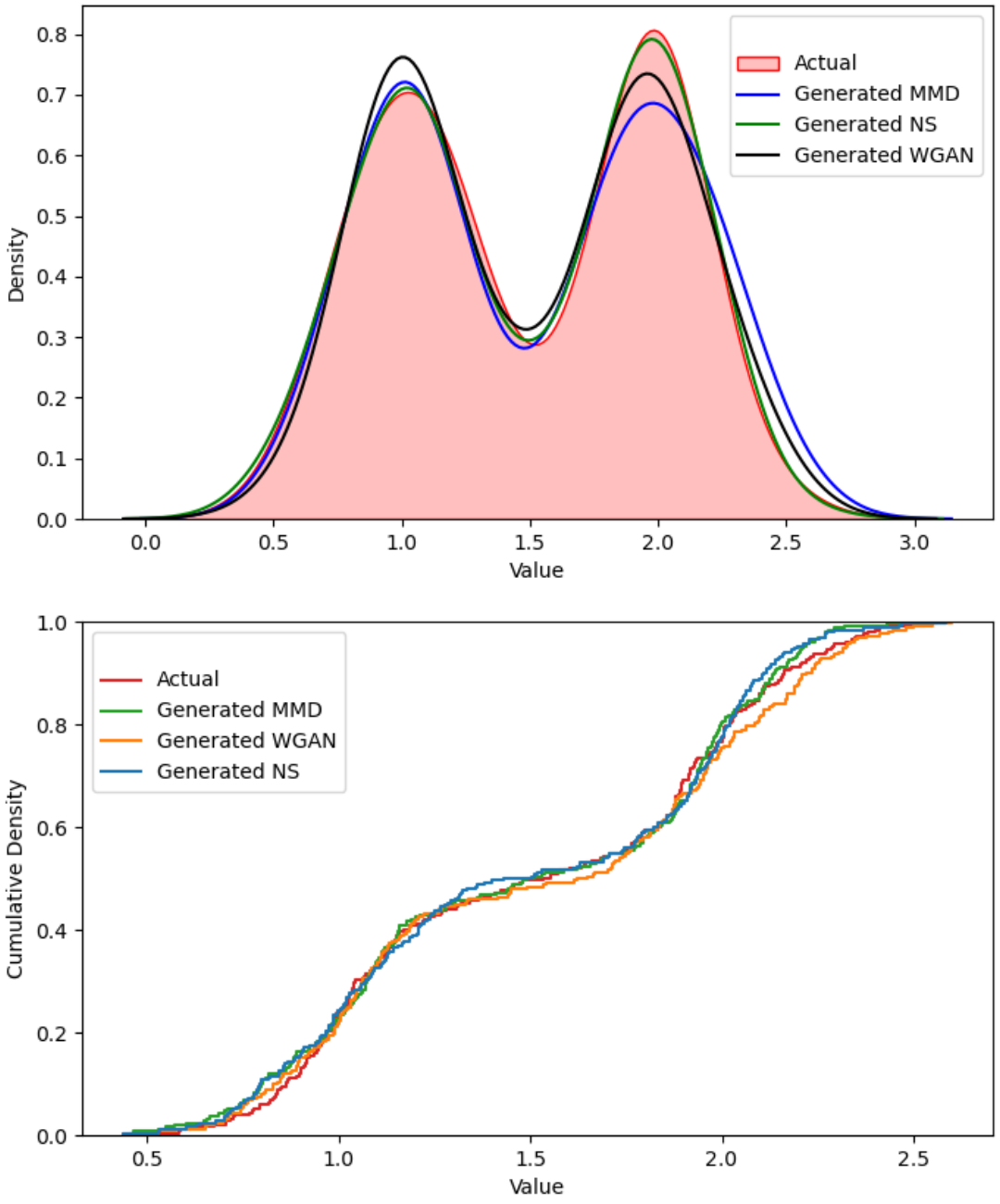
Table 14: Result table showing the Wasserstein (W) distance, CVM p-value, and the KS test p-value for the selected GAN models, trained on $N(0, 1)$ to produce a GMM of $0.5f(1, 0.2) + 0.5f(2, 0.2)$.

GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.1989	1.52e-9	3.3875	1.034e-8	0.1078
NS	640	0.1716	7.504e-10	5.826	3.9684e-11	0.0934
NS	1280	0.110	8.298e-10	6.620	6.429e-10	0.0907
MMD	320	0.0934	0.007	9.656	1.005e-7	0.1018
MMD	640	0.0722	0.0024	1.5023	0.0001	0.0576
MMD	1280	0.0553	0.0008	1.1603	0.001	0.0401
WGAN	320	0.0598	0.1955	0.1352	0.4381	0.0265
WGAN	640	0.0329	0.8673	0.052	0.8647	0.0197
WGAN	1280	0.0655	0.00003	0.9765	0.0027	0.0361

Table 15: Result table showing the Wasserstein (W) distance, CVM p-value, and the KS test p-value for the selected GAN models, trained on $U(-1, 1)$ to produce a GMM of $0.5f(1, 0.2) + 0.5f(2, 0.2)$.

Tables 14 and 15 gave more information to differentiate the best performing GAN models, as the results in Tables 12 and 13 were similar for each variant. The Wasserstein distances were the smallest, for each variant, when the GAN was trained on a larger training sample set. The GANs trained on the 1280 training sample set in Table 14 also had larger p-values for both the CVM and KS tests, with the MMD GAN variant being the exception. This was the main basis for model selection as most other metrics were very similar across each variant.

Figure 6: KDE and ECDF showing the MMD, WGAN and NS GAN results for the target GMM.



3.4.3 Three Component Gaussian Mixture Model Results

The second mixture was constructed to represent an empirical return distribution. The daily returns of the FTSE/JSE Top40 from January 2000 to October 2022, listed on the JSE, were split into three components. Daily returns less than negative two percent made up the first component, daily returns greater than two percent made up the second and returns in between these points made up the third component. The mean, standard deviation and component weight were calculated for each component to define the parametric GMM. More formally, with x being the daily returns of FTSE/JSE Top40 and N being the total samples:

$$\begin{aligned} f_2(x) =& \frac{n_1}{N} f(x < -0.02; \mu_{n_1}, \sigma_{n_1}) \\ &+ \frac{n_2}{N} f(-0.02 \leq x \leq 0.02; \mu_{n_2}, \sigma_{n_2}) \\ &+ \frac{n_3}{N} f(x > 0.02; \mu_{n_3}, \sigma_{n_3}). \end{aligned} \tag{10}$$

The resulting parameters are:

Table 16: Empirical FTSE/JSE Top40 GMM parameters

Parameter	$x < -0.02$	$-0.002 < x < 0.002$	$x > 0.02$
Weight	0.05	0.88	0.07
μ	-0.0315	-0.0001	0.0282
σ	0.01356	0.0092	0.0099

The best GAN models for each variant were selected based on training set results. Table 17 describes the hyperparameters for each variant.

Hyperparameter	NS	WGAN	MMD
Latent Distribution	$N(0,1)$	$U(-1,1)$	$N(0,1)$
Latent Space Dimension	20	20	10
Generator Architecture	G^2	G^2	G^1
Discriminator Architecture	D^1	D^1	D^1
Sample Size	640	640	320

Table 17: The NS, MMD, and WGAN GAN model hyperparameters for the best-tuned models (based on a combination of result metrics) for producing the target three component GMM.

The results for the three-component GMM were less convincing when compared to the two-component mixture presented above. Table 18 and 19 display the results for the models. It is clear that each model is capable of capturing the mean and variance of the target distribution well, however, the skew and kurtosis were not accurately modelled by the generator models. This is indicated by the large distributional moment scores. When looking at the distributional moment scores only, the NS GAN trained on 640 samples performed the best. It is interesting to note that the MMD GAN models were worse at capturing kurtosis when, theoretically, they should capture the higher order moments within the loss function through the KMEs, elaborated in subsection 2.2.5. KMEs may be more beneficial for generating realistic outputs in higher-order domains, such as generating images, when compared to the 1D space.

GAN models trained on a normal latent distribution performed similarly to GANs trained on a uniform latent distribution, when only looking at the distribution moment scores. Looking at the results presented in Tables 18 and 19, the test statistic p-values vary wildly across the different GAN models. These values don't seem to correspond to a good distribution moment score.

GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.0003	1.75e-7	0.151	1.0636
NS	640	0.0001	3.45e-5	0.06995	0.0993
NS	1280	0.0002	1.593e-5	0.550	0.086
MMD	320	4.570e-5	2.032e-5	2.431	15.416
MMD	640	0.0004	0.0001	0.7899	4.979
MMD	1280	0.0023	0.0001	2.3917	13.911
WGAN	320	0.0005	0.0001	0.5635	1.4679
WGAN	640	0.0054	0.0002	0.3126	1.8478
WGAN	1280	0.0025	0.0002	0.2094	1.8054

Table 18: The distribution moment scores, as defined in 3.4, for the best performing MMD, WGAN and NS GANs, trained to produce a GMM in Equation 10, for a range of sample sizes s with a latent distribution of $N(0, 1)$.

GAN	s	Mean	Variance	Skew	Kurtosis
NS	320	0.0003	0.0001	0.0386	2.6482
NS	640	0.0001	0.0008	0.0174	1.6691
NS	1280	0.0009	0.00001	0.1793	0.2916
MMD	320	0.0024	0.0001	0.6197	11.8472
MMD	640	0.0021	0.00001	1.9418	6.2659
MMD	1280	0.0014	0.00004	0.5605	5.1762
WGAN	320	0.0019	0.00002	0.4830	2.5103
WGAN	640	0.0007	0.00001	0.4193	2.2272
WGAN	1280	0.0026	0.00003	0.4015	2.4099

Table 19: The distribution moment scores, as defined in 3.4, for the best performing MMD, WGAN and NS GANs, trained to produce a GMM in Equation 10, for a range of sample sizes s with a latent distribution of $U(-1, 1)$.

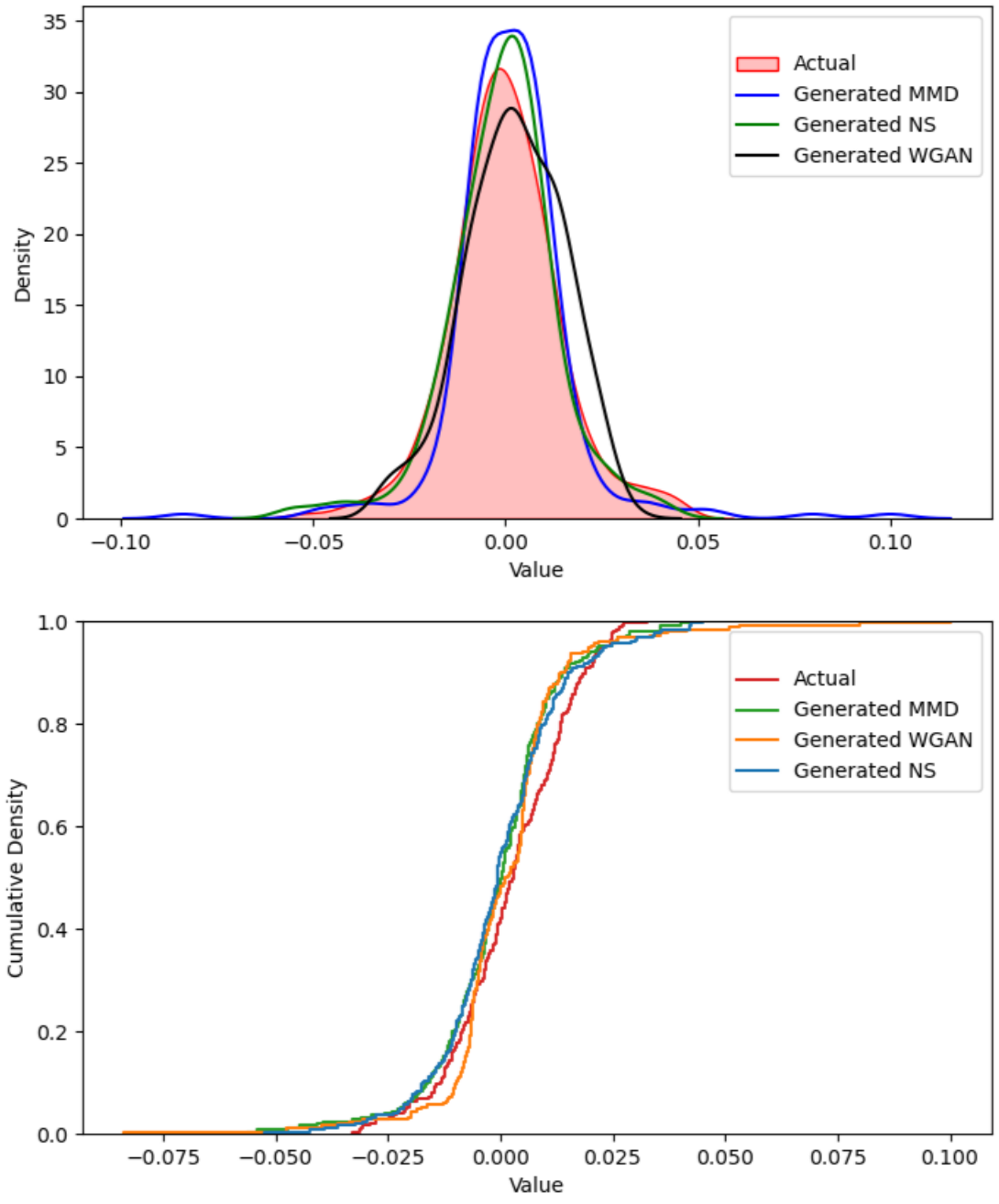
GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.0456	0.7021	0.01258	0.0213	0.0011
NS	640	0.03513	0.4022	0.2063	0.2554	0.0008
NS	1280	0.0686	0.00001	1.2336	0.0007	0.001
MMD	320	0.0762	0.0464	0.5318	0.033	0.0026
MMD	640	0.0679	0.0053	0.7820	0.008	0.0022
MMD	1280	0.0421	0.0216	0.5276	0.0338	0.0028
WGAN	320	0.144	0.00003	1.9841	0.00001	0.0047
WGAN	640	0.2303	3e-7	11.175	3.5e-9	0.0083
WGAN	1280	0.1494	1e-7	10.23	2.72e-9	0.0033

Table 20: Result table showing the Wasserstein distance, CVM p-value, and the KS test p-value for the selected GAN models, trained to produce a GMM in Equation 10 on a normal latent distribution, with varying training set sample sizes.

GAN	s	KS Value	KS p-value	CVM Value	CVM p-value	W distance
NS	320	0.0946	0.0061	0.6078	0.0213	0.0027
NS	640	0.2203	1.324e-10	12.553	2.429e-9	0.0145
NS	1280	0.0550	0.0009	1.033	0.002	0.0014
MMD	320	0.1342	0.00001	27.771	1.7541e-9	0.003
MMD	640	0.0678	0.0054	0.9393	0.0034	0.003
MMD	1280	0.0536	0.0013	0.8486	0.0055	0.0015
WGAN	320	0.1103	0.0007	1.0229	0.0021	0.0025
WGAN	640	0.069	0.0044	0.8382	0.0059	0.0016
WGAN	1280	0.0898	2.474e-8	4.191	1.794e-9	0.0027

Table 21: Result table showing the Wasserstein distance, CVM p-value, and the KS test p-value for the selected GAN models, trained to produce a GMM in Equation 10 on a uniform latent distribution, with varying training set sample sizes.

Figure 7: The ECDF showing the NS, MMD and WGAN results, trained on different training set sizes, for the target GMM.



4 Value-at-Risk with GANs

There are multiple measures that can be used to measure risk in a financial institution. One of the more popular metrics being Value-at-Risk (VaR). VaR is a loss that, at a certain level of confidence, we are sure will not be exceeded if the portfolio in question is held over a defined period of time. It is a quantile risk metric that depends on the defined confidence level $(1 - \alpha)$ and the risk horizon h. The risk horizon is the period of time, usually denoted in number of trading days, over which the VaR is measured.

Currently, almost all financial institutions use some form of VaR as a risk metric, however VaR has some weaknesses. The BCBS has been revising its market risk framework since 2012. The result of its *fundamental review of the trading book* means there will be a move from VaR to Expected Tail Loss (ETL) [2]. ETL, otherwise known as the conditional value-at-risk, provides an estimate of the potential loss beyond the VaR level. ETL represents the expected value of the loss that exceeds the VaR level, given that the loss is larger than the VaR level. It closely associated to VaR and it is sub-additive which means it accounts for the benefit of diversification across portfolios or assets. It is simple to calculate once a VaR model has been developed.

An accurate VaR estimation relies wholly on a reliable estimate of the portfolio profit and loss (PnL) or return distribution. VaR models differ only by the manner in which the PnL distribution is constructed and consequently, the VaR forecast is heavily dependent on the model in use. GANs are beneficial as they can be used to enrich the historical sample set with realistic synthetic returns. This, in theory, allows for a more reliable return distribution as the GAN approach allows us to deal with complex, high-dimensional data without the need to make any assumptions about the underlying statistical distribution. This enriched set of samples can be used in non-parametric VaR models to, in theory, produce a more accurate VaR forecast. The GAN can be trained to sample from either the aggregate portfolio returns or from the individual risk factors. In this dissertation the GAN will be trained to sample from the stationary distribution of the portfolio return time series.

There has been little research into VaR modelling with GANs. We are aware of two blog posts, which focus on modelling the aggregated portfolio return, and do not subsequently backtest their model, and a masters thesis [4] which extends the work to include conditional and unconditional VaR modelling on a risk factor and aggregate portfolio level with backtesting. One of the main goals of this dissertation is to contribute to this work by looking at training the GAN on synthetically generated returns, using parametric distributions, and subsequently backtesting these models. A historical VaR model is used as the baseline for comparison with the enriched GAN

[2] Although the deadline for this move keeps being extended

sample. By assessing the model performance on synthetic returns, more confidence can be given to the expected results. Three GAN loss variants are assessed for VaR estimation, namely the WGAN, the NS GAN and the MMD GAN. The MMD GAN was not included in [4], [12] or [18]. Our goal is to obtain reliable VaR forecasts at the 95% and 99% confidence level on an aggregate portfolio level for the synthetically generated return distributions created in Section 3.4.

In this dissertation, we focus on historical VaR even though it is theoretically less appealing than ETL as it remains the industry standard until the FRTB is implemented. This section formally defines historical VaR and how the forecast is backtested. The results for the 1D portfolios are discussed in subsection 4.3.

4.1 Historical Value at Risk Definition

In this section we set up the notational framework, following [15]. We shall assume that VaR is measured at a portfolio level. As mentioned in 4 VaR is a quantile risk metric that depends on the defined confidence level $(1-\alpha)$ and the risk horizon h. The value of the portfolio of risky assets at time t is denoted by V_t. If we consider a period $[t, t+h]$ then the actual profit and loss over this period is

$$\triangle V_t = V_{t+h} - V_t.$$

There are many different methods to estimate VaR. For this dissertation, only unconditional historical VaR models will be considered. This model enables one to estimate the stationary distribution of the portfolio PnL, relying on the assumption that the portfolio returns follow a stationary time-series, and to calculate the corresponding VaR quantile.

Historical simulation is completely data-driven. It does not require any *apriori* distributional assumptions. It is easy to implement as it uses historical returns to form an empirical distribution for the portfolio profit and loss. The corresponding quantile of the ECDF is seen as the VaR estimate.

More formally, historical portfolio values $\{V_t, V_{t-h}...V_{t-mh}\}$ are used to find the simulated value for the portfolio return x_j at $t+h$

$$x_j = \frac{V_{t-(j-1)h} - V_{t-jh}}{V_{t-jh}},$$

for $j \in \{1, 2...m\}$.

The historical VaR at a certain confidence level (e.g., 95%) is calculated as follows:

1. Sort the historical returns in ascending order:

$$x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(m)}$$

2. Determine the index j such that the j-th percentile corresponds to the desired confidence level. For example, if the confidence level is 95%, then $j = 0.95 \times m$.
3. Calculate the historical VaR:

$$\text{VaR} = -x_{(j)}$$

Here, $x_{(j)}$ represents the j-th percentile of the historical returns. The negative sign is used because VaR is a measure of potential loss.

It's important to note that historical VaR has limitations, such as its sensitivity to the time period chosen and the assumption that historical returns are representative of future returns. Additionally, the choice of the confidence level determines the level of risk that the VaR is estimating. In the case above $\alpha = 1 - \frac{j}{m}$.

4.2 Backtesting VaR

Backtesting a VaR model is evaluating the model's accuracy. It is a formal statistical framework that verifies whether or not estimated losses are in line with the actual losses. This is done by comparing the history of VaR forecasts with the associated portfolio returns over the testing period.

The aim of backtesting is to identify whether the actual losses exceed the expected losses at a given level of confidence. A breach is when the actual loss is above the expected level. The percentage of breaches should not exceed α. In other words, at a $(1 - \alpha)$ confidence, breaches should occur less than α of the time. When there are more breaches observed than expected, this indicates the model is not estimated properly and underestimates risk.

Definition 4.1 (Historical VaR Breaches)**.** Historical VaR breaches are defined as the number of observations in Q that are smaller than the target sample's, P_x, historical VaR quantile x_j, calculated as described in 4.1.

$$B = \frac{Q[\hat{x} < x_j]}{t}$$

With $Q = \{\hat{x}_1, \hat{x}_2 ... \hat{x}_t\}$.

4.3 Evaluation of Results

This dissertation considers fictitious portfolios that have one-period returns and that are distributed according to a normal or a GMM distribution. The VaR for these portfolios is assessed at a 95% and 99% confidence level. The enriched sample set is compared to the results for the corresponding historical VaR estimate.

The historical VaR tests are completed using a randomly drawn, 252-size sample set from the target distribution P_x. The target distribution's historical VaR is not constant as it is recalculated each run so there is some random sampling error. The target sample is used to find the quantile x_j that is used to calculate the number of breaches with regards to the calculated VaR estimate from the various GAN models. The various GAN model VaR estimates are made using a sample set of 10,000 from Q. The historical VaR estimate is calculated using the methodology discussed in subsection4.1 and the number of breaches for each GAN model is calculated as discussed in 4.1.

4.3.1 Normal Distribution

The VaR breaches for the generated normal distributions are presented in Table 22. Most models performed in line with expected breaches. The MMD GAN trained on 640 samples was not able to capture the tails correctly and consequently underestimated the risk at the 1% percentile. The best performing GAN model, with regards to VaR, was the NS GAN. The results in Table 23 were less promising and performed in line with the historical VaR estimate. When looking at the results in their entirety, it does not seem like GAN models would be a beneficial addition to a historical VaR model as the results are often unpredictable and perform very similarly to historical VaR in most cases. Historical VaR is much easier to implement than a GAN model, as discussed in subsection 2.3.

GAN	s	Hist 5%	Hist 1%	GAN 5%	GAN 1%
NS	320	3.17	0.39	5.15	0.39
NS	640	4.76	0.39	6.75	1.98
NS	1280	3.96	0.79	5.95	1.19
MMD	320	3.57	0.39	3.96	0.39
MMD	640	4.76	0.39	3.174	0.2
MMD	1280	5.55	0.79	6.34	1.19
WGAN	320	5.92	0.1	3.57	0.39
WGAN	640	9.12	0.79	8.33	2.38
WGAN	1280	3.96	1.19	3.57	0.79

Table 22: The historical VaR percentage breaches compared to the GAN model VaR breaches for the target normal distribution. GAN models were trained on a normal latent distribution.

GAN	s	Hist 5%	Hist 1%	GAN 5%	GAN 1%
NS	320	3.57	1.19	2.38	0.39
NS	640	4.76	1.19	12.69	4.36
NS	1280	3.17	0.0	5.15	1.58
MMD	320	5.92	1.58	2.77	1.19
MMD	640	5.95	1.58	2.384	0.79
MMD	1280	7.14	2.90	10.31	4.76
WGAN	320	7.14	1.98	2.77	0.0
WGAN	640	3.17	1.19	1.58	0.79
WGAN	1280	9.12	1.77	6.34	0.79

Table 23: The historical VaR percentage breaches compared to the GAN model VaR breaches for the target normal distribution. GAN models were trained on a uniform latent distribution.

4.3.2 Three-Component Gaussian Mixture Models

Only the three-component GMM is tested for VaR. The results are presented in Tables 24 and 25. As in subsection 4.3.1, the results, when compared to the historical VaR models, are quite disappointing. There is not a clear outperformance for the GAN generated samples versus the historical samples, in most cases the GAN model performs in line with the historical model in Table 24, however in Table 25 the GAN models perform worse than the historical VaR models.

GAN	s	Hist 5%	Hist 1%	GAN 5%	GAN 1%
NS	320	4.8	0.8	4.8	2.8
NS	640	4.4	1.2	6.0	1.2
NS	1280	3.6	2.4	3.6	2.4
MMD	320	4.0	0.4	4.0	1.6
MMD	640	3.6	1.6	3.2	0.4
MMD	1280	4.8	1.6	3.2	0
WGAN	320	2.5	1.56	4.0	2.0
WGAN	640	4.4	1.2	2.8	1.2
WGAN	1280	6.0	0.8	6.0	1.2

Table 24: The historical VaR breaches compared to the GAN model VaR breaches for the target GMM distribution. GAN models were trained on a normal latent distribution.

GAN	s	Hist 5%	Hist 1%	GAN 5%	GAN 1%
NS	320	3.2	1.6	5.2	3.6
NS	640	4.8	0.8	0.8	0.0
NS	1280	5.6	2.0	4.8	0.8
MMD	320	4.4	0.0	4.8	0.0
MMD	640	4.8	0.0	8.4	3.6
MMD	1280	4.8	1.2	4.4	0.0
WGAN	320	7.6	2.0	4.8	2.4
WGAN	640	2.8	2.0	3.2	2.0
WGAN	1280	4.4	0.8	7.6	3.2

Table 25: The historical VaR breaches compared to the GAN model VaR breaches for the target GMM distribution. GAN models were trained on a normal latent distribution.

5 Conclusion

The aim of this dissertation was to present the GAN framework in an accessible way as well as to apply GANs to the familiar financial scenario of VaR estimation for a 1D portfolio of returns and in doing so, assess whether the GAN models were able to adequately capture the statistical features of these 1D distributions. We first introduced the concept of implicit generative models and then went into an in-depth discussion on the GAN framework and the various loss variants that were selected for investigation. This section highlighted the ability of GANs to approximate probability distributions by being able to model the sampling process. This feature makes GANs attractive for tasks where we want to draw a large sample set from the target distribution and are not necessarily interested in the probability distribution itself.

High dimensional applications of GANs have been thoroughly researched, particularly natural images - this dissertation focused on one-dimensional applications, which are less common in prevalent research. We began this study by evaluating if the various GAN models could accurately generate a fictitious portfolio of returns. The 1D parametric distributions chosen were a normal distribution and two GMMs. Some models were relatively successful, however, it turned out to be incredibly difficult and tedious to find a set of hyperparameters for each model such that the resultant GAN could capture the distributional moments accurately. Using these GAN models, this dissertation introduced readers with the concepts and practical implementations of GAN variants to generate these 1D distributions. Five GAN loss variants were introduced and three of these models are practically implemented to estimate VaR. The GAN estimates were compared to more traditional VaR estimation techniques and all models were backtested. It was clear that some GAN models were able to capture the target 1D distribution, and consequently, these models performed comparably to historical models when used for historical VaR estimation.

The training procedure for GANs highlighted the strengths and weaknesses of this framework when applied to a low-dimensional settings. These models are very difficult to train and difficult to assess when not using a defined parametric target distribution. They lack a defined method for choosing hyperparameters in a robust way which makes it difficult to efficiently apply the framework. The models are, additionally, very sensitive to slight changes in hyperparameter choices which means slight variations could lead to inaccurate estimations of VaR. This is a large risk for practitioners looking to employ this methodology.

References

[1] Grigory Antipov, Moez Baccouche, and Jean-Luc Dugelay. Face Aging with Conditional Generative Adversarial Networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2089–2093, 2017.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. In *International Conference on Machine Learning*, pages 214–223. PMLR, 2017.

[3] Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. *arXiv preprint arXiv:1801.01401*, 2018.

[4] Lukas-Benedikt Fiechtner. Risk Management with Generative Adversarial Networks. `https://www.maths.ox.ac.uk/node/34183`, 2019.

[5] Ian Goodfellow. NIPS 2016 Tutorial: Generative Adversarial Networks. `https://arxiv.org/abs/1701.00160`, 2017.

[6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[7] A Gretton, K. Borgwardt, Malte Rasch, Bernhard Schölkopf, and AJ Smola. A Kernel Two-Sample Test. *The Journal of Machine Learning Research*, 13:723–773, 03 2012.

[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. *Advances in Neural Information Processing Systems*, 30:5769–5779, 2017.

[9] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-Image Translation with Conditional Adversarial Networks. *CVPR*, 2017.

[10] Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the Automatic Anime Characters Creation with Generative Adversarial Networks. *CoRR*, abs/1708.05509, 2017.

[11] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[12] Santanu Khan. Calculate Value-at-Risk Using Wasserstein Generative Adversarial Networks (WGAN-GP) for Risk Management System.

https://chatbotslife.com/calculate-value-at-risk-using-wasserstein-generative-adversarial-networks-wgan-gp-for-risk-2b1d320fde59, 2019. Date Accessed: 2022.

[13] Chun-Liang Li. Learning Generative Models using Transformations. `https://kilthub.cmu.edu/articles/thesis/Learning_Generative_Models_using_Transformations/11878179`, 2019.

[14] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. MMD GAN: Towards Deeper Understanding of Moment Matching Network. *Advances in neural information processing systems*, 30, 2017.

[15] Obeid Mahomed. *Risk Management of Financial Instruments*. University of Cape Town, 2016.

[16] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which Training Methods for GANs Do Actually Converge? In *International Conference on Machine learning (ICML)*, pages 3481–3490. PMLR, 2018.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[18] Hamaad Shah. "Using Bidirectional Generative Adversarial Networks to estimate Value-at-Risk for Market Risk Management". https://towardsdatascience.com/using-bidirectional-generative-adversarial-networks-to-estimate-value-at-risk-for-market-risk-c3dffbbde8dd, 2018. Date Accessed: 2022.

[19] Rajhans Singh, Pavan Turaga, Suren Jayasuriya, Ravi Garg, and Martin W. Braun. Non-Parametric Priors For Generative Adversarial Networks. In *International Conference on Machine Learning*, pages 5838–5847. PMLR, 2019.

[20] Sergios Theodoridis. Chapter 14 - Monte Carlo Methods. In Sergios Theodoridis, editor, *Machine Learning*, pages 707–744. Academic Press, Oxford, 2015.

[21] Wei Wang, Yuan Sun, and Saman Halgamuge. Improving MMD-GAN Training with Repulsive Loss Function. `https://arxiv.org/abs/1812.09916`, 2018.

[22] Zhengwei Wang, Qi She, and Tomas E. Ward. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. `https://arxiv.org/abs/1906.01529`, 2019.

[23] Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. Deep Hedging: Learning to Simulate Equity Option Markets. *arXiv preprint arXiv:1911.01700*, 2019.

[24] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. Quant GANs: Deep Generation of Financial Time Series. *Quantitative Finance*, 20(9):1419–1440, apr 2020.

[25] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An Empirical Study on Evaluation Metrics of Generative Adversarial Networks. `https://arxiv.org/abs/1806.07755`, 2018.

[26] Manzil Zaheer and Barnabás Póczos. Connoisseur: Can GANs Learn Simple 1D Parametric Distributions ? In *Semantic Scholar*, 2018.

[27] Kang Zhang, Guoqiang Zhong, Junyu Dong, Shengke Wang, and Yong Wang. Stock Market Prediction Based on Generative Adversarial Network. *Procedia Computer Science*, 147:400–406, 01 2019.

A Appendices

A.1 Code

For all code used please refer to the GitHub Repository. Code was developed by Rachel Swallow in Python using PyTorch [17] as the machine learning framework.

A.2 GAN Architectures

A.2.1 Generator Models

```
Generator(
  (gen): Sequential(
    (0): Linear(in_features=1, out_features=7, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=7, out_features=13, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=13, out_features=7, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=7, out_features=1, bias=True)
  )
)

GeneratorLeak(
  (gen): Sequential(
    (0): Linear(in_features=1, out_features=7, bias=True)
    (1): LeakyReLU(negative_slope=0.01, inplace=True)
    (2): Linear(in_features=7, out_features=13, bias=True)
    (3): LeakyReLU(negative_slope=0.01, inplace=True)
    (4): Linear(in_features=13, out_features=7, bias=True)
    (5): LeakyReLU(negative_slope=0.01, inplace=True)
    (6): Linear(in_features=7, out_features=1, bias=True)
  )
)

Generator2(
  (gen): Sequential(
    (0): Linear(in_features=1, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=248, bias=True)
    (3): ReLU(inplace=True)
```

```
    (4): Linear(in_features=248, out_features=496, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=496, out_features=992, bias=True)
    (7): ReLU(inplace=True)
    (8): Linear(in_features=992, out_features=1, bias=True)
  )
)

Generator_z2(
  (gen): Sequential(
    (0): Linear(in_features=20, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=128, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=128, out_features=128, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=128, out_features=1, bias=True)
  )
)
```

A.2.2 Discriminator Models

```
Discriminator(
  (disc): Sequential(
    (0): Linear(in_features=1, out_features=7, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=7, out_features=13, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=13, out_features=7, bias=True)
    (5): LeakyReLU(negative_slope=0.2)
    (6): Linear(in_features=7, out_features=1, bias=True)
  ))

  Discriminator2(
  (disc): Sequential(
    (0): Linear(in_features=1, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.2)
    (2): Linear(in_features=512, out_features=256, bias=True)
    (3): LeakyReLU(negative_slope=0.2)
    (4): Linear(in_features=256, out_features=128, bias=True)
    (5): LeakyReLU(negative_slope=0.2)
```

```
    (6): Linear(in_features=128, out_features=1, bias=True)
  )
)
```

A.2.3 Encoder and Decoder Models

```
Encoder(
  (model): Sequential(
    (0): Linear(in_features=1, out_features=11, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=11, out_features=29, bias=True)
    (3): ELU(alpha=1.0)
  )
)

Decoder(
  (model): Sequential(
    (0): Linear(in_features=29, out_features=11, bias=True)
    (1): ELU(alpha=1.0)
    (2): Linear(in_features=11, out_features=1, bias=True)
  )
)
```