Risk Management with Generative Adversarial Networks

Student number: 1028447

University of Oxford

A thesis submitted in partial fulfillment of the MSc in

Mathematical and Computational Finance

June 27, 2019

Abstract

Effective risk management has become an essential task for any major financial institution. Correctly assessing the risks associated with the positions it holds is not only in an institution's own best interest but also required by the financial regulators. A particularly important risk measure is Value-at-Risk (VaR). Over the years a myriad of techniques have been developed to accurately estimate VaR. In this work we set out to apply the novel technique of Generative Adversarial Networks (GANs) to the problem of Value-at-Risk estimation. We investigate both unconditional and conditional forecasts. Despite the difficulties associated with the training of GANs we empirically show that one can obtain unconditional VaR forecasts that outperform classical unconditional statistical methods. In the conditional case the problem turns out to be a bit more complicated. However, we still obtain performance that is competitive with that of classical conditional methods. GANs therefore constitute a promising candidate for risk management in the future.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Contributions

Every day at 4:15 pm former JPMorgan Chairman Dennis Weatherstone was sitting in his office waiting for the famous "4-15 report" to be delivered [53, p. 33-34]. The number that interested him the most in that one-page document was the one-day Value-at-Risk (VaR) at the 95%-confidence level. It was so important because it allowed to break down the bank's short term risk exposure into a single number and therefore proved to be a very useful reporting tool for the management. Since its inception at JP Morgan in the late 1980s Value-at-Risk has found its way into virtually any financial institution and modern risk management without it would be unthinkable. With the publication of the Basel II accord in 2004 Value-at-Risk also became an integral part of the regulatory framework of the Basel Committee on Banking Supervision (BCBS) as a means to determine the capital buffer that banks have to set aside to cover eventual extreme losses [5].

Due to its importance for effective risk management it is crucial that the reported VaR indeed is a realiable estimate of the corresponding true quantile of the loss distribution. The choice of an appropriate model to estimate the loss distribution is therefore of utmost importance and so a myriad of methods have been proposed over the years (see [45] for an extensive survey). Most of these approaches are based on classical tools from statistics and probability theory such as maximum likelihood estimation, extreme value theory (EVT) or time series analysis. With the recent rise of machine learning came the developement of alternative approaches to the problem of distribution estimation, known as *generative models*. A particularly important instance of generative models are Generative Adversarial Networks (GANs) [21]. Instead of estimating the parameters of a closed-form probability density function from the available data they learn to generate samples following the true underlying distribution. This is achieved by playing an adversarial game between two networks, known as the generator and discriminator. The generator takes as input a random sample from some fixed distribution and applies to it a nonlinear transformation. The task of the discriminator then is to distinguish these fake samples from the real data samples. Ideally, the competition between the two networks results in a Nash equilibrium in which the samples drawn from the generator are indistinguishable from the real data samples and the discriminator can do no better than randomly guess whether a given sample is real or fake.

GANs have recently been successfully applied to a variety of tasks such as creation of high-resolution images [8], open dialog-generation in natural language processing [32] or video generation [2], just to name a few. Therefore the question arises whether they can also learn to sample from the much lower-dimensional distribution of portfolio returns and hence might constitute a credible competitor of the classical methods for Value-at-Risk estimation. Up to now, very little work has been done to answer this question. To our knowledge there exist only two blog-posts [25, 51] in which the authors address the topic on a very high level and also do not backtest their models. In the present thesis we therefore set out to further investigate this topic and improve on the current results. The main contributions are the following:

1. We prove a theorem which we term *universal approximation theorem for probability distributions.* It generalizes the well-known universal approximation theorem for multilayer feedforward neural networks [27] to the approximation of probability distributions with random variables transformed by neural networks. The result is often used to argue for the representative power of GANs but we did not find its proof anywhere and so give it for completeness.

2. We train various types of GANs to sample from the stationary distribution of the log-returns of an equity portfolio as well as from the joint distribution of the constituents and show that we can considerably improve the model fit achieved in the two aforementioned blog-posts. Furthermore, the VaR forcasts from this unconditional GAN model turn out to be more accurate than those of the classical unconditional methods which we confirm by backtesting.

3. The results are then extended to the conditional case. That is, we apply the GAN methodology to sample stock price trajectories conditional on their previous history. Taking empirical quantiles of these conditional forecasts allows us to compute VaR forecasts that are competitive with the classical time series approaches.

4. For both the conditional and unconditional case we suggest a performance metric to assess the model fit on a validation set. This performance metric us used in a grid search to find good hyperparameters of the GAN models.

1.2 Related Work

As already mentioned, little work has been done directly on the estimation of Value-at-Risk with GANs. There are, however, some papers in which the authors employ GANs to simulate equity price time series. Zhang et al. [55] and Zhou et al. [57] train a GAN type network to predict the closing prices of several stocks and indices. Both their models are competitive with other state-of-the-art machine learning methods as measured by the root-mean-squared-error. However, they are purely deterministic and so do not allow for the computation of quantiles. In [33] Li et al. use a GAN to simulate limit order book dynamics. In particular, they manage to achieve very good fits of the interarrival time distributions. The learned price distribution, on the other hand, only roughly approximates the true distribution and is therefore not suitable for VaR estimation purposes. Other authors have applied GANs to the simulation of non-financial time series. Esteban et al. [18] propose the *real-valued conditional GAN (RCGAN)* that learns to sample realizations of a multivariate real-valued time series conditional on some time window of previous observations. They apply the RCGAN to the simulation of medical data that can then be used to create realistic training scenarios for education of doctors. The GAN we propose in section 4.2 to simulate log-returns will be similar in nature to the RCGAN.

1.3 Scope and Structure

This thesis is structured as follows: In chapter 2 we first formally define Value-at-Risk and then present the most widely used estimation and backtesting methods. This is followed by an introduction to the theory of GANs in chapter 3. We start by stating and proving the aforementioned *universal approximation theorem for probability distributions* and then introduce the original GAN as proposed by Goodfellow et al. in [21] as well as one of its major extensions, the *Wasserstein GAN* by Arjovsky [3]. Finally, in chapter 4 we apply the GAN methodology to the problem of VaR estimation and compare its backtesting performance with that of the classical methods presented in chapter 2.
Note: The corresponding Python and R codes are appended to the electronic version only.

Chapter 2

Value-at-Risk

Roughly speaking, the Value-at-Risk at a confidence level α is the maximum portfolio loss that will not be exceeded within some prespecified time period with probability α. Despite its popularity VaR has some undesirable features. First, it does not contain any information about the severity of losses above the VaR level. Secondly, it is not a coherent risk measure in the sense of Artzner [4] because it is not subadditive which might discourage diversification. A risk measure which does not exhibit the aforementioned deficiencies is the expected shortfall (ES) which is defined as the expected loss above the VaR level. Even though VaR is theoretically less appealing than ES it remains the industry standard and is firmly anchored in the regulatory requirements. In the present thesis we will therefore focus on a Value-at-Risk as the measure for market risk management.
The chapter is organized as follows: We first formally define VaR and then introduce some of the most widely used financial models to estimate and backtest it. These will constitute the baseline for comparison with the GAN methodology later on.

2.1 Definition of Value-at-Risk

In the following we set up the notational framework whereby we follow [39, Chapter 2].
Consider a portfolio of risky assets and denote its value at time t by V_t. Given a prespecified time horizon Δt we write

$$L_{\text{abs}}(t; \Delta t) := -\left[V_{t+\Delta t} - V_t\right] \tag{2.1}$$

for the absolute loss of the portfolio over the time interval $[t, t + \Delta t]$. The portfolio value V_t usually is a deterministic function $V_t = f(t, Z_t)$ of time and *risk-factors* $Z_t = (Z_t^1, ..., Z_t^d)^\top$. In this thesis we will focus on equity portfolios for which the risk-factors are chosen to be the log-prices of the constituting stocks. That is, if the portfolio consists of d stocks with price processes S_t^i, $i = 1, ..., d$, the risk-factors are $Z_t^i = \log(S_t^i)$. As the portfolio value is $V_t = \sum_{i=1}^d S_t^i$, the function $f(t, z)$ is given by $f(t, z) = \sum_{i=1}^n \exp(z_i)$. In particular, $f(t, z)$ does not depend on time (as opposed to e.g. in a portfolio of stock options) and so we just write $f(z)$. Using the new notation the loss $L(t; \Delta t)$ becomes

$$L_{\text{abs}}(t; \Delta t) = -\left[f(Z_{t+\Delta t}) - f(Z_t)\right].$$

In practice, one often works with the *linearised loss* which is given by the first order Taylor approximation

$$L^{\Delta}_{\text{abs}}(t;\Delta t) = -\sum_{i=1}^{d} f_{z_i}(Z_t)X^i_t \tag{2.2}$$

where $X^i_t = Z^i_{t+\Delta t} - Z^i_t$ are the *risk-factor changes* and f_{z_i} denotes the partial derivative of f with respect to z_i. In practical applications to market risk the time horizon Δt is usually chosen to be one or ten days and so the linear approximation is reasonable as long as the risk-factors do not vary too rapidly. In our case of a stock portfolio the risk-factor changes are just the log-returns $X^i_t = \log(S^i_{t+\Delta t}) - \log(S^i_t)$ over the interval $[t, t+\Delta t]$. If we write $w_i = S^i_t/V_t$ for the portfolio weights at time t, we obtain

$$L^{\Delta}_{\text{abs}}(t;\Delta t) = -V_t\sum_{i=1}^{d} w_i X^i_t = -V_t w^\top \mathbf{X}_t$$

where we used that in our particular case $f_{z_i}(z) = \exp(z_i)$ and write w and X_t for the vectors of portfolio weights and risk-factor changes, respectively. Often we are just interested in the relative change of the position and therefore investigate the *relative loss*

$$L_{\text{rel}}(t;\Delta t) = -\frac{V_{t+\Delta t} - V_t}{V_t}. \tag{2.3}$$

If we again consider an equity portfolio and linearise the numerator as before we obtain the *linearised relative loss*

$$L^{\Delta}_{\text{rel}}(t;\Delta t) = -\sum_{i=1}^{d} w_i X^i_t = -w^\top X_t \tag{2.4}$$

which is the portfolio weighted average of the log-returns. Furthermore, let $(\mathcal{F}_t)_{t\in\mathbb{R}}$ denote a filtration to which Z_t is adapted. This can either be the σ-algebra generated by Z_t or a larger one if we want to include additional information. We can now give the definition of the Value-at-Risk.

Definition 2.1.1 ((Un)conditional absolute and relative Value-at-Risk, adapted from [39, Definition 2.10])**.** *Given a confidence level $\alpha \in (0,1)$ and a time horizon Δt, the* ***unconditional absolute Value-at-Risk*** $\mathrm{VaR}_{\text{abs}}(t,\Delta t;\alpha)$ *is defined as*

$$\mathrm{VaR}_{\text{abs}}(t,\Delta t;\alpha) = \inf\{l \mid \mathbb{P}(L_{\text{abs}}(t;\Delta t) > l) \leq 1-\alpha\}.$$

The ***unconditional relative Value-at-Risk*** $\mathrm{VaR}_{\text{rel}}(t,\Delta t;\alpha)$ *is defined analogously by replacing* $L_{\text{abs}}(t;\Delta t)$ *with* $L_{\text{rel}}(t;\Delta t)$*.*
The ***conditional absolute Value-at-Risk*** $\mathrm{VaR}^{\text{cond}}_{\text{abs}}(t,\Delta t;\alpha)$ *is defined as*

$$\mathrm{VaR}^{\text{cond}}_{\text{abs}}(t,\Delta t;\alpha) = \inf\{l \mid \mathbb{P}(L_{\text{abs}}(t;\Delta t) > l \mid \mathcal{F}_t) \leq 1-\alpha\}.$$

Again, the ***conditional relative Value-at-Risk*** $\mathrm{VaR}^{\text{cond}}_{\text{rel}}(t,\Delta t;\alpha)$ *is defined by replacing the absolute loss with its relative counterpart.*

In this thesis we will mainly focus on the relative Value-at-Risk. As is common market practice we also replace the true relative loss in the above definition with its linearised version. In order to ease notation we just write $\mathrm{VaR}_\alpha(t)$ for the relative Value-at-Risk (based on the linearised relative loss) if Δt is fixed

and if it is clear from the context whether we deal with unconditional or unconditional VaR. Similarly, we just write L_t for the linearised realitive loss and when speaking of a loss distribution in the following we always mean the distribution of the linearised relative loss over some fixed time horizon Δt. We consider both the conditional and unconditional case.
In applications, time is usually discretised. That is, if the time horizon is Δt we investigate the time series $L_t = L(t \cdot \Delta t; \Delta t)$ for $t \in \mathbb{Z}$. This ensures that the intervals over which we compute the losses do not overlap.

2.2 Classical VaR Estimation Methodology

From its very definition it is clear that VaR is a quantile of the (un)conditional loss distribution. Therefore, the main task in estimating VaR is to obtain reliable estimates of that distribution. In particular, the tail behaviour should be accurately captured.
Over the years, many different methods for VaR estimation have been proposed an extensive survey of which can be found in [45]. We broadly classify the approaches using the following criteria:

1. **Unconditional or conditional:** Unconditional modelling approaches rely on the assumption that the risk-factor changes or portfolio losses follow a stationary time series. The goal then is to estimate the corresponding stationary distribution and to compute the VaR by taking the corresponding quantile. Hence, unconditional approaches do not incorporate any dynamic behaviour of the underlying risk-factors. Conditional approaches, on the other hand, take the history of the time series into account by conditioning on the corresponding filtration. They therefore usually give more accurate forecasts.

2. **Parametric or Non-parametric:** Parametric approaches to VaR estimation use a parametric family to model the risk-factor changes or the portfolio loss (depending on the level of aggegration; see next point). The VaR can then be estimated by taking the theoretical quantile of the fitted distribution. Non-parametric models, on the other hand, rely on tools from non-parametric statistics such as kernel density estimation. In the simplest case they just amount to taking an empirical quantile.

3. **Modelling on risk-factor or portfolio level:** The modelling can either be done on risk-factor level or on the aggregate portfolio level.

In the following we will discuss the most prominent methods for VaR estimation and classify them according to the above criteria. These more classical models will then constitute the baseline for comparison with the GAN methodology later on.

2.2.1 Unconditional Methods

We first discuss parametric unconditional methods in section 2.2.1.1 and then focus on non-parametric unconditional ones in section 2.2.1.2. If not stated otherwise we follow [39] in this chapter.

2.2.1.1 Parametric Unconditional Methods

Parametric unconditional methods fit a single parametric distribution either to the risk-factors or the portfolio losses depending on the level of aggregration. However, most of the existing models concentrate

on modelling the risk-factor distributions and then aggregate these distributions using the portfolio weights to arrive at the portfolio loss distribution.
In the following we discuss two very common models from this category. The first one, commonly known as the Variance-Covariance method, fits a multivariate normal distribution to the log-returns. As is widely known, financial return data usually exhibits fat tails and so the normal distribution might not be appropriate. We therefore introduce a multivariate t-distribution as an alternative. Of course, one can propose any multivariate distribution or a copula to model the joint behaviour of the risk-factors. However, if the distribution function of a linear affine transformation of the proposed risk-factor distribution cannot be computed in closed form, it might be difficult to aggregrate the log-returns to a portfolio loss. The multivariate normal and t-distribution are very appealing in this regard as they are closed under linear transformations which facilitates the computation of portfolio loss quantiles. In the presentation of the Variance-Covariance and t-distribution method we follow [39, Chapter 2].

Variance-Covariance Method

The Variance-Covariance method is built on the assumption that the log-returns $X = (X_1, ..., X_d)^\top$ are multivariate normal with mean vector μ and covariance matrix Σ. If the portfolio weights are $w = (w_1, ..., w_d)^\top$, the portfolio return is given by

$$R = w^\top X \sim \mathcal{N}(w^\top \mu, w^\top \Sigma w). \tag{2.5}$$

It is clear that the linearised relative loss of the portfolio as defined in (2.4) is $L = -R \sim \mathcal{N}(-w^\top \mu, w^\top \Sigma w)$. The Value-at-Risk can now easily be computed to be

$$\mathrm{VaR}_\alpha = -w^\top \mu + \sqrt{w^\top \Sigma w}\ \Phi^{-1}(\alpha) \tag{2.6}$$

where $\Phi^{-1}(\cdot)$ denotes the inverse cdf of a standard normal. In practice, the mean vector μ and covariance matrix Σ are replaced by their empirical counterparts which can be obtained from a maximum likelihood estimation.

Multivariate t-distribution

In order to capture fat tails in the asset returns one can replace the normal distribution with a t-distribution. For a location vector $\mu \in \mathbb{R}^d$ and a scale matrix $\Sigma \in \mathbb{R}^{d\times d}$ the density of a d-dimensional t-distribution with ν degrees of freedom is given by

$$f(x) = \frac{\Gamma((\nu+d)/2)}{\Gamma(\nu/2)\nu^{d/2}\pi^{d/2}|\Sigma|^{1/2}} \left[1 + \frac{1}{\nu}(x-\mu)^\top \Sigma^{-1}(x-\mu)\right]^{-(\nu+d)/2} \tag{2.7}$$

and we write $X \sim t_d(\nu, \mu, \Sigma)$. It can be verified that $\mathbb{E}[X] = \mu$ and $\mathrm{Cov}(X) = \frac{\nu}{\nu-2}\Sigma$. The multivariate t-distribution belongs to the class of elliptical distributions and can also be understood as a variance mixture distribution. As such it is closed under linear transformations, i.e. for $A \in \mathbb{R}^{k\times d}$ and $b \in \mathbb{R}^k$ it holds

$$AX + b \sim t_k(\nu, A\mu + b, A\Sigma A^\top),$$

see [39, Proposition 3.9]. In particular, for the portfolio return one obtains

$$R = w^\top X \sim t_1(\nu, w^\top \mu, w^\top \Sigma w). \tag{2.8}$$

From this the Value-at-Risk can be computed to be

$$\mathrm{VaR}_\alpha = -w^\top \mu + \sqrt{w^\top \Sigma w}\; t_\nu^{-1}(\alpha) \tag{2.9}$$

where $t_\nu^{-1}(\cdot)$ denotes the inverse cdf of a univariate t-distribution with ν degrees of freedom. In practice, one again replaces the true values for μ, Σ and ν by their empirical counterparts which are usually estimated using an Expectation Maximization (EM) based approach. In our applications we will use the ECME algorithm by Liu and Rubin [35]. Unfortunately, the available Python packages do not contain a function to automate the fitting. We therefore had to implement the algorithm ourselves. In appendix B we present the ECME algorithm and then provide the corresponding Python code in appendix **??**. The code has also been made available at Github for the convenience in future applications.

2.2.1.2 Non-parametric Unconditional Methods

Whereas parametric unconditional methods usually operate on the risk-factors, non-parametric unconditional approaches mostly work directly with portfolio losses. We introduce two very simple but often used methods.

Historical Simulation

The historical simulation method is probably the most simplistic of all. Yet, it is still widely used within banks due to its ease of implementation.
The idea behind the historical simulation approach is very simple: One looks at how the current portfolio would have performed in the past and uses these simulated losses to obtain an empirical distribution function for the portfolio loss. The corresponding quantile of that empirical cdf then serves as the VaR estimate.
More formally, let today be time $t = t_n$ and X_{t_i}, $i = 1, ..., n$, historical observations of the risk-factor changes (log-returns in our case). Further, let $w = (w_1, ..., w_d)^\top$ denote the current portfolio weights. Then construct the simulated historical losses via

$$L_i = -w^\top X_{t_i}, \quad i = 1, ..., n. \tag{2.10}$$

Now let $L_{1,n} \geq ... \geq L_{n,n}$ be the corresponding order statistics. From these the VaR can be estimated by

$$\widehat{\mathrm{VaR}}_\alpha = L_{m,n} \quad \text{where} \quad m = \lfloor N(1-\alpha) \rfloor. \tag{2.11}$$

Kernel density approximation

Another unconditional non-parametric approach is to fit a kernel density estimator (KDE) [22] to the simulated historical losses in (2.10) and to take the corresponding quantile thereof as the VaR estimate. That is, we choose some symmetric *kernel* $K(x)$ satisfying $K(x) \geq 0$, $K(x) = K(-x)$ and $\int_{\mathbb{R}} K(x)\mathrm{d}x = 1$

and estimate the pdf of the loss distribution by

$$f_L(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - L_i}{h}\right) \tag{2.12}$$

for some *bandwidth h*. In our applications we use the *Gaussian kernel* $K(x) = \frac{1}{\sqrt{2\pi}}\exp\left(-\frac{x^2}{2}\right)$ and choose the bandwidth according to Silverman's rule of thumb [52, p. 45]. That is, we first compute the empirical standard deviation $\hat{\sigma}$ and then set $h = \left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}}$.

2.2.2 Conditional Methods

Conditional approaches to VaR estimation usually rely on time series modelling. Especially popular are the *Generalised Autoreressive Conditional Heteroskedasticity (GARCH)* models introduced by Engle [17] and Bollerslev [7]. As before, the modelling can either be done on the risk-factor or portfolio level. In the former case one would need to make use of one of the many multivariate extensions of the GARCH model. In our analyses we will use the *Dynamic Conditional Correlation (DCC)* GARCH model by Engle and Shepard [16] which we introduce in the following whereby we assume knowledge of standard univariate GARCH theory (see e.g. Tsay [54]).
We first need to introduce the notion of a **strict white noise process**.

Definition 2.2.1 (Strict white noise, adapted from [39, Definition 4.34])**.** *A d-dimensional time series* $(Z_t)_{t\in\mathbb{Z}}$ *is a* ***strict white noise*** *if it is a sequence of i.i.d. random vectors with finite covariance matrix. If* μ *and* Σ *are the mean vector and covariance matrix, respectively, we write* $Z \sim \mathrm{SWN}(\mu, \Sigma)$.

Using the definition of a strict white noise process we can define a general multivariate GARCH process:

Definition 2.2.2 (Multivariate GARCH, adapted from [39, Definition 4.39])**.** *A d-dimensional time series* $(X_t)_{t\in\mathbb{Z}}$ *is a* ***multivariate GARCH process*** *if it is strictly stationary and satisfies*

$$X_t = \Sigma_t^{1/2} Z_t \tag{2.13}$$

where $Z \sim \mathrm{SWN}(0, I_d)$ *and the* ***conditional covariance matrix*** Σ_t *is measurable with respect to* $\mathcal{F}_{t-1} := \sigma(\{X_s : s \leq t-1\})$. *In* (2.13) $\Sigma_t^{1/2}$ *denotes any matrix for which* $\Sigma_t^{1/2}\left(\Sigma_t^{1/2}\right)^\top = \Sigma_t$, *e.g. the Cholesky decomposition.*

Before we state the definition of a DCC GARCH process we have to introduce the following operators acting on a positive definite matrix $\Sigma \in \mathbb{R}^{d\times d}$:

$$\Delta(\Sigma) = \mathrm{diag}(\sqrt{\Sigma_{1,1}}, ..., \sqrt{\Sigma_{d,d}})$$
$$\mathcal{P}(\Sigma) = \Delta(\Sigma)^{-1}\Sigma\Delta(\Sigma)^{-1}$$

If the matrix Σ is a covariance matrix then it is easy to see that $\mathcal{P}(\Sigma)$ is the corresponding correlation matrix and $\Delta(\Sigma)$ contains the standard deviations of the components on the diagonal. We are now ready to state the definition of the process of interest.

Definition 2.2.3 (DCC(P, Q)-GARCH process, adapted from [39, Definition 4.42])**.** *A d-dimensional GARCH process* $(X_t)_{t\in\mathbb{Z}}$ *as defined in definition 2.2.2 is a* ***DCC*(P, Q)*-GARCH process*** *if the con-*

ditional covariance matrix Σ_t can be written as

$$\Sigma_t = \Delta_t P_t \Delta_t$$

where

1. *Δ_t is the diagonal matrix of volatilities $\sigma_{t,k}$ which satisfy the univariate GARCH equations:*

$$\sigma_{t,k}^2 = \alpha_{k0} + \sum_{i=1}^{p_k} \alpha_{ki} X_{t-i,k}^2 + \sum_{j=1}^{q_k} \beta_{kj} \sigma_{t-j,k}^2$$

where $\alpha_{k0} > 0$, $\alpha_{ki} \geq 0$, $i = 1, ..., p_k$ and β_{kj}, $j = 1, ..., q_k$; and

2. *the conditional correlation matrix P_t satisfies*

$$P_t = \mathcal{P}\left(\left(1 - \sum_{i=1}^{P} a_i - \sum_{j=1}^{Q} b_j\right) P_c + \sum_{i=1}^{P} a_i Y_{t-i} Y_{t-i}^\top + \sum_{j=1}^{Q} b_j P_{t-j}\right)$$

with a positive definite matrix P_c and the devolatised process $Y_t = \Delta_t^{-1} X_t$. The constants a_i and b_j statisfy $a_i \geq 0$, $i = 1, ..., P$, b_j, $j = 1, ..., Q$ and

$$\sum_{i=1}^{P} a_i + \sum_{j=1}^{Q} b_j < 1.$$

From the above definition it can be seen that the individual components of X_t follow univariate GARCH(p_k, q_k) processes and are coupled with one another via the conditional correlation matrix P_t which can easily be checked to be positive definite. It can further be seen that the matrix P_c is the unconditional correlation matrix of the process. The DCC process is relatively parsimonious with a total of $\sum_{k=1}^{d}(1 + p_k + q_k) + (1 + P + Q) + d^2$ parameters. Yet, it still allows to model a variety of volatility structures.

The parameters of the process can be estimated using a maximum likelihood approach. In practice however, one usually resorts to a two step procedure in which one first estimates the univariate GARCH models and the estimates the correlation matrix parameters using the devolatised series. The details of this procedure can be found in appendix C.

It remains to specify the distribution of the strict white noise process Z_t. In our applications we will consider the multivariate normal $Z_t \sim \mathcal{N}(0, I_d)$ and the multivariate t-distribution $Z_t \sim t_d(\nu, 0, \frac{\nu-2}{\nu} I_d)$ normed to have identity covariance matrix.

When we use a DCC-GARCH process to model log-returns we implicitely assume a constant mean of zero. As the time periods under consideration are usually very short this is a reasonable assumption. However, if necessary we can model the conditional mean with a *Vector Autoregressive Moving Average (VARMA)* (see e.g. Tsay [54]) process resulting in a VARMA(P_1, Q_1)-DCC(P_2, Q_2)-GARCH process. Hence, if μ_t denotes the conditional mean modelled by the VARMA part of the model the VaR of the portfolio is given by

$$\widehat{\text{VaR}}_t = -\mu_t + w^\top \Sigma_t w \ \Phi^{-1}(\alpha)$$

if the strict white noise process is a multivariate standard normal and

$$\widehat{\mathrm{VaR}}_t = -\mu_t + w^\top \Sigma_t w \; t_\nu^{-1}(\alpha)$$

if it is a multivariate t-distribution.

2.3 Backtesting VaR

Once a method for the estimation of Value-at-Risk has been chosen and implemented, the goodness of the model fit has to be assessed. The procedure of doing so is known as *backtesting* and usually consists of a series of hypothesis tests peformed on data the model has not been calibrated on. The following section is based on [56].
In VaR terminology an *exception* occurs if the actual loss/return exceeds the VaR estimate of the model. By the definition of VaR, the fraction of days on which an exception occurs should not significantly differ from the theoretical VaR confidence level. Furthermore, exceptions should not cluster in time, i.e. they should be independent events.
Over the years several hypothesis tests for testing the above two assumptions have been proposed an extensive survey of which can be found in [56]. The tests proposed in the literature can be broadly classified into *coverage* and *independence tests* and adress the two desired properties of exceptions described above. Coverage tests compare the number of observed exceptions with the expected number thereof. Independence tests, on the other hand, check if exceptions cluster in time. Before we present the most prominent tests of both categories we introduce the notation we will use.
Suppose a model is in place which gives VaR estimate $\widehat{\mathrm{VaR}}_\alpha(t)$ for the time period t. As before, denote by L_t the actual observed loss in period t. We backtest the model on periods $t \in \{1, ..., T\}$. To do so let $I_t := \mathbb{1}_{\{\widehat{\mathrm{VaR}}_\alpha(t) < L_t\}}$ denote the indicator function of an exception in period t and write $M = \sum_{i=1}^{\top} I_t$ for the total number of exceptions during the backtesting period. If the model were correct the random variables I_t would be Bernoulli distributed with success probability $1 - \alpha$. In the following three very prominent tests will be presented.

Kupiec's Proportion of Failure Test (POF)

In 1995 Kupiec [31] introduced the *Proportion of Failure Test* (POF) which is based on likelihood ratios. The test statistic is given by

$$LR_{\mathrm{POF}} = -2 \log \left(\frac{(1-\alpha)^M \alpha^{T-M}}{\left(1 - \frac{M}{T}\right)^{T-M} \frac{M}{T}^M} \right).$$

Under the null hypothesis that the model is correct the test statistic is asymptotically χ^2-distributed with one degree of freedom. A model will therefore be rejected if $LR_{\mathrm{pof}} \geq (\chi^2)^{-1}(p)$ where p is the confidence level of the test and $(\chi^2)^{-1}(\cdot)$ denotes the quantile function of the χ^2-distribution with one degree of freedom. Clearly, the POF test only tests coverage and not independence.

Christofferson Interval Forecast Test (IF)

The *Christofferson Interval Forecast Test* [11] measures whether an exception on the previous day has an influence on the probability of an exception on the next day and hence tests for independence. Let us introduce the following random variables:

- N_{00}: Number of periods with no exception followed by a period with no exception
- N_{01}: Number of periods with no exception followed by a period with an exception
- N_{10}: Number of periods with an exception followed by a period with no exception
- N_{11}: Number of periods with an exception followed by a period with an exception

The test statistic of the IF is then given by the likelihood-ratio

$$LR_{\text{IF}} = -2\log\left(\frac{(1-\pi)^{N_{00}+N_{10}}\pi^{N_{01}+N_{11}}}{(1-\pi_0)^{N_{00}}\pi_0^{N_{01}}(1-\pi_1)^{N_{10}}\pi_1^{N_{11}}}\right)$$

where the probabilities π, π_0 and π_1 are given by

- $\pi = \frac{N_{01}+N_{11}}{N_{00}+N_{01}+N_{10}+N_{11}}$, i.e. π is the estimated probability of an exception occuring conditional on no further information,
- $\pi_0 = \frac{N_{01}}{N_{00}+N_{01}}$, i.e. π_0 is the estimated probability of an exception occuring conditional on no exception in the previous period,
- $\pi_1 = \frac{N_{11}}{N_{10}+N_{11}}$, i.e. π_1 is the estimated probability of an exception occuring conditional on an exception in the previous period.

Under the null hypothesis the test statistic is asymptotically χ^2-distributed with one degree of freedom and we reject the model if $L_{\text{IF}} \geq (\chi^2)^{-1}(p)$. The IF test can be combined with the POF test in order to both check whether the fraction of exceptions is reasonable and whether there is no clustering of exceptions. The test statistic of the resulting *conditional coverage test* (CC) is just the sum of the single statistics, i.e. $L_{\text{CC}} = L_{\text{POF}} + L_{\text{IF}}$, and one can show that L_{CC} is χ^2-distributed with two degress of freedom.

Haas's Time Between Failures Test (TBF)

Haas [24] proposed the *Time Between Failures Test* (TBF) which is based on the fact that the number of periods between exceptions should be independent and geometrically distributed with parameter $1-\alpha$ under the null hypothesis. The test statistic is again based on likelihood ratios and given by

$$LR_{\text{tuff}} = -2\sum_{i=1}^{M}\log\left(\frac{\alpha^{N_i-1}(1-\alpha)}{\frac{1}{N_i}\left(1-\frac{1}{N_i}\right)^{N_i-1}}\right)$$

where M denotes the number of exceptions and N_i the number of periods between the $(i-1)$-th and i-th exception. Under the null hypothesis this statistic is χ^2-distributed with M degrees of freedom and we reject a model if $L_{\text{tuff}} \geq (\chi^2_M)^{-1}(p)$. The TBF tests for both independence and coverage.

Chapter 3

Generative Adversarial Networks

In this chapter we introduce the basics underlying Generative Adversarial Networks (GANs). The outline is as follows: In section 3.1 we first introduce the basic idea underlying GANs and then proof a theorem which we refer to as the *universal approximation theorem for probability distributions*. This theorem guarantees that virtually any random variable can be approximated arbitrarily closely in distribution with a GAN. We provide the proof as we could not find it elsewhere even though the result is often used in theoretical arguments. In section 3.2 we present several GAN objective functions. In particular, we present the original GAN [21] and one of its most prominent extensions, the so-called *Wasserstein-GAN* (WGAN) [3]. Finally we cover the details of the training and problems associated with it in section 3.3.

3.1 Universal Approximation of Probability Distributions

The basic idea underlying all types of GANs is to draw a sample from a *latent random variable* Z taking values in $\mathbb{R}^d$, called the *latent space*, and to then map it into a (usually higher dimensional) space $\mathbb{R}^n$, called the *data space*, by applying to it a function $G : \mathbb{R}^d \to \mathbb{R}^n$. The set of available transformation functions is usually chosen to be a rich class of neural networks with weights $\theta \in \Theta$ and denoted by $\mathcal{G}_\Theta = \{G_\theta(\cdot) : \theta \in \Theta\}$. One then writes $\mathbb{P}_\theta$ for the distribution of the transformed random variable $G_\theta(Z)$ and denotes the class of resulting probability measures by $\mathcal{P}_\Theta(\mathbb{R}^n) = \{\mathbb{P}_\theta : \theta \in \Theta\}$. Naturally, the question arises how well the transformed variables $G_\theta(Z)$ can approximate an arbitrary random variable X with values in the data space. It is widely known [27] that a sufficiently large class of neural networks can approximate any continuous function arbitrarily well on compacta and hence it is reasonable to suspect that this can be extended to the approximation of probability distributions with random variables transformed by neural networks. In the following we prove a theorem which we call the *universal approximation theorem for probability distributions* that tells us that this is indeed possible. Before stating the theorem we have to introduce the kind of admissible activation functions and also recall the universal approximation theorem.

Definition 3.1.1 (Squashing function [27, Definition 2.3]). *A nondecreasing function $\phi : \mathbb{R} \to [0, 1]$ is called **squashing function** if $\lim\limits_{\lambda \to -\infty} \phi(\lambda) = 0$ and $\phi(\lambda) = 1$.*

The universal approximation theorem now states that neural networks with one hidden layer are universal function approximators.

Theorem 3.1.1 (Universal approximation theorem [27, Theorem 2.4]). *Let ϕ be a squashing function, $K \in \mathbb{R}^n$ a compact set and $f : \mathbb{R}^n \to \mathbb{R}^m$ a continuous function. Then, for every $\varepsilon > 0$ there exists $N \in \mathbb{N}$, matrices $W_1 \in \mathbb{R}^{N\times n}$ and $W_2 \in \mathbb{R}^{m\times N}$ and a bias vector $b \in \mathbb{R}^N$ such that the function*

$$F : \mathbb{R}^n \to \mathbb{R}^m, \quad x \mapsto W_2\phi(W_1 x + b) \tag{3.1}$$

satisfies

$$\sup_{x\in K} \|F(x) - f(x)\| < \varepsilon. \tag{3.2}$$

In (3.1) the application of ϕ has to be understood componentwise. The statement still holds true when there is more than one hidden layer.

Using theorem 3.1.1 we can show that any continuous random variable X taking values in a d-dimensional differentiable manifold M can be approximated arbitrarily well in distribution by a sequence $G_{\theta_n}(Z)$ of transformed random variables where $G_{\theta_n}(\cdot)$ are feedforward neural networks with weights θ_n and Z is a random variable that takes values in $\mathbb{R}^d$ and is absolutely continuous with respect to the Lebesgue measure.
Before we state the theorem we have to make precise what is meant when we say that X is continuous random variable on a manifold. We assume basic knowledge of vector calculus (for details see e.g. [37]). For simplicity we assume that the manifold $M \subset \mathbb{R}^n$ can be parametrised by a single continuously differentiable chart $T : \mathbb{R}^d \supset \Omega \to \mathbb{R}^n$. The arguments in the following proof can easily be extended to manifolds parametrised by an atlas of charts. We say that X is a continuous random variable on M if there exists a function $f : M \to \mathbb{R}^+$ such that

$$\begin{aligned}\mathbb{P}(X \in A) &= \int_{M\cap A} f(x)\mathrm{d}S(x)\\ &= \int_{\Omega} f(T(x))\mathbb{1}_{\{x:T(x)\in A\}}\sqrt{\det\left(DT^T(x)DT(x)\right)}\mathrm{d}x.\end{aligned}$$

We can now formulate the universal approximation theorem for probability distributions the proof of which can be found in appendix A.

Theorem 3.1.2 (Universal Approximation of Probability Distributions). *Let Z be a continuous random variable taking values in $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ and X a continuous random variable on a d-dimensional differentiable manifold in $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ where $n \geq d$. Then there exists a sequence of neural networks $G_{\theta_n} : \mathbb{R}^d \to \mathbb{R}^n$ with weights θ_n such that $G_{\theta_n}(Z)$ converges to X in distribution.*

3.2 GAN Objective Functions

From the last section we know that there exists a sequence of neural networks $G_{\theta_n}(\cdot)$ such that the transformed variables $G_{\theta_n}(Z)$ converge to X in distribution. In practice, however, we need to actually find that sequence. We therefore need a criterion that measures how well the generator distribution approximates the data distribution. Because we are mainly interested in convergence in distribution, minimisation of the proposed criterion should ideally imply convergence in distribution.
There are essentially two approaches to GAN training: game-theoretic approaches and approaches based on distance minimisation. Often, a game-theoretic approach also admits an interpretation as a

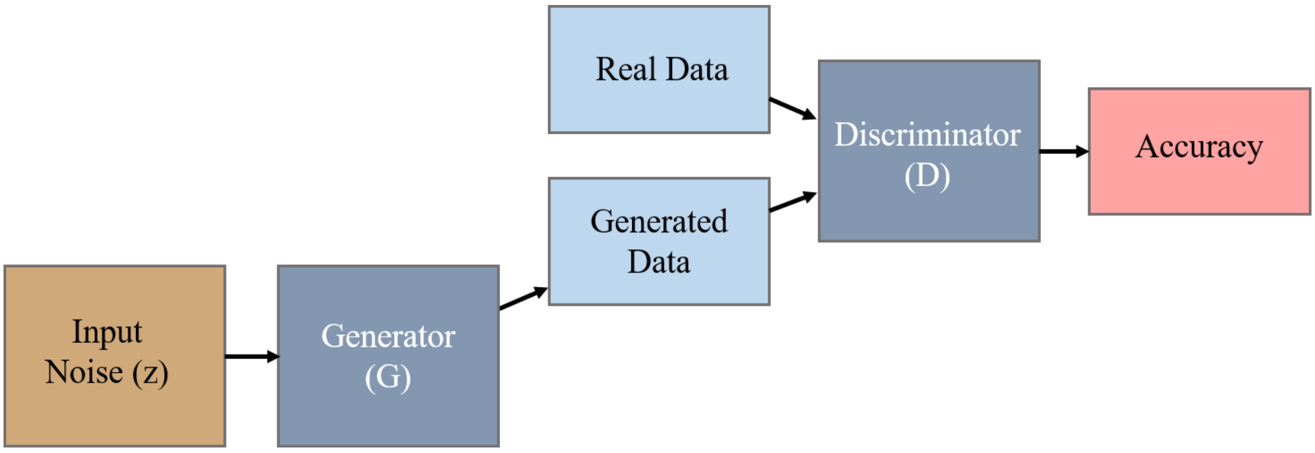

Figure 3.1: Generic GAN architecture (Adapted from [36, p. 2])

distance-minimisation approach and vice versa. In subsection 3.2.1 we first discuss the game-thereotic approach as the original GAN paper [21] was based on it. Approaches based on distance minimisation are then investigated in subsection 3.2.2. Finally, the actual implementation and associated problems are discussed in section 3.3.

3.2.1 Game-theoretic approaches

In the original GAN paper [21] the generator network G is trained by playing an adversarial game with a second network D known as the *discriminator*. The authors use the following analogy: The generator can be thought of as a counterfeiter who produces fake money. The discriminator, on the other hand, plays the role of the police and seeks to identify the forged money. In the beginning, the fake money is usually of bad quality and so the police can easily detect it. This in turn incentivizes the counterfeiter to produce even more realistically looking money. In the long run, this competition between the counterfeiter and the police is expected to result in an equilibrium in which the counterfeiter produces money that is indistinguishable from real money and the police can do no better than randomly guessing whether a given bank note is fake or not.
In the classical GAN framework the generator G works as described in the previous section: It takes a sample Z from some fixed distribution and transforms it into a random variable $G(Z)$. The discriminator, on the other hand, is a binary classifier which tries to distinguish between samples generated by the generator and real data samples. The loss function of the discriminator is the usual binary crossentropy and so the GAN objective is

$$\min_G \max_D V(D,G) = \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(1-D(G(z)))]. \tag{3.3}$$

Figure 3.1 visualises this setup. In the general formulation (3.3) the generator and discriminator functions are not yet restricted to classes of neural networks. This allows for theoretical investigation of the objective (3.3). The following theorem gives the optimal discriminator for a fixed generator.

Theorem 3.2.1 (Optimal Discriminator [21, Proposition 1])**.** *For any fixed generator G the optimal discriminator D^*_G is given by the Radon-Nikodym derivative*

$$D^*_G : \mathbb{R}^n \to [0,1], \quad x \mapsto \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\left(\mathbb{P}_{\text{data}} + \mathbb{P}_G\right)}(x)$$

of $\mathbb{P}_{\text{data}}$ *with respect to* $\mathbb{P}_{\text{data}} + \mathbb{P}_G$ *where* $\mathbb{P}_G$ *denotes the distribution of* $G(Z)$.

Proof. Define the probability measure $\mathbb{P}_m := \frac{\mathbb{P}_{\text{data}}+\mathbb{P}_G}{2}$. Both $\mathbb{P}_{\text{data}}$ and $\mathbb{P}_G$ are absolutely continuous with respect to $\mathbb{P}_m$ and hence there exist Radon-Nikodym derivatives f_1 and f_2 of $\mathbb{P}_{\text{data}}$ and $\mathbb{P}_G$ with respect to $\mathbb{P}_m$. It is also clear that these Radon-Nikodym derivatives are non-zero and sum to one. The objective $V(G,D)$ can then be rewritten as follows:

$$\begin{aligned} V(G,D) &= \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(1-D(G(z)))] \\ &= \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log(D(X))] + \mathbb{E}_{X\sim\mathbb{P}_G}[\log(1-D(X))] \\ &= \mathbb{E}_{X\sim\mathbb{P}_m}[f_1(X)\log(D(X))] + \mathbb{E}_{X\sim\mathbb{P}_m}[f_2(X)\log(1-D(X))] \\ &= \mathbb{E}_{X\sim\mathbb{P}_m}[f_1(X)\log(D(X)) + f_2(X)\log(1-D(X))] \end{aligned} \tag{3.4}$$

For any $(a,b) \in \mathbb{R}^2 \setminus \{(0,0)\}$ the function $x \mapsto a\log(x) + b\log(1-x)$ achieves its unique maximum in $[0,1]$ at $x = \frac{a}{a+b}$ which gives us the desired result by maximising the integrand in (3.4) pointwise. $\square$

Inserting the optimal discriminator D_G^* into the objective reduces it to

$$\begin{aligned} C(G) &:= \max_D V(G,D) \\ &= \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\left(\mathbb{P}_{\text{data}}+\mathbb{P}_G\right)}\right)\right] + \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\left(\mathbb{P}_{\text{data}}+\mathbb{P}_G\right)}\right)\right]. \end{aligned} \tag{3.5}$$

Before we proceed with the optimisation we need to recall the definition of the Kullback–Leibler divergence and introduce the Jensen–Shannon divergence.

Definition 3.2.1 (Kullback–Leibler (KL) divergence, adapted from [30]). *Let μ and ν be two probability measures with μ being absolutely continuous with respect to ν. The* ***Kullback–Leibler (KL) divergence*** *between μ and ν is defined as*

$$d_{\text{KL}}(\mu\|\nu) = \mathbb{E}_\mu\left[\log\left(\frac{\mathrm{d}\mu}{\mathrm{d}\nu}\right)\right].$$

An application of Jensen's inequality shows that the KL-divergence is nonnegative and vanishes if and only if μ and ν coincide. However, it is not symmetric and does not satisfy the triangle inequality and therefore does not define a proper probability metric. Another weakness is the required absolute continuity of μ with respect to ν, which usually cannot be assumed in practice. The Jensen–Shannon divergence solves this problem.

Definition 3.2.2 (Jensen–Shannon (JS) divergence, adapted from [3, p. 4]). *The* ***Jensen–Shannon (JS) divergence*** *between two measures μ and ν is defined as*

$$d_{\text{JS}}(\mu\|\nu) = \frac{1}{2}d_{\text{KL}}\left(\mu\middle\|\frac{\mu+\nu}{2}\right) + \frac{1}{2}d_{\text{KL}}\left(\nu\middle\|\frac{\mu+\nu}{2}\right).$$

In [15] it was proven that the square root of then JS-divergence indeed defines a probability metric. In particular, we can make use of the positive-definiteness to further simplify the GAN objective (3.5):

Theorem 3.2.2 (Optimal Generator [21, Theorem 1]). *The function $C(G)$ achieves its global optimum at G^* if $\mathbb{P}_{\text{data}} = \mathbb{P}_{G^*}$. In this case it holds $V(G^*, D^*_{G^*}) = -\log(4)$ and $D^*_{G^*} = \frac{1}{2}$.*

Proof. Using the notation $\mathbb{P}_m = \frac{\mathbb{P}_{\text{data}}+\mathbb{P}_G}{2}$, we can simplify the objective as follows:

$$\begin{aligned} C(G) &= \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\left(\mathbb{P}_{\text{data}}+\mathbb{P}_G\right)}\right)\right] + \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\left(\mathbb{P}_{\text{data}}+\mathbb{P}_G\right)}\right)\right] \\ &= \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right)\right] + \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\mathbb{P}_m}\right)\right] - \log(4) \\ &= 2d_{\text{JS}}(\mathbb{P}_{\text{data}}\|\mathbb{P}_G) - \log(4) \end{aligned}$$

The positive definiteness of the JS-divergence then implies the desired result. □

This is the result that we depicted in the beginning: The counterfeiter indeed wins the game and produces realistic bank notes and the police can only randomly guess whether a given note is fake or not. It also shows that a game-theoretic approach can often be interpreted as a distance minimisation approach as we end up minimising the JS-divergence.
In order to put the above theory in a more game-theoretic framework and to be able to compare it with other game-theoretic approaches that do not permit an interpretation as a distance minimisation problem, we introduce the necessary concepts from game theory whereby we follow [47]. We start by giving the definition of a strategic game and a pure strategy.

Definition 3.2.3 (Strategic game, pure strategy). *A **strategic game** is a tuple $(\mathcal{D}, \{S_i\}_{i=1}^n, \{u_i\}_{i=1}^n)$ where $\mathcal{D} = \{1, ..., n\}$ is the set of players, S_i the set of **pure strategies** for player i and $u_i : \mathcal{S} \to \mathbb{R}$ the payoff function for player i which is defined on the set of pure strategy profiles $\mathcal{S} = S_1 \times ... \times S_n$. If the number of players and the available strategies are finite, we say that it is a finite game.*

One of the most important concepts in game-theory is that of a Nash equilibrium which we define next.

Definition 3.2.4 (Pure Nash equilibrium). *A pure strategy profile $s = (s_1, ..., s_n) \in \mathcal{S}$ is in a **Nash equilibrium** if*

$$u_i(s_1, ..., s_i, ..., s_n) \geq u_i(s_1, ..., s_i', ..., s_n) \quad \forall s_i' \in S_i.$$

In the GAN setting we are only interested in games involving two players - the generator and the discriminator. Usually these games are **zero-sum games**.

Definition 3.2.5 (Zero-sum game). *A strategic game between two players is called a **zero-sum game** if $u_1(s_1, s_2) = -u_2(s_1, s_2)$ for all $s_1 \in S_1, s_2 \in S_2$.*

In particular, the game in equation (3.3) is a zero-sum game with payoff functions

$$u_D(G, D) = \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log(D(X))] + \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(1 - D(G(Z)))] \quad \text{and} \tag{3.6}$$

$$u_G(G, D) = -u_D(G, D). \tag{3.7}$$

Furthermore, it is easy to see that the derived optimal generator-discriminator pair $(G^*, D^*) = (\mathbb{P}_{\text{data}}, \frac{1}{2})$ (we identify the generator with the resulting distribution) is in a Nash equilibrium. Hence, we showed that minimizing the Jensen–Shannon divergence between the data and generator distribution amounts to finding a Nash equilibrium in the corresponding two-player zero-sum game. In the following we call this GAN the **Minimax-GAN (MM-GAN)**. We will now contrast it with a second GAN that

the authors introduce in the original GAN paper [21, 19], called the **non-saturating GAN (NS-GAN)**. The NS-GAN only admits a game-theoretic interpretation which is strategically equivalent (see definition 3.2.6) to the game underlying the MM-GAN.
The reason for introducing this second type of GAN is the following serious weakness of the Minimax-GAN in practice: In the beginning of the training process, when the generator performs poorly, the discriminator can almost perfectly classify samples as real or fake. This means that gradients in the optimisation procedure (see section 3.3) are taken at points where the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ in the loss of the discriminator is very flat which in turn results in very small gradient descent updates to the generator. This problem is widely known as *vanishing gradient* and will be further discussed in 3.3.2 when we focus on the implementation of the models. To remedy this problem the authors suggest to change the objective of the generator from minimising $\mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(1-D(G(Z)))]$ to instead maximising $\mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(D(G(Z)))]$ as this gives stronger gradients and also updates into the desired direction. This means that the payoff function of the generator in the adversarial game between the generator and discriminator becomes

$$u_G(D,G) = \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(D(G(Z)))]$$

while the payoff function of the discriminator stays the same as in (3.6). It is clear that this is not a zero-sum game anymore and it does not provide an obvious interpretation as a distance minimisation objective.
Naturally, the question arises whether these two games are equivalent in some appropriate sense. We will investigate them for **strategic equivalence**.

Definition 3.2.6 (Strategic equivalence)**.** *Two strategic games $(\mathcal{D}, \{S_i\}_{i=1}^n, \{u_i\}_{i=1}^n)$ and $(\mathcal{D}, \{S_i\}_{i=1}^n, \{\bar{u}_i\}_{i=1}^n)$ are called **strategically equivalent** if the set of Nash equilibria is the same.*

It is now very easy to see that both games are strategically equivalent. However, this has not been shown in the original paper [21] and therefore we formulate the result as a lemma the proof of which we give in appendix A.

Lemma 3.2.1 (Strategic equivalence of MM-GAN and NS-GAN)**.** *The MM-GAN and the NS-GAN are strategically equivalent.*

The above discussion illustrates that two GAN games can be strategically equivalent and still perform very differently in practice. In particular, when actually implementing the methods, the MM-GAN suffers from vanishing gradients while the NS-GAN does not. In our applications to risk management later on we will therefore mainly use the NS-GAN.

3.2.2 Approaches based on Distance Minimization

We have already seen that the MM-GAN game amounts to minimising the JS-divergence between the generator and data distribution. This suggests that one can also train a GAN generator by minimizing the distance between these two distributions under some other probability metric. In the following we therefore introduce one of the most commonly used probability distance measures in the GAN literature, the so-called integral probability metrics (IPM). A particularly important IPM is the Wasserstein distance which is used as the objective in the *Wasserstein GAN (WGAN)* [3]. This type of GAN has some desirable theoretical properties that the NS-GAN is lacking. We start by giving the definition of an IPM.

Definition 3.2.7 (Integral probability metric, adapted from [44, p. 1]). *For a class $\mathcal{F}$ of functions the* ***integral probability metric (IPM)*** *is defined as*

$$d_{\mathcal{F}}(\mu, \nu) = \sup_{f \in \mathcal{F}} \mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)].$$

It is easy to see that if $f \in \mathcal{F}$ implies $-f \in \mathcal{F}$, then an IPM is nonnegative, symmetric and satisfies the triangle inequality. If the class $\mathcal{F}$ is chosen large enough one can also ensure that it is positive-definite and therefore defines a proper metric on the space of probability measures.
We now give a first definition of the Wasserstein distance which does not yet lend itself to an interpretation as an IPM.

Definition 3.2.8 (Wasserstein/Earth-Mover distance, adapted from [3, p. 4]). *The* ***Wasserstein*** *or* ***Earth-Mover (EM) distance*** *is defined as*

$$W : \mathcal{P}_1(\mathbb{R}^n) \times \mathcal{P}_1(\mathbb{R}^n) \to \mathbb{R}^+, \quad (\mu, \nu) \mapsto \inf_{\gamma\in\Pi(\mu,\nu)} \mathbb{E}_{(X,Y)\sim\gamma}[\|X - Y\|] \tag{3.8}$$

where $\Pi(\mu, \nu)$ denotes the set of joint distributions $\gamma(x, y)$ with marginals μ and ν and $\mathcal{P}_1(\mathbb{R}^n)$ is the space of all probability measures μ for which $\mathbb{E}_\mu[\|X\|]$ is finite.

As its name suggests, the Earth-Mover distance can be interpreted as the amount of "mass" that has to be transported when transforming probability measure μ into probability measure ν using an optimal transport plan. It can be confirmed that the Wasserstein distance defines a proper probability metric [6, p. 106] and is therefore suitable as a GAN objective.
From definition 3.2.8 it is not yet clear that the Wasserstein distance is indeed an integral probability metric. Furthermore, the definition does not lend itself to a direct application to GANs as it requires minimisation over the set of joint probability measures which is hard to parametrise using neural networks. However, there exists a duality result, known as the Kantorovich–Rubinstein duality, that allows us to interpret the Wasserstein distance as an IPM and also makes it a tractable objective for GAN training.

Theorem 3.2.3 (Kantorovich–Rubinstein duality, adapted from [6, Theorem 5.10]). *The Wasserstein distance can equivalently be written as*

$$W(\mu, \nu) = \sup_{\|f\|_L\leq 1} \left(\mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)]\right) \tag{3.9}$$

where $\|f\|_L := \sup_{x,y\in\mathbb{R}^n} \frac{|f(x)-f(y)|}{\|x-y\|}$ denotes the Lipschitz constant of a function f. The Kantorovich–Rubinstein duality lets us interpret the Wasserstein distance as an IPM with $\mathcal{F} := \{f(\cdot) \mid \|f\|_L \leq 1\}$ being the class of 1-Lipschitz functions.

In the Wasserstein-GAN paper [3] the Kantorovich–Rubinstein duality is used to define the following objective for the Wasserstein-GAN:

$$\min_G \max_{\|D\|_L<1} W(G, D) = \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[D(X)] - \mathbb{E}_{Z\sim\mathbb{P}_Z}[D(G(Z))]. \tag{3.10}$$

The function D in this context is usually referred to as the *critic*. Equation (3.10) shows that we can understand the WGAN as a zero-sum game between the generator and the critic and so the WGAN also admits a game-theoretic interpretation. In section 3.3 we describe how to enforce the Lipschitz constraints on the critic when training the GAN in practice.
From a theoretical perspective the WGAN is more appealing than the NS-GAN. This is because convergence in Wasserstein distance is weaker than convergence in JS-divergence and therefore it might be possible to find sequences converging in Wasserstein distance and not in JS-divergence. Another desirable property is that convergence in Wasserstein distance is equivalent to convergence in distribution. The following theorem makes these statements precise.

Theorem 3.2.4 (Theoretical properties of the Wasserstein distance, adapted from [3, Theorem 2])**.** *Let $\mathbb{P}$ be a distribution on a compact space $\mathcal{X}$ and $(\mathbb{P}_n)_{n\in\mathbb{N}}$ a sequence of distributions. Then the following statements hold:*

1. *The following two statements are equivalent:*
   - *The sequence $\mathbb{P}_n$ converges to $\mathbb{P}$ in distribution: $\mathbb{P}_n \xrightarrow{\mathcal{D}} \mathbb{P}$*
   - *The sequence $\mathbb{P}_n$ converges to $\mathbb{P}$ in Wasserstein-distance: $W(\mathbb{P}_n, \mathbb{P}) \to 0$*
2. *Convergence in Jensen–Shannon divergence, i.e. $d_{\mathrm{JS}}(\mathbb{P}_n \| \mathbb{P}) \to 0$, implies both statements in (1).*
3. *If $d_{\mathrm{KL}}(\mathbb{P}_n \| \mathbb{P}) \to 0$ or $d_{\mathrm{KL}}(\mathbb{P} \| \mathbb{P}_n) \to 0$, then the sequence $\mathbb{P}_n$ also converges in Jensen–Shannon divergence towards $\mathbb{P}$.*

The above theorem states that among the notions of convergence introduced so far the KL-divergence is the strongest and the Wasserstein distance the weakest. All of them, however, imply convergence in distribution.
Apart from the Wasserstein distance several other IPM based GANs have been proposed in the literature. The most notable one probably is the Generative Moment Matching Network (GMMN) [34] for which $\mathcal{F}$ is the unit sphere in a Reproducing Kernel Hilbert Space (RKHS) and IPM minimization amounts to moment matching in a very high-dimensional feature space.

3.3 GAN Training

Despite their theoretical appeal as universal function approximators neural networks are notoriously hard to train and finding a good model often requires a brute force approach which involves trying out various configurations. This problem is even more aggravated when training GANs. In the following we will therefore first outline how GANs are trained in practice and then discuss the various problems that arise.

3.3.1 Implementation

All of the GAN objectives introduced so far can be written as two player adversarial games. The approach to numerically finding a Nash equilibrium is obvious: First, the generator and discriminator/critic are restricted to classes of neural networks which we denote by $\mathcal{G}_\Theta := \{G_\theta(\cdot) \mid \theta \in \Theta\}$ and

$\mathcal{D}_\Omega := \{D_\omega(\cdot) \mid \omega \in \Omega\}$. The objective $V(G, D)$ of the MM-GAN can then be written as a function $g(\theta, \omega)$ instead:

$$\min_{\theta\in\Theta} \max_{\omega\in\Omega} g(\theta,\omega) = \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[\log(D_\omega(X)] + \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(1 - D_\omega(G_\theta(Z)))]$$

In the game-theoretic-formulation this means $u_D(\theta,\omega) = g(\theta,\omega)$ and $u_G(\theta,\omega) = -g(\theta,\omega)$. Similarly, the payoff functions of the NS-GAN game become $u_D(\theta,\omega) = g(\theta,\omega)$ and $u_G(\theta,\omega) = \mathbb{E}_{Z\sim\mathbb{P}_Z}[\log(D(G(Z)))]$. The restriction to neural networks with bounded size introduces the first problems: In the non-restricted case we could easily show that there exists a unique Nash equilibrium. This might not be the case anymore if we restrict ourselves to neural networks. Instead of finding proper Nash equilibria one the resorts to finding **local Nash equilibria**.

Definition 3.3.1 (Local Nash-equilibrium in GAN Game, adapted from [49, Definition 1])**.** *A set of parameters (θ,ω) is in a local Nash equilibrium for the restricted GAN game if there exist open sets Θ' and Ω' with $\theta \in \Theta'$ and $\omega \in \Omega'$ such that*

$$u_G(\theta,\omega) \geq u_G(\theta',\omega) \quad \forall\theta' \in \Theta' \quad \textit{and} \quad u_D(\theta,\omega) \geq u_D(\theta,\omega') \quad \forall\omega' \in \Omega'.$$

The local Nash equilibria in the restricted games can now be found by alternating gradient ascent/descent. The expected values in the objective are approximated by their empirical counterparts based on mini-batches.

The WGAN can be trained in similar fashion. However, one has to somehow enforce the Lipschitz constraints. First, observe that we have that

$$K \cdot W(\mu,\nu) = \sup_{\|f\|_L \leq K} \left(\mathbb{E}_{x\sim\mu}[f(x)] - \mathbb{E}_{x\sim\nu}[f(x)]\right).$$

That is, scaling the Lipschitz constant also scales the Wasserstein distance by the same factor. Therefore, we can replace the class of 1-Lipschitz functions in (3.10) by any other class of Lipschitz functions with maximal Lipschitz constant K. When the critic D is approximated by a neural network, the Lipschitz constraint can be enforced by restricting the network parameters $\omega \in \mathbb{R}^m$ to a bounded set $[-C, C]^m$. Hence, the objective becomes

$$\min_{\theta\in\Theta} \max_{\omega\in[-C,C]^m} \left(\mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[D_\omega(X)] - \mathbb{E}_{Z\sim\mathbb{P}_Z}[D_\omega(G_\theta(z))]\right).$$

The generator and critic networks can then be trained as usual by backpropagating the gradient through the network and performing a gradient descent step. The difficulty comes in through the restriction of ω to $[-C, C]^d$. In the [3] the authors use a very rough approximation by clipping the weights to the set $[-C, C]^d$ (see [20] for a discussion of weight-clipping). Clearly, this is not an optimal approach and so an extension of the WGAN called the Wasserstein GAN with gradient penalty (WGAN-GP) has been proposed in [23] where the restriction on ω is lifted and instead a regularisation term is added which forces the expected value of the network gradient to be one under a measure interpolating the data and generator distribution. Algorithm 1 summarises the training procedure for the MM-GAN, NS-GAN and WGAN.

We have already seen that the Wasserstein distance might be preferable from a theoretical perspective as it is weaker than the KL- and JS-divergence for instance. The following theorem indicates that it might

Algorithm 1 (Adapted from [21] and [3]) Minibatch stochastic gradient descent based training of Generative Adversarial Networks. Hyperparameters: learning rate α, minibatch size m, number of optimisation steps k for the discriminator, clipping parameter c (for WGAN only).

```
1: for number of epochs do
2:     for $k$ steps do
3:         Sample minibatch of $m$ samples $\{z^{(1)}, ..., z^{(m)}\}$ from $\mathbb{P}_Z$.
4:         Sample minibatch of $m$ examples $\{x^{(1)}, ..., x^{(m)}\}$ from $\mathbb{P}_{\text{data}}$.
5:         Update the discriminator by gradient ascent
               NS-GAN and MM-GAN:
               $$\omega \leftarrow \omega + \alpha\ \nabla_\omega \frac{1}{m} \sum_{i=1}^{m} \left[\log\left(D_\omega(x^{(i)})\right) + \log\left(1 - D_\omega(G_\theta(z^{(i)}))\right)\right]$$
               WGAN:
               $$\omega \leftarrow \omega + \alpha\ \nabla_\omega \frac{1}{m} \sum_{i=1}^{m} \left[D_\omega(x^{(i)}) - D_\omega(G_\theta(z^{(i)}))\right]$$
6:         Clip gradients (only for WGAN):
               $$\omega \leftarrow \text{clip}(\omega, -c, c)$$
7:     end for
8:     Sample minibatch of $m$ samples $\{z^{(1)}, ..., z^{(m)}\}$ from the latent distribution $\mathbb{P}_Z$.
9:     Update the generator by gradient descent/ascent:
           NS-GAN:    $\theta \leftarrow \theta + \alpha\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} \log(D_\omega(G(z^{(i)})))$
           MM-GAN:  $\theta \leftarrow \theta - \alpha\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} \log(1 - D_\omega(G(z^{(i)})))$
           WGAN:  $\theta \leftarrow \theta + \alpha\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} D_\omega(G(z^{(i)}))$
10: end for
```

also be preferable from a numerical perspective as it tells us that the gradient exists almost everywhere. This is not the case for the KL- and JS-divergence.

Theorem 3.3.1 (Adapted from [3, Theorem 1 and Corollary 1]). *Let $\mathbb{P}_{\text{data}}$ be a fixed data distribution on $\mathbb{R}^n$ and Z a random variable on $\mathbb{R}^d$. Let $g : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}^n, \quad (z, \theta) \mapsto f_\theta(z)$ be a neural network with weights $\theta \in \mathbb{R}^m$ mapping the latent space $\mathbb{R}^d$ into the data space $\mathbb{R}^n$. Denote by $\mathbb{P}_\theta$ the distribution of $g_\theta(Z)$. Then the following statements hold:*

1. *$W(\mathbb{P}_\theta, \mathbb{P}_{\text{data}})$ is continuous in θ and differentiable almost everywhere with gradient given by*

$$\nabla_\theta W(\mathbb{P}_{\text{data}}, \mathbb{P}_\theta) = -\mathbb{E}_{Z\sim\mathbb{P}_Z}[\nabla_\theta f(g_\theta(Z))]$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the function maximizing

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{X\sim\mathbb{P}_{\text{data}}}[f(X)] - \mathbb{E}_{X\sim\mathbb{P}_\theta}[f(X)].$$

2. *The same does not hold true for the KL- and JS-divergence.*

3.3.2 Difficulties in GAN training

The training of GANs has proved to be a notoriously hard task as a variety of problems can occur. In the following we give a brief overview of the main challenges. A very readable survey of this topic is given in [36].

Mode collapse

Mode collapse, also referred to as *Helvetica scenario* in the original GAN paper [21], occurs when the generator ends up producing a very limited number of different samples which results in a lack of diversity. This especially poses a problem in the NS-GAN where the generator tries to fool the discriminator. It can easily do so by mapping the whole latent space to the single real data sample that appeared the most often in the training data set. A variety of heuristic remedies have been proposed to cope with this problem, see for example [50].

Vanishing gradient

Another major problem is that of vanishing gradients [21]. This happens when the gradient updates for the generator become very small as the current parameters are in the saturated part of the discriminator domain. As already mentioned this can often be avoided by using an NS-GAN instead of the classical MM-GAN. One of the reasons for introducing the Wasserstein-GAN particularly was to propose a method that gives clean gradients [3].

Non-convergence

Finding a Nash equilibrium in the minimax game $\min_\theta \max_\omega V(\theta, \omega)$ between the generator and discriminator amounts to finding a saddle point of the corresponding objective function $V(\theta, \omega)$. Local convergence of algorithm 1 is most often only guaranteed when $V(\theta, \omega)$ is locally convex in θ and concave in ω. In the GAN setting this is rarely the case which might result in oscillations during the training and the equilibrium might not be reached. In [41] the authors discuss conditions under which many of the popular GANs converge locally.

Hyperparameter Choice

In our analyses later on, hyperparameter tuning turned out be an incredibly difficult task. Often a slight variation in the model's hyperparameters results in a completely different fit. We therefore introduce two performance metrics in sections 4.1 and 4.2 that enable us to find a good set of hyperparameters through a grid search.

Chapter 4

Application of GANs to Risk Management

Value-at-Risk estimation is all about obtaining reliable estimates of the portfolio loss distribution function. Therefore, the quality of VaR forecasts heavily depends on the model in use. The classical methods presented in section 2.2 approximate the loss distribution function either by fitting a parametric family of distributions or using non-parametric methods such as historical simulation or kernel density estimation. GANs, on the other hand, lend themselves to yet another approach. By example they can learn to draw unseen samples of the risk-factors or portfolio losses. Hence, the set of available historical observations can be greatly enlarged by just generating new samples from the generator network. One can then apply non-parametric methods such as the historical simulation method or kernel density estimators to the enlarged data set to hopefully obtain more accurate VaR forecasts.
As with the classical methods, GANs can either learn to sample the single risk-factors or the aggregated portfolio loss. Moreover, they can produce unconditional or conditional forecasts. In the unconditional case a GAN is trained to sample from the stationary distribution of the risk-factors/portfolio loss time series. Conditional GANs incorporate the dynamics of the time series by generating new samples conditional on the observed history.
Up to now very little research has been done on VaR modelling with GANs. We are aware of only two blog-posts [25, 51] covering the topic both of which focus solely on unconditional modelling of the aggregated portfolio return. The authors also do not backtest their models and hence little can be said about the accuracy of the obtained forecasts. Our main goal in this report is to extend the results of these works in the following ways:

1. We not only do the modelling on the aggregated portfolio level but also on the more granular risk factor level.

2. We investigate conditional approaches which enable us to sample portfolio returns conditional on some window of past returns and hopefully can improve the forecast accuracy.

3. All of the models will be backtested in order to assess their performance.

4. For both the conditional and unconditional case we suggest a heuristic procedure to find a good set of model hyperparameters which is a very crucial task in GAN modelling.

We start with the seemingly easier case of unconditional modelling whereby we also present the results of the aforementioned works and then turn to conditional models.

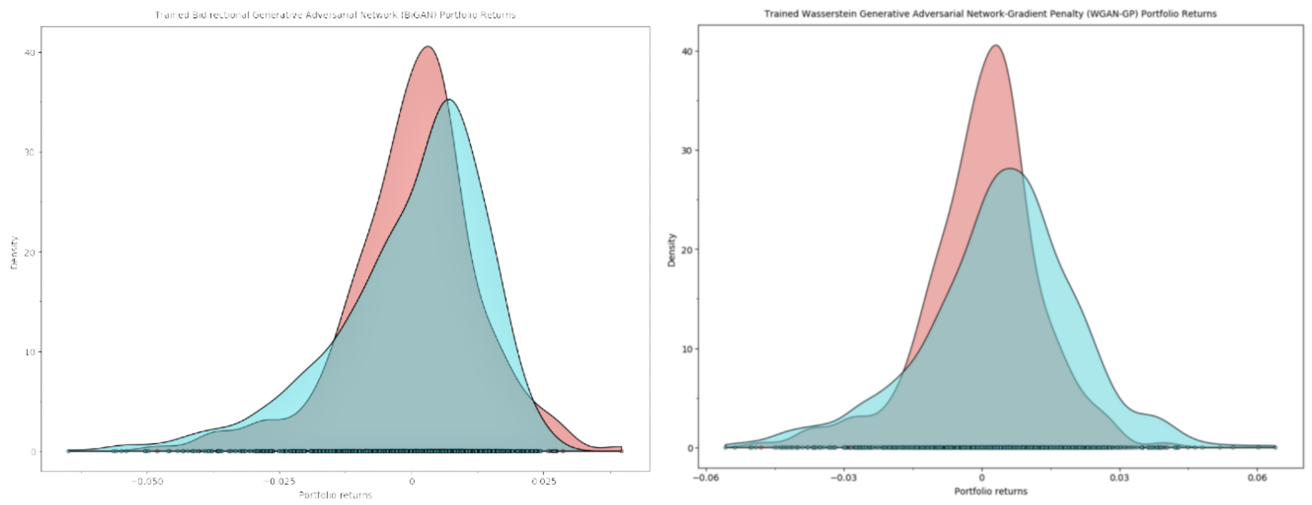

Figure 4.1: Left: Real data KDE estimate (red) vs the KDE estimate of a Bidirectional GAN (blue). Right: Real data KDE estimate (red) vs the KDE estimate of a Wasserstein GAN with gradient penalty (WGAN-GP). In both cases the portfolio is an equal weight portfolio consisting of Apple, Microsoft, Intel, Cisco and Box stocks. The images were taken from [25] and [51], respectively.

4.1 Unconditional VaR Estimation with GANs

We first review the existing results of unconditional VaR estimation with GANs and then present our approach to improve on these.

4.1.1 Previous results

Unconditional modelling of portfolio returns has been the topic of the two aforementioned blog-posts [25, 51]. In both works the authors apply the GAN methodology to sample returns of an equal-weight portfolio consisting of Apple, Microsoft, Intel, Cisco and Box stocks. The type of GAN they use for learning the distribution is different though. In [25] a *Bidirectional GAN (BiGAN)* [1] is used to learn the data distribution whereas the author in [51] employs a Wasserstein GAN with gradient penalty. Figure 4.1 shows the kernel density estimators of the real data distribution and the two GANs after training. It can be seen that the BiGAN manages to approximate the underlying data distribution better than the WGAN-GP. However, it is still far from being a good fit. The authors then use the GANs to obtain VaR forcasts by taking empirical quantiles of a sufficiently large set of GAN samples. They do, however, not backtest their models. In the following we therefore set out to first improve the fit of the GAN and then backtest its performance. Furthermore, we consider modelling the risk factors jointly.

4.1.2 Improvements of the previous results

In this section we consider a portfolio consisting of the five most liquidly traded Dow Jones stocks measured by the total trading volume in the time period from January 2010 to December 2018. These stocks are 1) Apple, 2) Microsoft, 3) Intel, 4) Cisco and 5) Pfizer. The data was obtained from Yahoo! Finance [1, 43, 28, 13, 48]. Our goal is to obtain reliable one-day VaR forecasts at the 95% and 99% confidence level for an equal-weight portfolio consisting of these five stocks. The forecasts will

[1]BiGANs [14] are an extension of the classical GAN setup in which the generator also contains an encoder which learns to map samples from the data space to their corresponding latent representation. The discriminator is then fed both the latent and real data representation to distinguish between real and fake samples. This is hoped to improve the stability of the GAN.

be unconditional in the sense of section 2.2, i.e. we only estimate the stationary distribution of the 5-dimensional time series of log-returns or the 1-dimensional time series of the aggregated portfolio loss depending on whether the modelling is done on risk-factor or portfolio level.
This section is structured as follows: In subsection 4.1.2.1 we first present the classes of candidate GAN architectures and describe the procedure we use to choose the optimal one out of those. The results of the hyperparameter tuning for our case of an equal-weight portfolio of five stocks are then discussed in subsection 4.1.2.2. Finally we backtest the readily trained model in subsection 4.1.2.3.

4.1.2.1 Hyperparameter Tuning Procedure

As already indicated in section 3.3 the training of GANs is a very complicated procedure and highly sensitive with respect to the choice of the model hyperparameters. Modifying one of them only slightly can affect the model fit tremendously. We therefore need a criterion which allows us to choose a "good" network architecture. We first present the range of admissible models and then explain how we pick the final model out of those.

Model Hyperparameters

1. **NS-GAN or WGAN:** The first question is whether to use the non-saturating GAN or the theoretically more appealing Wasserstein-GAN. In our analysis we compare both types of GANs with various architectures for the generator and discriminator networks. In the case of a WGAN we set the clipping value to $c = 0.01$ as suggested in the original WGAN paper [3].

2. **The latent variable:** Theorem 3.1.2 about the universal approximation of probability distributions showed us that in principle it does not matter which distribution we use for the latent variable as long as it is continuous. It must only be guaranteed that the dimension of the latent space is at least as large as the dimension of the manifold on which the data distribution is supported. In practice, however, we have to constrain ourselves to a bounded set of network architectures. This restriction makes the model performance dependent on the distribution and dimension of the latent variable. In fact, these two hyperparameters turn out to be highly sensitive, i.e. increasing the dimension only slightly or choosing a different distribution often gives considerably different results. In our analysis we therefore try out the following setups:

   (a) **Latent space dimension:** In principle a 5-dimensional latent variable should be sufficient to model the 5-dimensional time series of the risk factors. Similarly, to model the aggregated portfolio return a one-dimensional latent variable should suffice. However, our analysis has shown that this does not give satisfying results and so we tested various higher-dimensional configurations. More specifically, we let the latent space be 10-, 20- or 30-dimensional.

   (b) **Latent variable distribution:** We try out a multivariate standard normal distribution and a vector of independent one-dimensional t-distributed variables with four degrees of freedom as this is often observed to be the degrees of freedom in financial practice.

3. **Network architectures:** The next thing that needs to be tuned is the architecture of the generator and discriminator networks. We constrain both networks to standard fully connected feed-forward networks and consider a variety of combinations for the number of hidden layers and neurons per hidden layer in both networks. More specifically, we consider networks with 2, 3 and 4 hidden layers and 64, 128 and 256 neurons per layer.

4. **Activation functions:** As suggested in [10] we use LeakyRELU activation functions

$$a(x) = -\mathbb{1}_{(-\infty,0]}(x)\alpha x + \mathbb{1}_{(0,\infty)}(x)x \tag{4.1}$$

in both networks. The hyperparameter α is chosen to be 0.2.

5. **Optimizers:** For both the generator and discriminator we use the ADAM optimiser [29] with a very small learning rate of 0.0002 and an exponential decay rate for the first moment of 0.5. In our analyses these values have shown to give good results. A standard stochastic gradient method has also been tested but performed considerably worse.

6. **Homogeneous batches:** When training the network we only feed batches solely consisting of fake or real samples at once. This was suggested in [10].

7. **Normalization:** As is standard in machine learning applications we normalize the data to zero mean and unit variance before the training. When sampling from the generator later on we of course have to apply the inverse scaling to the samples again.

Performance metric to assess model fit

Now that we have fixed the class of admissible GAN architectures it remains to find the best configuration among that class. In order to compare different architectures, a criterion is needed that tells us how well a given model performs on data that was not part of the training set. This criterion can then be used as a performance metric in a grid search to find the best GAN configuration. For our analysis we propose the following approach:
Suppose we have a set of real data samples which we split into a training and a validation set. We then train a model with some given architecture on the training set and assess its performance on the validation set as follows: First, a kernel density estimator is fit on a sufficiently large set of samples generated by the readily trained GAN. In a second step, one evaluates the log-likelihood of the held out validation set under this fitted KDE. The higher this log-likelihood the more preferable the model is. This approach essentially amounts to choosing the model for which the KL-divergence $d_{\text{KL}}(\mathbb{P}_G||\mathbb{P}_{\text{validation}})$ between the corresponding generator distribution $\mathbb{P}_G$ and the distribution $\mathbb{P}_{\text{validation}}$ of the validation samples is minimised.
An alternative approach would be to fit the kernel density estimator on the validation set and evaluate the log-likelihood on samples from the trained GAN. This is equivalent to choosing the model for which the *reverse KL-divergence* $d_{\text{KL}}(\mathbb{P}_{\text{validation}}||\mathbb{P}_G)$ becomes minimal (note that the role of $\mathbb{P}_{\text{validation}}$ and $\mathbb{P}$ is interchanged here).
We decided to choose the former approach as the GAN allows us to draw as many samples as we wish and therefore the KDE estimate obtained from GAN samples is expected to be more reliable than a KDE estimate based on the validation set.
The best model architecture can now be found via a grid search with K-fold cross validation using the above criterion as a performance metric. In our investigations we employed a 3-fold cross validation as more than three folds make the grid search computationally too expensive. The complete procedure is shown in algorithm 2.

Algorithm 2 Hyperparameter tuning using a grid search with the proposed performance metric as a selection criterion.

```
Input: Training data $\mathcal{D} = \{x_i\}_{i=1}^n$, grid $\mathcal{S} = \left\{ \left( h_1^{(i)}, ..., h_p^{(i)} \right) : i = 1, ..., m \right\}$ of $m$ combinations for the $p$ hyperparameters to be tuned, number $K$ of folds in the cross-validation, number $N$ of samples to draw from the GAN.
Output: Best set $(h_1^*, ..., h_p^*)$ of hyperparameters, readily trained optimal model $\mathcal{M}^*$.
 1: LL$^* \leftarrow \infty$
 2: for $(h_1^{(i)}, ..., h_p^{(i)})$ in $\mathcal{S}$ do
 3:     Divide training set into $K$ disjoint folds $\mathcal{D} = \dot{\bigcup}_{k=1}^K \mathcal{D}_k$
 4:     for $k = 1, ..., K$ do
 5:         Train a GAN $\mathcal{M}(h_1^{(i)}, ..., h_p^{(i)})$ on $\mathcal{D} \setminus \mathcal{D}_k$.
 6:         Draw $N$ samples $\{\hat{x}^{(1)}, ..., \hat{x}^{(N)}\}$ from the GAN.
 7:         Fit a kernel density $p(x)$ to these samples.
 8:         Evalute the log-likelihood on the held-out set
                $$\mathrm{LL}_k = \frac{1}{|\mathcal{D}_k|} \sum_{x^{(i)} \in \mathcal{D}_k} \log(p(x^{(i)}))$$
 9:     end for
10:     Compute the average value of the performance metric over all folds
                $$\mathrm{LL} = \frac{1}{K} \sum_{k=1}^K \mathrm{LL}_k \,.$$
11:     if LL $>$ LL$^*$ then
12:         $(h_1^*, ..., h_p^*) \leftarrow (h_1^{(i)}, ..., h_p^{(i)})$
13:     end if
14: end for
15: Retrain a GAN on $(h_1^*, ..., h_p^*)$ to get the optimal model $\mathcal{M}^*$.
16: return best model $\mathcal{M}^*$
```

4.1.2.2 Hyperparameter Tuning Results

We now employ the performance criterion described in the last section to find a good GAN model to either sample from the individual log-returns of the five stocks or from the log-returns of the equal-weight portfolio directly. The grid search is performed on the stock data from the period from January 2010 to December 2013. The backtesting of the resulting model will later be performed on the time period from January 2014 to December 2018. We separate these two sets in order to avoid look-ahead-bias. Table 4.1 shows the optimal architecture for both cases.

Hyperparameter	GAN (Risk-factor level)	GAN (Portfolio level)
GAN type	NS-GAN	NS-GAN
Latent distribution	Normal	Normal
Latent space dimension	20	20
Number of layers	3	3
Number of neurons	128	128

Table 4.1: Optimal hyperparameters of a GAN to sample (a) from the 5-dimensional vector of stock log-returns or (b) from the log-return of the corresponding equal-weight portfolio.

Interestingly, the set of optimal hyperparameters is identical for both cases: Despite its theoretical appeal the WGAN performed worse than the simpler NS-GAN and so we only consider the latter in the following. The most critical hyperparameters, however, seem to be the latent distribution and dimension. If the latent variable is chosen to be a vector of univariate t-distributed random variables instead of a multivariate normal the model fit worsens considerably. Similarly, if the latent space dimension differs substantially from 20 the model fit deteriorates.

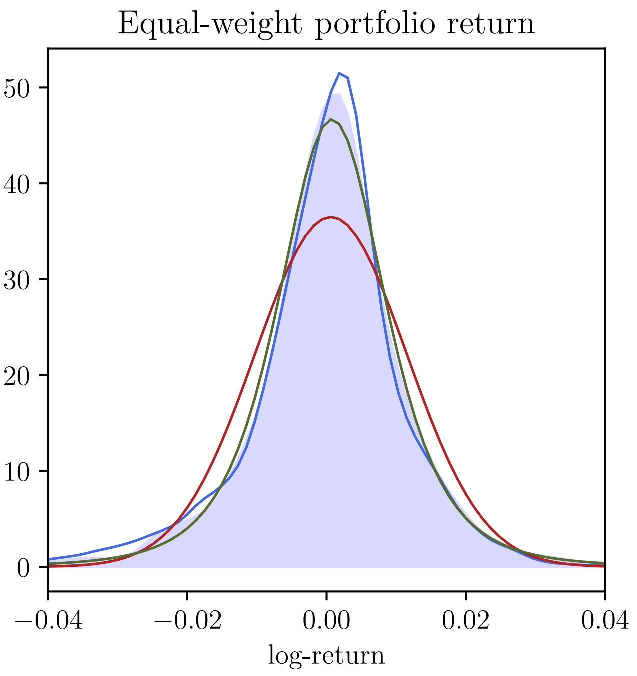

Figure 4.2: Comparison of the densities corresponding to the estimated models. The area shaded in light blue is the KDE of the original data. The blue line shows the KDE of the trained GAN. The red and green line represent the portfolio log-return densities obtained from a variance-covariance and multivariate t-distribution fit, respectively.

Figure 4.2 exhibits the model fit of the GAN methodology when the modelling is done on portfolio level. In the same plot we also show the densities of the portfolio log-returns estimated by the variance-covariance method and the multivariate t-distribution. It can be seen that the normal distribution is unable to correctly model the underlying distribution. This is expected as it is known to be a bad model for financial data due to its light tails and the lack of strong peak around the mean [12]. The t-distribution, on the other hand, exhibits a very good fit which also seems to be slightly better than the GAN approximation. However, compared to the GAN fits obtained in the blog-posts [25, 51] which were shown in figure 4.1 our GAN performs substantially better. The behaviour in the left tails seems to be especially desirable as it captures the small hump in the data KDE better than the normal and t-distribution. We will soon see that this indeed differentiates the GAN model from its classical competitors as it gives more realiable VaR forecasts over the long run.
In figure 4.3 we show the resulting fits when the modelling is instead done on the risk-factor level. The variance-covariance method again seems to be inadequate whereas the t-distribution and the GAN KDE approximate the data distribution fairly well. Especially on the portfolio level the GAN fit seems to be very good.

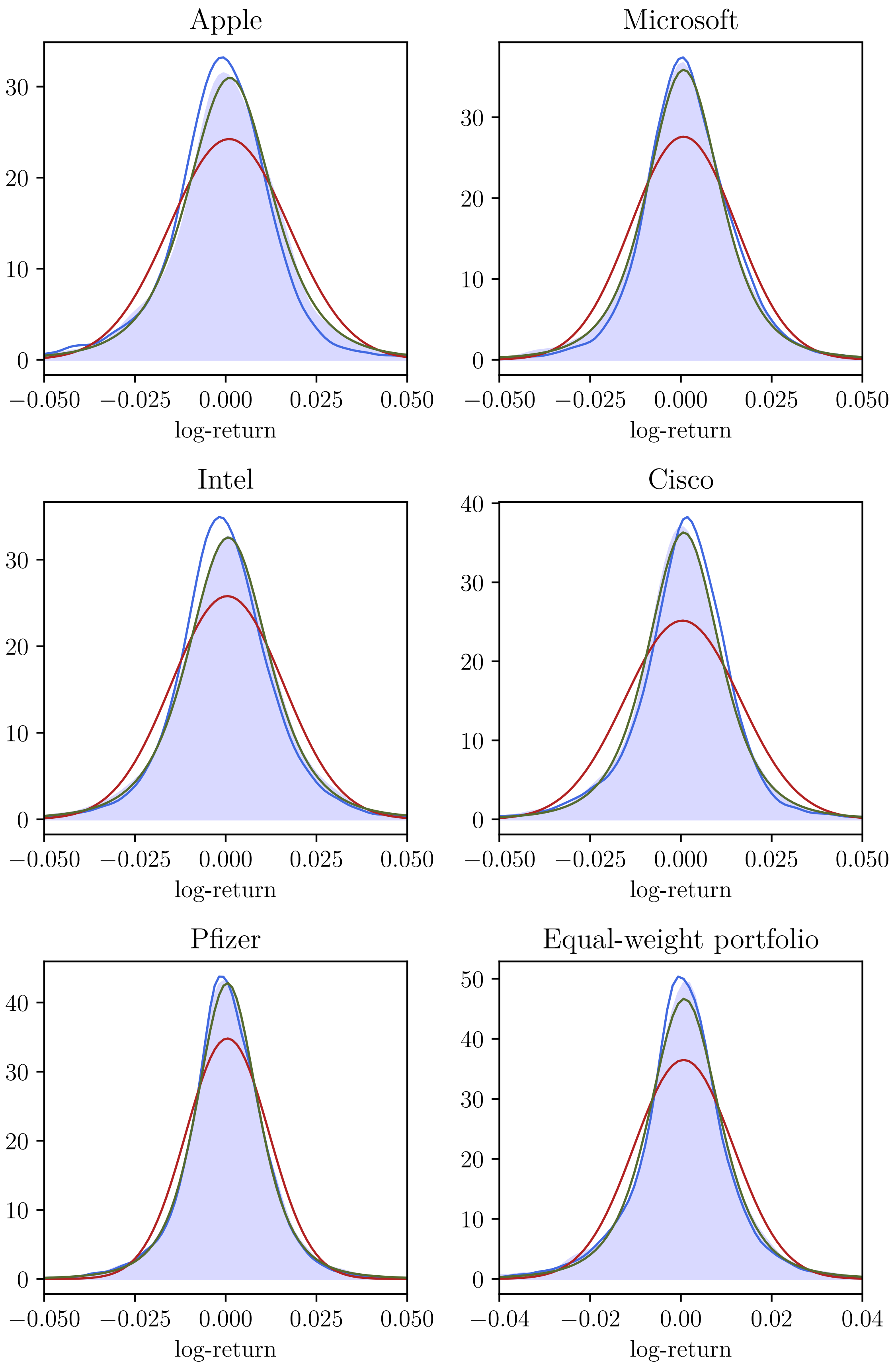

Figure 4.3: Comparison of the densities corresponding to the estimated models. The area shaded in light blue is the KDE of the original data. The blue line is the KDE of the trained GAN. The red and green line represent the marginal densities of a fitted multivariate normal and t-distribution, respectively.

4.1.2.3 Backtesting the unconditional model

Now that we have chosen the model architectures and already gotten a taste of the potential of GANs we backtest both new models in order to check if GANs can indeed give superior VaR forecasts. The backtesting performance is compared with that of the classical unconditional methods, namely the variance-covariance method, the multivariate *t*-distribution, the historical simulation method and a kernel-density based approach whereby we proceed as follows:

- The backtesting is done for the five year period from January 2014 to December 2018. Each day, a VaR forecast for the next day is computed based on a time window of the past 1000 trading days which amounts to roughly four years (The choice of 1000 days for the time window was adapted from [39, p. 57]).
- For both GAN models we use the log-returns from January 2010 to December 2013 to find the optimal network architecture as described in the previous section. This results in the hyperparameters presented in table 4.1 and avoids look-ahead-bias.
- When the rolling VaR forecasts are computed, the GAN models are only updated each tenth trading day for computational efficiency reasons. We also do not fully retrain the GANs every time on the new time window of the past 1000 trading days. Instead another 100 training epochs are performed on the current model using the data from the new time window.
- Finally, the VaR forecasts of a GAN model are computed as follows: We sample 20000 log-returns from the generator, fit a KDE to these samples and then take the corresponding quantile of the KDE (we could have also just chosen the empirical quantile as the estimate but the KDE approach performed slightly better).

The Basel II accord requires banks to report the 10-day VaR at the 99% confidence level every day. The backtesting, however, has to be done using 1-day VaR forecasts at the 95% level [5]. The reason for this is obvious: Reducing the time horizon of the VaR forecasts increases the number of available observations and reducing the confidence level makes the hypothesis tests in backtesting more reliable as exceptions are rare events and one at least needs a few of them to obtain meaningful test statistics. Therefore, in the following we will focus on 1-day VaR forecasts at the 95% confidence level. The results for the 99%-VaR can be found in appendix D.

Year	2014	2015	2016	2017	2018	Overall
Trading days	252	252	252	251	251	1258
Expected no. of exceptions	12.6	12.6	12.6	12.6	12.6	62.9
VC	4 (0.00)	19 (0.08)	14 (0.69)	4 (0.00)	28 (0.00)	69 (0.44)
VC-t	4 (0.00)	19 (0.08)	14 (0.69)	4 (0.00)	30 (0.00)	71 (0.30)
HS	4 (0.00)	21 (0.03)	14 (0.69)	4 (0.00)	28 (0.00)	71 (0.30)
KDE	3 (0.00)	19 (0.08)	14 (0.69)	4 (0.00)	28 (0.00)	68 (0.51)
GAN-RF	4 (0.00)	21 (0.03)	14 (0.69)	4 (0.00)	31 (0.00)	74 (0.16)
GAN-P	3 (0.00)	17 (0.23)	14 (0.69)	4 (0.00)	27 (0.00)	65 (0.79)

Table 4.2: Number of exceptions of the 95% Value-at-Risk calculated using the classical as well as the GAN methods. The value in brackets is the p-value of Kupiec's POF test. The p-values of the IF and TBF test are below 0.05 in all cases. The abbreviations of the methods are as follows: Variance-Covariance (VC), t-distribution (VC-t), Historical Simulation (HS), Kernel Density Estimator (KDE), GAN on Risk-Factor level (GAN-RF), GAN on Portfolio level (GAN-P).

Table 4.2 shows for each year in the backtesting period the expected number of exceptions as well as the number of actually observed ones for the various models under consideration. The value in brackets is the p-value of Kupiec's POF test which tests for coverage. We can see that backtesting performance varies substantially from year to year. All models overestimated the Value-at-Risk in 2014 and 2017 and underestimated it in 2015 and 2018. Only in 2016 the number of exceptions is close to its expected value leading to a fairly high p-value of 0.69 for all models. This quite volatile behaviour in the forecast accuracy, however, is expected as unconditional models only aim to correctly estimate the stationary distribution. Over short time periods deviations are therefore natural but over the long run the fraction of exceptions should be close to its expected value. This can indeed be observed in the last column of table 4.2: Over the whole period from 2014 to 2018 the numbers of exceptions are much closer to their expected value than in a single year. This assertion is further strengthened by the high p-values of the POF test. The GAN which models the portfolio returns seems to be particularly appealing as it gives the highest p-value of all models. Its backtesting performance is visualised in figure 4.4 where we can clearly see that it is not capable of capturing volatility clustering. The other GAN which models the risk-factors individually, however, performs worse than the classical methods which possibly is due to the greater modelling difficulty in higher dimensions. Nevertheless, we showed that GANs can indeed produce reliable unconditional forecasts and constitute a sensible alternative to the classical methods. The p-values for the Interval Forecast (IF) and Time Between Failure (TBF) tests have also been computed for each year and each method. In all cases they lie below 0.05 and therefore indicate that the unconditional methods cannot incorporate volatility clustering which is what we expected.

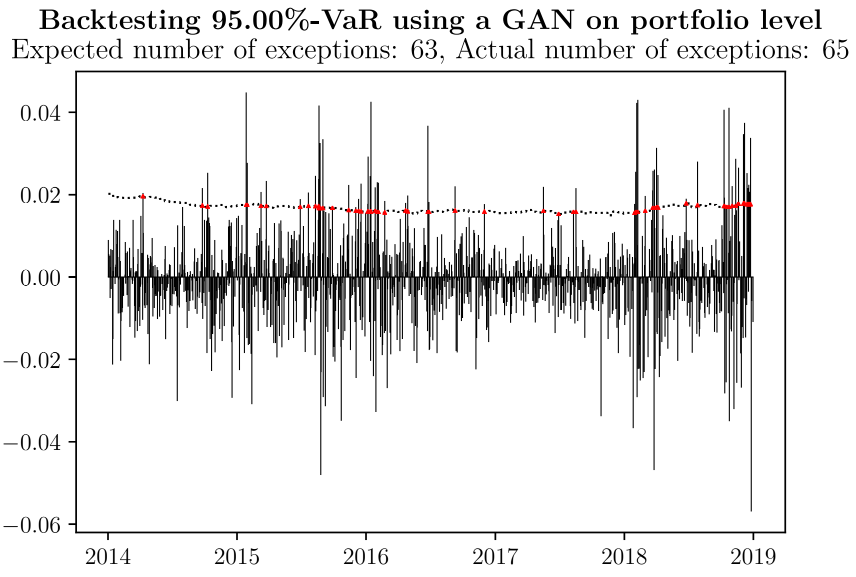

Figure 4.4: Backtesting the 95%-VaR forecast performance of the GAN model that samples from the equal-weight portfolio log-returns. The vertical lines indicate the negative daily log-returns and the dotted line gives the corresponding VaR forecast. Exceptions are indicated by red triangles.

4.2 Conditional VaR

The goal of this section is to extend the results from the previous one to the conditional case. That is, the GAN should learn to sample the next log-return given some time window of past log-returns. By doing so we hope to capture volatility clustering effects and therefore give more accurate VaR forecasts. We can then again compare the backtesting performance of the GAN methodology with that of the conditional classical approaches such as DCC-GARCH.

Remember that a standard GAN learns to generate realisations of a random variable X. A minor extension of it, the *Conditional GAN (CGAN)* [42], instead learns to generate samples from the conditional random vector $X|Y = y$ where Y denotes another random variable. It achieves this by also feeding both the generator and the discriminator the realization y of Y. In the CGAN paper [42], which is mainly concerned with image generation, X represents an image and Y is the label of the object shown in the image. By training a CGAN on labeled images one can then learn to generate new images showing certain objects. In our application we have to condition on the realisation of some continuous time series as opposed to a discrete label. This makes the training considerably harder. More formally, let $(x_{t-d}, ..., x_{t-1})$ be the realisation of the time series of interest in some time interval $[t - d, t - 1]$ with d being the window size. The goal is then to generate a sample of X_t given $(x_{t-d}, ..., x_{t-1})$. Since in the last section we found that modelling on the portfolio level gave better results than modelling on the risk-factor level, we only consider the former case in this section. Hence, all time series in the following are assumed to be one-dimensional.

Th section is organised as follows: In subsection 4.2.1 we discuss the model architecture we will be using to incorporate the dynamic behaviour of the time series. We also present a heuristic performance metric for choosing optimal hyperparameters for this new architecture. Results of the hyperparameter tuning and the corresponding model fit are discussed in subsection 4.2.1.1. Finally, in subsection 4.2.1.2 the backtesting performance of the model will be compared with that of the classical conditional models such as DCC-GARCH.

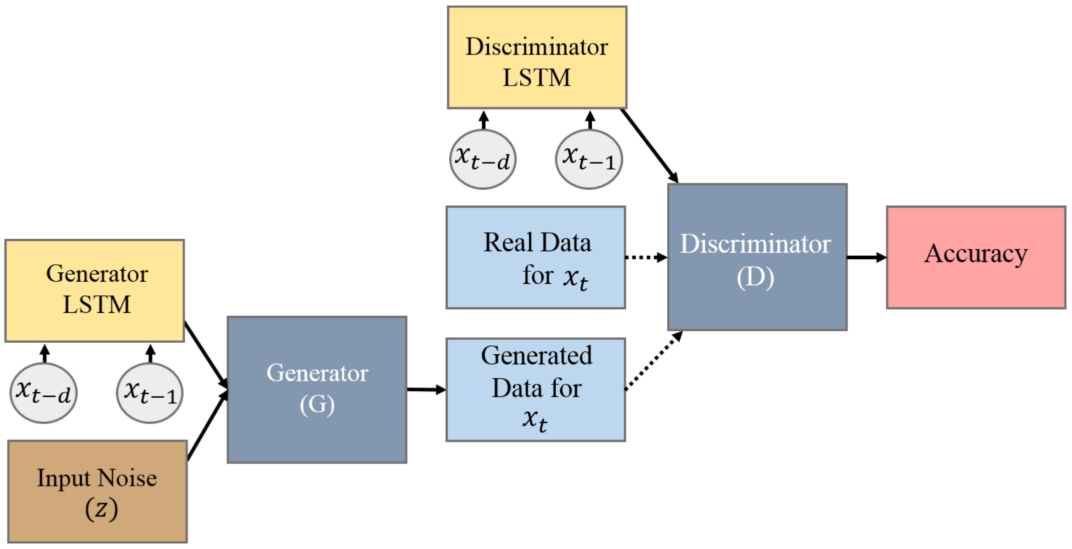

Figure 4.5: Architecture of the RCGAN for conditional log-return sampling. The dotted lines indicate that only one of the two is fed to the discriminator.

4.2.1 Model Architecture and Hyperparameter Tuning

The architecture we choose for the GAN is shown in figure 4.5. In order to take the time series character of the problem into account we first feed the history $(x_{t-d}, ..., x_{t-1})$ through an LSTM layer (Hochreiter et al. [26]). Note that we use a separate LSTM for the generator and the discriminator. The output of the generator LSTM is then concatenated with the noise vector and fed to the generator. Similarly, the output of the discriminator LSTM is concatenated with a real or fake sample and fed to the discriminator. In the following we refer to this GAN as *Real-valued conditional GAN (RCGAN)* as it is very similar to the original RCGAN proposed by Esteban et al. [18].
Just like for the unconditional GAN it is crucial to determine a good set of model hyperparameters. This can again be achieved with a grid search. However, we now need another performance metric as evaluation of a conditional likelihood is infeasible. We therefore proceed as follows: Assume again that we have fitted a RCGAN on the training set and now want to evaluate it on the validation set. If the model were close to optimal, the L^2-distance between the conditional mean predicted by the RCGAN and the actual observed log-return the next day should be small. We therefore prefer a model for which the mean squared error between the sample mean of the conditional samples and the observed log-returns the next day becomes minimal. More formally, let $(x_1, ..., x_T)$ be the sequence of all log-returns in the validation set and d the window size used for conditioning. We further write $\hat{x}_t^{(1)}, ..., \hat{x}_t^{(n)}$ for a set of n RCGAN samples for X_t conditional on $(x_{t-d}, ..., x_{t-1})$, $t \in \{d+1, ..., T\}$. The performance metric of the GAN is then given by the following MSE:

$$\text{MSE} = \frac{1}{T} \sum_{t=d+1}^{T} \left(\frac{1}{n} \sum_{i=1}^{n} \hat{x}_t^{(i)} - x_t \right)^2$$

This metric can then again be used in a grid search to find a good set of hyperparameters as detailed in algorithm 3.

Algorithm 3 Hyperparameter tuning using a grid search with the proposed performance metric as a selection criterion.

Input: Grid $\mathcal{S} = \left\{ \left(h_1^{(i)}, ..., h_p^{(i)} \right) : i = 1, ..., m \right\}$ of m combinations for the p hyperparameters to be tuned, training data $\mathcal{D} = \{((x_{t-d}, ...x_{t-1}), x_t)\}_{t=d+1}^{T}$, number K of folds in the cross-validation, number N of samples to draw from the GAN.

Output: Best set $(h_1^*, ..., h_p^*)$ of hyperparameters, readily trained optimal model $\mathcal{M}^*$.

```
1: MSE* ← ∞
2: for $(h_1^{(i)}, ..., h_p^{(i)})$ in $\mathcal{S}$ do
3:     Divide training set into $K$ disjoint folds $\mathcal{D} = \dot{\bigcup}_{k=1}^{K} \mathcal{D}_k$
4:     for $k = 1, ..., K$ do
5:         Train a GAN $\mathcal{M}(h_1^{(i)}, ..., h_p^{(i)})$ on $\mathcal{D} \setminus \mathcal{D}_k$.
6:         for $((x_{t-d}, ...x_{t-1}), x_t) \in \mathcal{D}_k$ do
7:             Given $(x_{t-d}, ...x_{t-1})$ draw $N$ samples $\hat{x}_t^{(1)}, ..., \hat{x}_t^{(N)}$ for $X_t$ from the GAN
8:         end for
9:         Compute the MSE via
               $$\mathrm{MSE}_k = \frac{1}{|\mathcal{D}_k|} \sum_{x_t \in \mathcal{D}_k} \left( \frac{1}{n} \sum_{i=1}^{n} \widehat{x}_t^{(i)} - x_t \right)^2 .$$
10:    end for
11:    Compute the average value of the performance metric over all folds
               $$\mathrm{MSE} = \frac{1}{K} \sum_{k=1}^{K} \mathrm{MSE}_k .$$
12:    if MSE < MSE* then
13:        $(h_1^*, ..., h_p^*) \leftarrow (h_1^{(i)}, ..., h_p^{(i)})$
14:    end if
15: end for
16: Retrain a GAN on $(h_1^*, ..., h_p^*)$ to get the optimal model $\mathcal{M}^*$.
17: return best model $\mathcal{M}^*$
```

4.2.1.1 Hyperparameter Tuning Results

The RCGAN has three additional hyperparameters - the window size and the output dimension of the generator and discriminator LSTM. We only optimize over these new hyperparameters and choose the remaining ones to be the same as in the unconditional GAN (see table 4.1). We let the output dimension of the generator LSTM be 40, 70 or 100 and that of the discriminator to be 10, 20 or 40. For the window size we consider the values 10, 20 and 30. In our preliminary analysis values different from these gave considerably different models and so we restrict the search to these values. The resulting hyperparameters are shown in table 4.3.

We now illustrate the model fit via visualisation. In the next section when we backtest the model this will be made more mathematically rigorous. Figure 4.6 shows the evolution of the portfolio price over the time period from February to August 2018. It also shows three trajectories sampled from the RCGAN starting at some date in the middle of the time period. This is achieved by rolling forecasts

Hyperparameter	RCGAN (Portfolio level)
Window size	10
Generator LSTM output dimension	40
Discriminator LSTM output dimension	40

Table 4.3: Optimal hyperparameters for the RCGAN.

and using the previous prediction as an input for the conditioning when the next price is sampled. We also show the distribution over the generated paths as well as the mean prediction. It can be observed that the true price evolution lies close to the predicted mean. Also, the generated trajectories look like realistic outcomes.

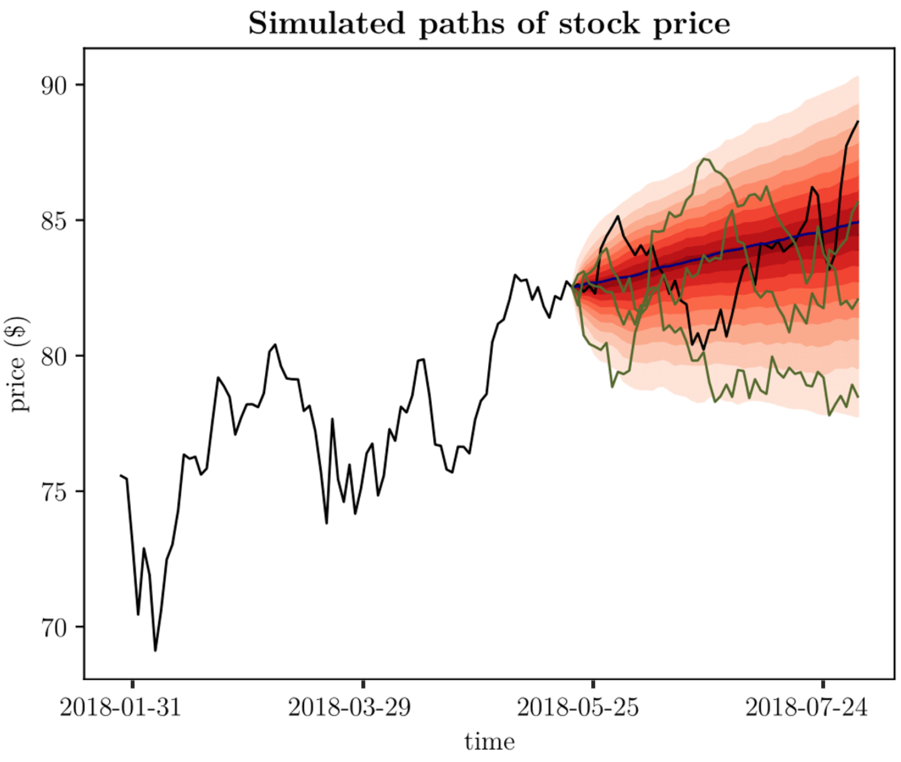

Figure 4.6: Simulation of generic paths of the trained GAN vs real real portfolio price evolution. The black line shows the evolution of the portfolio value, the green lines show three random paths generated by the RCGAN. The predicted mean is shown by the blue line and the red area is a fan plot of the corresponding distribution over the RCGANs paths.

Another thing one might want to check are the (partial) autocorrelations of the generated and real time series. Figure 4.7 shows the autocorrelation function (ACF) of the generated and true times series. Conforming with the stylized facts of financial time series [12] the plots do not exhibit much first order dependence. However, the ACFs of the corresponding squared times series as shown in 4.8 exhibit substantial autocorrelation which gives evidence for volatility clustering. The ACFs look very similar indicating that the RCGAN has indeed learned the dynamic behaviour of the underlying time series. The corresponding plots for the partial autocorrelation function (PACF) can be found in appendix D.1 and further strengthen the assertion that the RCGAN captures the true dynamics.

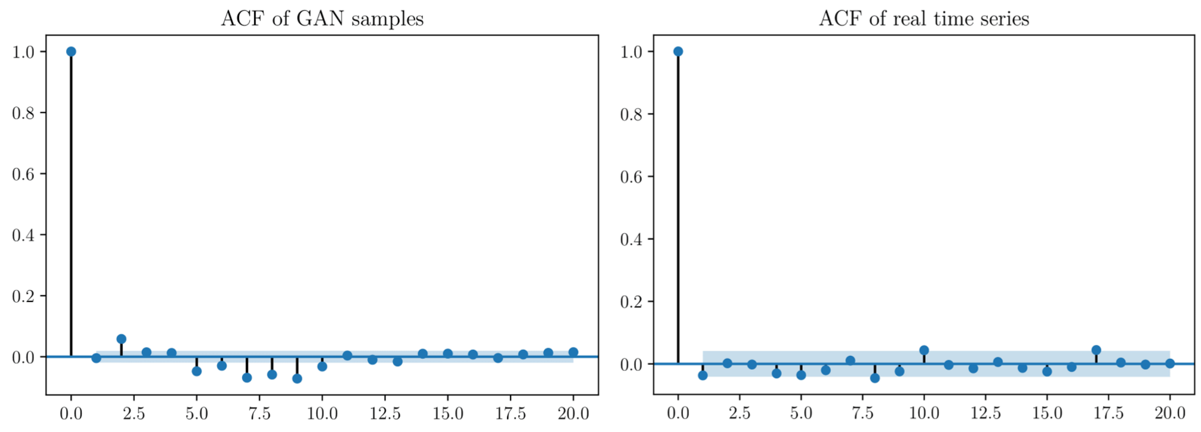

Figure 4.7: ACF of the time series generated by the RCGAN (left) and of the real time series (right).

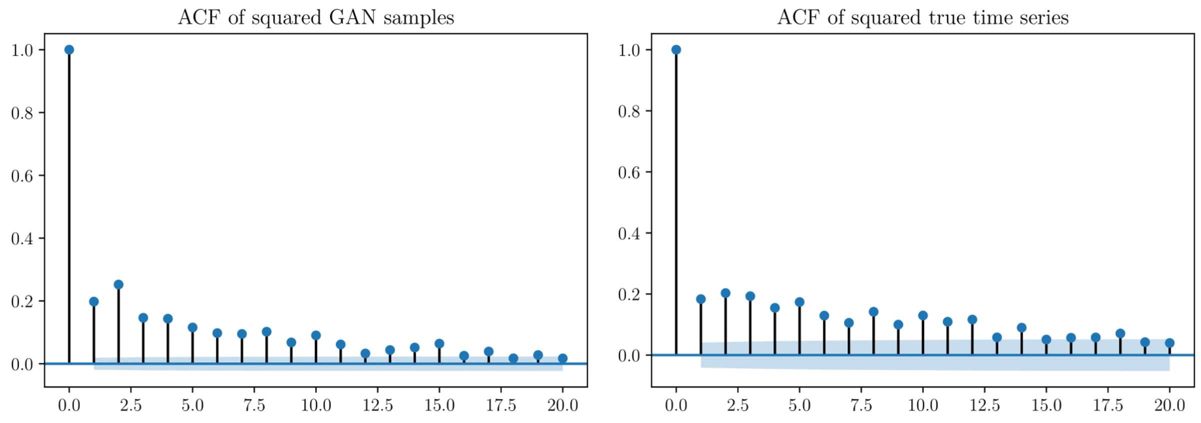

Figure 4.8: ACF of the squared time series generated by the RCGAN (left) and of the squared real time series (right).

As a third sanity check we compare the stationary distribution of the portfolio returns with the stationary distribution of the RCGAN time series. To obtain unconditional samples from the RCGAN we randomly sample time windows of the true data, condition the RCGAN on it and then draw a sample of the next return. This way we essentially integrate out the conditioning variable. The two corresponding KDEs are compared in figure 4.9 and show a very good fit of the RCGAN.

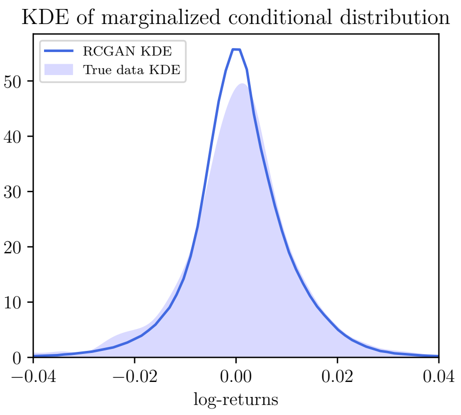

Figure 4.9: Kernel density estimate of the true return distribution and kernel density estimate of the marginalzed GAN samples.

4.2.1.2 Backtesting the conditional model

In this final section we compare the backtesting performance of the RCGARCH with that of classical time series approaches. In particular we consider the following time series models:

1. A univariate GARCH(1,1) model with standard normal innovations modelling the portfolio returns.
2. A univariate GARCH(1,1) model with t-distributed innovations modelling the portfolio returns.
3. A multivariate DCC(1,1)-GARCH(1,1) model with multivariate standard normal innovations modelling the single risk-factors.
4. A multivariate DCC(1,1)-GARCH(1,1) model with multivariate t-distributed innovations modelling the single risk-factors.

The backtesting procedure is exactly as described in section 4.1.2.3, that is the model hyperparameters are tuned on the time period from 2010 to 2013 and the backtesting is done from 2014 to 2018 using rolling time windows of size 1000. Tables 4.4 and 4.5 show the backtesting results. It can be observed that the models suffer much less from underestimated volatility clustering than the unconditional ones. In all five years the number of exceptions of the classical methods lies within a reasonable range around the expected value as confirmed by the corresponding p-values. Overall, the univariate GARCH(1,1)-model with t-distributed innovations seems to outperform all other models. The RCGAN, however, performs the worse out of all the models. This is due to lacking robustness of the model as updating the RCGAN on a new time window often changes the forecasts considerably. This instability can also be seen in figure 4.10. Overall the model can capture the volatility clustering fairly well. However, it is too sensitive to small changes. For comparison we also show the backtesting performance of the univariate GARCH(1,1) model with t-distributed innovations in appendix D.2 which reacts much slower to changes in the underlying process.

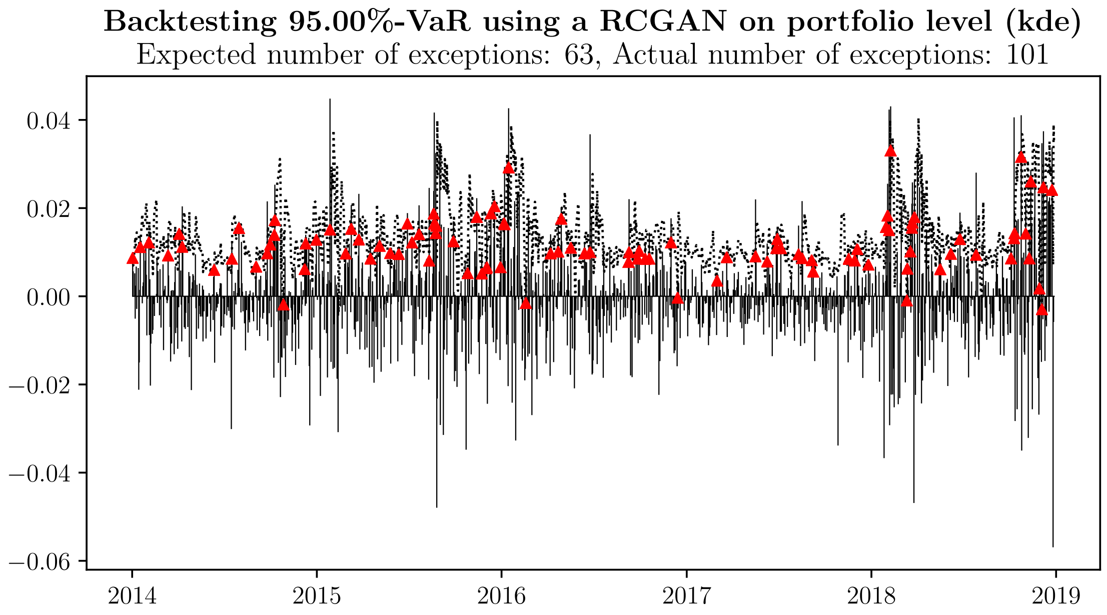

Figure 4.10: 95% VaR backtesting performance of the RCGAN model.

Year	2014	2015	2016
Trading days	252	252	252
Expected no. of exceptions	12.6	12.6	12.6
GARCH(1,1)-norm	10 (0.44; 0.36; 0.49; 0.08)	15 (0.50; 0.28; 0.44; 0.08)	12 (0.86; 0.59; 0.85; 0.06)
GARCH(1,1)-t	10 (0.44; 0.36; 0.49; 0.08)	15 (0.50; 0.28; 0.44; 0.08)	13 (0.91; 0.69; 0.92; 0.06)
DCC(1,1)-GARCH(1,1)-norm	10 (0.44; 0.40; 0.52; 0.03)	17 (0.23; 0.40; 0.33; 0.05)	15 (0.50; 0.28; 0.44; 0.01)
DCC(1,1)-GARCH(1,1)-t	13 (0.91; 0.13; 0.32; 0.01)	18 (0.14; 0.48; 0.26; 0.10)	15 (0.50; 0.28; 0.44; 0.01)
RCGAN-P	18 (0.14; 0.88; 0.34; 0.20)	25 (0.00; 0.32; 0.00; 0.16)	18 (0.14; 0.53; 0.28; 0.12)

Table 4.4: First part of the backtesting results. For each method the expected as well as the actual number of exceptions is stated. The values in the brackets are the *p*-values of the *p*-values of the POF, IF, CC and TBF test, respectively.

Year	2017	2018	Overall
Trading days	251	251	1258
Expected no. of exceptions	12.6	12.6	62.9
GARCH(1,1)-norm	7 (0.08; 0.53; 0.18; 0.44)	17 (0.21; 0.44; 0.34; 0.27)	51 (0.81; 0.25; 0.51; 0.00)
GARCH(1,1)-t	10 (0.44; 0.36; 0.49; 0.17)	16 (0.33; 0.36; 0.41; 0.03)	54 (0.88; 0.34; 1.00; 0.00)
DCC(1,1)-GARCH(1,1)-norm	6 (0.04; 0.59; 0.09; 0.14)	20 (0.04; 0.74; 0.13; 0.02)	68 (0.51; 0.10; 0.00; 0.00)
DCC(1,1)-GARCH(1,1)-t	7 (0.08; 0.52; 0.18; 0.13)	24 (0.00; 0.82; 0.01; 0.01)	77 (0.08; 0.14; 0.00; 0.00)
RCGAN-P	15 (0.49; 0.17; 0.30; 0.41)	25 (0.00; 0.82; 0.01; 0.01)	101 (0.00; 0.15; 0.00; 0.00)

Table 4.5: Part 2 of table 4.4.

Chapter 5

Conclusion

The aim of the present thesis was to apply Generative Adversarial Networks to the problem of Value-at-Risk estimation. We first introduced the essentials of classical market risk management and then discussed in quite some depth the theoretical properties of GANs. In particular we showed that they can be understood as universal approximators of probability distributions. This feature makes them incredibly attractive for estimation tasks where the underlying distribution is very complex and might not be well described by any classical family of probability distributions. It therefore seems reasonable to apply GANs to the problem of financial time series modelling which is widely known to be a very difficult exercise. In particular, our goal was to model the whole (un)conditional distribution of the time series instead of only forecasting the conditional mean.
We began our study with the problem of modelling the stationary distribution of the log-returns of an equity portfolio. It turned out to be incredibly difficult to find a good set of model hyperparameters such that the corresponding GAN could accurately capture the tail-behaviour. After a brute-force grid search using our proposed performance metric we were able to find a set of hyperparameters for which the corresponding GAN was able to model the tail behaviour sufficiently well to outperform classical unconditional methods in the backtesting. This shows both the strengths and weaknesses of this novel approach. A major deficiency is the lackig robustness when it comes to the choice of the hyperparameters. Twisting some nuumbers a bit might render the VaR forecast extremely inaccurate. It is therefore of utmost importance that practioners seeking to employ this method are aware of the risks associated with it. If used diligently, however, GANs might be a useful tool for unconditional VaR forecasts due to their desirable approxmation properties.
The case of conditional VaR forcasts turned out to be even more involved. The right choice of good hyperparameters is considerably difficult and so the model was not able to compete with the classical models in the backtesting as it reacts to quickly to market changes. However, it could still be seen that the RCGAN captured the volatility clustering fairly well. Hence, it is expected that by making the calibration more stable one can obtain more accurate forecast. We leave this task for future research.

Appendix A

Proofs from Chapter 3

In this section we give the proofs of some of the theorems from chapter 3. We start with the universal approximation theorem for probability distributions which we restate for the sake of convenience.

Theorem A.0.1 (Universal approximation of probability distribution)**.** *Let Z be a continuous random variable taking values in $(\mathbb{R}^d, \mathcal{B}(\mathbb{R}^d))$ and X a continuous random variable on a d-dimensional differentiable manifold in $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n))$ where $n \geq d$. Then there exists a sequence of neural networks $G_{\theta_n} : \mathbb{R}^d \to \mathbb{R}^n$ with weights θ_n such that $G_{\theta_n}(Z)$ converges in distribution to X.*

Proof. The first step is to construct a function continuous $G : \mathbb{R}^d \to \mathbb{R}^n$ for which $G(Z) \overset{\mathcal{D}}{=} X$. If both Z and X are continuous univariate random variables then one can choose $G(z) = F_X^{-1}(F_Z(z))$ with the distribution functions $F_X(x)$ and $F_Z(z)$ of X and Z, respectively. The idea behind the choice of G is to first transform Z into a $\mathcal{U}([0,1])$-distributed random variable and then apply the inverse sampling method to generate a sample of X. This only works when Z is continuous. We now extend this idea to the multidimensional case where we have to take the dependency structure into account. To do so we first write

$$F_Z(z_1, ..., z_d) = C_Z(F_{Z_1}(z_1), ..., F_{Z_d}(z_1)) \tag{A.1}$$

where $C_Z(u_1, ..., u_d)$ is a copula and $F_{Z_k}(z_k)$, $k = 1, ..., d$, are the distribution functions of the marginals of Z. The existence of this representation is guaranteed by Sklar's theorem [46, Theorem 2.3.3]. To construct the function G we make use of a sampling method for copulas known as *conditional sampling* [9, p. 182-188] and invert it, i.e. instead of starting with independent uniforms to generate samples from the desired copula we start with a sample and reconstruct independent uniforms on $[0, 1]$.
In order to sample from a copula $C(u_1, ..., u_n)$ one first defines the functions

$$C_k(u_1, ..., u_k) = C(u_1, ..., u_k, 1, ..., 1), \quad k = 2, ..., n$$

which are just the copulas for the first k components. Then the distribution function of the k-th component conditioned on the first $k-1$ components can be computed via

$$\begin{aligned} C_k(u_k | u_1, ..., u_{k-1}) &= \mathbb{P}(U_k \leq u_k | U_1 = u_1, ..., U_{k-1} = u_{k-1}) \\ &= \frac{\left[\partial^{k-1} C_k(u_1, ..., u_k)\right] / \left[\partial u_1 ... \partial u_{k-1}\right]}{\left[\partial^{k-1} C_{k-1}(u_1, ..., u_{k-1})\right] / \left[\partial u_1 ... \partial u_{k-1}\right]}. \end{aligned}$$

One can then sample from the copula by applying the following steps:

1. Sample n independent $\mathcal{U}([0,1])$-distributed random variables $V_1, ..., V_n$.
2. Set $U_1 = V_1$ and $U_k = C_k^{-1}(V_k|U_1, ..., U_{k-1})$ where

$$C_k^{-1}(v|u_1, ..., u_{k-1}) = \inf\{t \mid C_k(t|u_1, ..., u_{k-1}) \geq v\}$$

   denotes the generalised inverse of $C_k(u_k|u_1, ..., u_{k-1})$.

In order to sample from Z with cdf as given in equation (A.1) we add the following third step:

3. Set $Z_k = F_{Z_k}^{-1}(U_k)$ for $k = 1, ..., d$.

We can now transform Z into a d-dimensional vector of independent uniform random variables on $[0,1]$ by inverting the above steps, i.e. we apply to Z the function

$$F : \mathbb{R}^d \to \mathbb{R}^d, \quad Z \mapsto (F_d \circ F_{d-1} \circ ... \circ F_1)(Z)$$

where

$$F_1 : \mathbb{R}^d \to \mathbb{R}^d, \quad Z \mapsto (F_{Z_1}(Z_1), Z_2, ..., Z_d)^T$$

and

$$F_k : \mathbb{R}^d \to \mathbb{R}^d, \quad Z \mapsto (Z_1, ..., Z_{k-1}, C_k(F_{Z_k}(Z_k)|Z_1, ..., Z_{k-1}), Z_{k+1}, ..., Z_d)^T, \quad k = 2, ..., d.$$

As all component functions are continuous F itself is continuous. It remains to transform the vector of d independent uniforms into a sample of X. As X is a continuous random variable on the manifold M (for simplicity represented by a single chart T) we have

$$\begin{aligned}\mathbb{P}(X \in A) &= \int_{M \cap A} f_X(x) \mathrm{d}S(x) \\ &= \int_{T(\Omega) \cap A} f_X(x) \mathrm{d}S(x) \\ &= \int_{\Omega} f_X(T(x)) \mathbb{1}_{\{T(x) \in A\}} \sqrt{\det(DT^T(x) DT(x))} \mathrm{d}x.\end{aligned}$$

This shows us that we can sample from X by first sampling from a continuous d-dimensional random variable Y with density

$$f_Y(y) = f_X(T(x)) \sqrt{\det(DT^T(x) DT(x))} \mathbb{1}_{\{x \in \Omega\}}$$

and then setting $X = T(Y)$. The sampling from Y can be done by again writing its distribution function $F_Y(y_1, ..., y_d)$ as

$$F_Y(y_1, ..., y_d) = C_Y(F_{Y_1}(y_1), ..., F_{Y_d}(y_d)).$$

An application of the conditional sampling method (this time in the usual direction) gives that for a d-dimensional vector $U = (U_1, ..., U_d)^T$ of independent uniforms on $[0,1]$ we get $Y \overset{\mathcal{D}}{=} H(U)$ where

$$H : \mathbb{R}^d \to \mathbb{R}^d, \quad U \mapsto (H_d \circ H_{d-1} \circ ... \circ H_1)(U)$$

with

$$H_1 : \mathbb{R}^d \to \mathbb{R}^d, \quad U \mapsto (F_{Y_1}^{-1}(U_1), U_2, ..., U_d)^T$$

and

$$H_k : \mathbb{R}^d \to \mathbb{R}^d, \quad U \mapsto (U_1, ..., U_{k-1}, F_{Y_k}^{-1}(C_k^{-1}(U_k|F_{Y_1}(U_1), ..., F_{Y_{k-1}}(U_{k-1}))), U_{k+1}, ..., U_d)^T$$

for $k = 2, ..., d$. Therefore, combining all the previous results we obtain that

$$X \overset{\mathcal{D}}{=} G(Z) \quad \text{with} \quad G : \mathbb{R}^d \to M, \quad z \mapsto (T \circ H \circ F)(z).$$

Note that the constructed function G is continuous as all of the functions in the composition are continuous by our assumptions on the random variables X and Z.
Now that we have constructed G for which $G(Z) \overset{\mathcal{D}}{=} X$, the next step is to apply the universal approximation theorem to approximate G by neural networks on compacta. First choose an increasing sequence $(R_n)_{n\in\mathbb{N}}$ such that $\mathbb{P}(|Z| > R_n) \leq \frac{1}{n}$. The existence of such a sequence follows from a straightforward application of Markov's inequality. Now choose a neural network G_{θ_n} with weights θ_n such that

$$\sup_{\|z\|\leq R_n} \|G(z) - G_{\theta_n}(z)\| \leq \frac{1}{n}.$$

The existence of such a neural networks is guaranteed by the universal approximation theorem (Theorem 3.1.1) as the function G is continuous.
In order to show convergence in distribution of $G_\theta(Z)$ to X we make use of the following equivalence [58, Theorem 7.5]:

A sequence X_n of random variables converges in distribution to X if and only if $\mathbb{E}[f(X_n)]$ converges to $\mathbb{E}[f(X)]$ for all bounded Lipschitz-continuous functions f.

Now pick an arbitrary bounded Lipschitz continuous function f with upper bound C_f on the absolute value and Lipschitz constant L_f. Then we have

$$\begin{aligned}
|\mathbb{E}[f(G_{\theta_n}(Z))] - \mathbb{E}[f(X)]| &= |\mathbb{E}[f(G_{\theta_n}(Z))] - \mathbb{E}[f(G(Z))]| \\
&\leq \mathbb{E}[|f(G_\theta) - f(G(Z))|] \\
&= \mathbb{E}[|f(G_\theta(Z)) - f(G(Z))| \ \mathbb{1}_{\{|Z|\leq R_n\}}] + \\
&\quad \mathbb{E}[|f(G_\theta(Z)) - f(G(Z))| \ \mathbb{1}_{\{|Z|>R_n\}}] \\
&\leq L_f \mathbb{E}[|G_{\theta_n}(Z) - G(Z)|\mathbb{1}_{\{|Z|\leq R_n\}}] + 2C_f\mathbb{P}(|Z| > R_n) \\
&\leq (L_f + 2C_f)\frac{1}{n}.
\end{aligned}$$

This proves the weak convergence. □

Next, we provide the proof of the strategic equivalence of the MM-GAN and the NS-GAN.

Lemma A.0.1 (Strategic equivalence of MM-GAN and NS-GAN)**.** *The MM-GAN and the NS-GAN are strategically equivalent.*

Proof. First observe that the generator-discriminator pair $(\bar{G}, \bar{D}) = (\mathbb{P}_{\text{data}}, \frac{1}{2})$ (we identify G with its distribution) is in a Nash equilibrium for both games. It also is the unique Nash equilibrium in both cases as we now show. For the MM-GAN and the NS-GAN a candidate (G, D) for a Nash equilibrium must satisfy $D(x) = \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}}+\mathbb{P}_G)}(x)$ as otherwise we can improve the payoff for the discriminator by changing D. We now show that we can improve the generator payoff for both the MM-GAN and the NS-GAN if $\mathbb{P}_{\text{data}} \neq \mathbb{P}_G$. For the MM-GAN we change the generator to $\mathbb{P}_m = \frac{\mathbb{P}_{\text{data}}+\mathbb{P}_G}{2}$ and leave the discriminator unchanged. In order to confirm that this indeed improves the generator payoff we need to show that

$$\mathbb{E}_{\mathbb{P}_m}\left[\log\left(1 - \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] < \mathbb{E}_{\mathbb{P}_G}\left[\log\left(1 - \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right].$$

To check this we reformulate the above inequality as follows:

$$\begin{aligned}
& \mathbb{E}_{\mathbb{P}_m}\left[\log\left(1 - \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] < \mathbb{E}_{\mathbb{P}_G}\left[\log\left(1 - \frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] \\
\Leftrightarrow \quad & \mathbb{E}_{\mathbb{P}_m}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] < \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] \\
\Leftrightarrow \quad & \mathbb{E}_{\mathbb{P}_m}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\mathbb{P}_m}\right)\right] < \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\mathbb{P}_m}\right)\right] \\
\Leftrightarrow \quad & \mathbb{E}_{\mathbb{P}_m}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_G}{\mathrm{d}\mathbb{P}_m}\right)\right] < d_{\text{KL}}(\mathbb{P}_m \| \mathbb{P}_G)
\end{aligned}$$

Now an application of Jensen's inequality to the left side of the last inequality shows that it is negative whereas the right hand side is a KL-divergence and therefore positive. Hence, the replacement has indeed improved the payoff for the generator. For the NS-GAN we change the generator to $\mathbb{P}_{\text{data}}$. This is an improvement if

$$\mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] > \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right].$$

This again can easily be shown as follows:

$$\begin{aligned}
& \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] > \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}(\mathbb{P}_{\text{data}} + \mathbb{P}_G)}\right)\right] \\
\Leftrightarrow \quad & \mathbb{E}_{\mathbb{P}_{\text{data}}}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right)\right] > \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right)\right] \\
\Leftrightarrow \quad & d_{KL}(\mathbb{P}_{\text{data}} \| \mathbb{P}_m) > \mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right)\right]
\end{aligned}$$

The KL-divergence is strictly positive as $\mathbb{P}_{\text{data}} \neq \mathbb{P}_G$ by assumption and hence it suffices to show negativity of the term on the right which can easily be achieved by an application of Jensen's inequality:

$$\mathbb{E}_{\mathbb{P}_G}\left[\log\left(\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right)\right] \leq \log\left(\underbrace{\mathbb{E}_{\mathbb{P}_G}\left[\frac{\mathrm{d}\mathbb{P}_{\text{data}}}{\mathrm{d}\mathbb{P}_m}\right]}_{<1}\right) < \log(1) = 0$$

□

Remark A.0.1. *In the above we have implicitely assumed that there are generators G for which the* *transformed random variable $G(Z)$ has the same distribution as some prescribed measure. This might not always be the case but in the light of theorem 3.1.2 it is a reasonable assumption.*

Appendix B

Estimating the multivariate t-distribution

In this appendix we give a review of some of the existing expectation maximisation (EM) based approaches for estimating a multivariate t-distribution. In particular, we will introduce the ECME algorithm by Liu and Rubin [35] which we will use in our applications. In all the content presented we follow [38] and [35].
Generally speaking, expectation maximisation methods allow to compute maximum likelihood estimates when the available samples are in some sense incomplete. There are essentially two kinds of incompleteness: Either the data samples might have some missing values (i.e. failed recordings) or there exists an unobservable latent variable which allows a "nice" representation of the joint likelihood of this latent variable and the observed variables. The multivariate t-distribution falls into the second category. EM algorithms then find the maximum-likelihood by an iterative procedure consisting of *Expectation Steps (E-steps)* and *Maximisation Steps (M-steps)*. In the former, the true likelihood function is approximated by its conditional expectation under the distribution corresponding to the current parameter set. In the M-step, this approximated likelihood function is then optimized to obtain a new parameter set. These two steps are repeated until convergence is achieved.
More formally, let $X = (X_1, ..., X_p)^T$ be the vector of observed random variables and $Z = (Z_1, ..., Z_q)^T$ the corresponding vector of latent random variables. Further assume that the completed random vector $Y = (X^T, Z^T)^T$ follows some distribution with density $f_Y(y;\theta)$ depending on a set $\theta = (\theta_1, ..., \theta_d)^T$ of parameters which are to be estimated. Given a set of complete samples $Y = \left(Y^{(i)}\right)_{i=1}^n$ we write

$$\ell(\theta; Y) = \sum_{i=1}^n \log \left(f_Y(X_1^{(i)}, ..., X_p^{(i)}, Z_1^{(i)}, ..., Z_q^{(i)}) \right)$$

for the log-likelihood function. The EM method now proceeds by first choosing some set $\theta^{(0)}$ of initial values for the parameters and then iteratively performing the following steps:
E-Step: Compute the *completed likelihood function*

$$\begin{aligned} Q(\theta; \theta^{(k)}) &= \mathbb{E}_{\theta^{(k)}}[\ell(\theta; Y)|X] \\ &= \sum_{i=1}^n \mathbb{E}_{\theta^{(k)}} \left[\log \left(f_Y(X_1^{(i)}, ..., X_p^{(i)}, Z_1^{(i)}, ..., Z_q^{(i)}) \right) \Big| \left(X_j^{(i)} \right)_{j=1}^p \right] \end{aligned}$$

where the expectation is taken under the measure corresponding to the current parameter set $\theta^{(k)}$.
M-Step: In the maximisation step one then finds the next parameter set by maximizing the completed likelihood function:

$$\theta^{(k+1)} = \underset{\theta}{\operatorname{argmax}}\ Q(\theta; \theta^{(k)})$$

In 1993, Meng and Rubin [40] introduced the class of *Expectation Conditional Maximisation (ECM)* methods which can often speed up the EM procedure. In these methods the M-step is replaced by a sequence of *conditional maximisation steps (CM-steps)* which are computationally easier to perform. In each of these conditional maximisation steps the completed log-likelihood function from the previous E-step is maximised subject to some constraints. The constraints are chosen such that their collection amouts to maximizing over the whole domain. For further details we refer the reader to [38, Section 5.2]. A further extension proposed in [40] is the *multicycle ECM* algorithm in which an E-Step is performed between all the CM-steps. Finally, Liu and Rubin [35] extended the multicycle ECM to the ECME where the last "E" stands for "either" which means that in some CM-steps the completed likelihood-function is maximised and in others the real log-likelihood function. Again, an E-step is performed between all CM-steps. For details we refer the reader to the paper [35].
We now apply the algorithms described above to the estimation of the multivariate t-distribution $t_p(\nu, \mu, \Sigma)$ with density

$$f(x) = \frac{\Gamma((\nu+d)/2)}{\Gamma(\nu/2)\nu^{d/2}\pi^{d/2}|\Sigma|^{1/2}} \left[1 + \frac{1}{\nu}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]^{-(\nu+d)/2}. \tag{B.1}$$

To do so we make use of the following representation of the t-distribution as a variance-mixture distribution:

$$X|\tau \sim \mathcal{N}(\mu, \Sigma/\tau), \quad \tau \sim \Gamma\left(\frac{\nu}{2}, \frac{\nu}{2}\right) \tag{B.2}$$

The density of a $\Gamma(\alpha, \beta)$-distributed random variable is given by

$$f(\tau) = \frac{\beta^\alpha \tau^{\alpha-1}\exp(-\beta\tau)}{\Gamma(\alpha)}.$$

Representation (B.2) allows to interpret the multivariate t-distribution as a normal distribution conditional on some latent variable τ which again follows a gamma distribution. Using this factorization the joint density of X and τ can easily be computed to be

$$\begin{aligned} f_Y(x,\tau) &= f_{X|\tau}(x|\tau)f_\tau(\tau) \\ &= \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)\frac{(\nu/2)^{\nu/2}\tau^{\nu/2-1}\exp(-\nu\tau/2)}{\Gamma(\nu/2)}. \end{aligned}$$

Integrating the latent variable τ out gives the density in (B.1). As the gamma distribution is a conjugate prior for the normal likelihood function, the posterior distribution of τ is known to be

$$\tau|X \sim \Gamma\left(\frac{\nu+p}{2}, \frac{\nu+\delta_X(\mu,\Sigma)}{2}\right) \tag{B.3}$$

where we denote by

$$\delta_X(\mu, \Sigma) = (X - \mu)^T \Sigma^{-1} (X - \mu)$$

the squared Mahalanobis distance between X and μ. For the application of the EM algorithm we need the following two expectations for a random variable $\tau \sim \Gamma(\alpha, \beta)$:

$$\mathbb{E}[\tau] = \frac{\alpha}{\beta}$$
$$\mathbb{E}[\log(\tau)] = \psi(\alpha) - \log(\beta)$$

Here $\psi(x) = \frac{\partial}{\partial x} \log(\Gamma(x))$ denotes the digamma function. In particular, for a sample (X, τ) we get

$$\mathbb{E}[\tau | X] = \frac{\nu + p}{\nu + \delta_X(\mu, \Sigma)} \tag{B.4}$$
$$\mathbb{E}[\log(\tau) | X] = \psi\left(\frac{\nu + p}{2}\right) - \log\left(\frac{\nu + \delta_X(\mu, \Sigma)}{2}\right) \tag{B.5}$$
$$\tag{B.6}$$

We are now ready to perform the E-step. Denote by $\theta^{(k)} = (\nu^{(k)}, \mu^{(k)}, \Sigma^{(k)})$ the current parameter estimate. Making use of equations (B.4), (B.5) and (B.3) the completed log-likelihood function $Q(\theta; \theta^{(k)})$ can be computed to be

$$\begin{aligned} Q(\theta; \theta^{(k)}) = &-\frac{np}{2}\log(2\pi) - \frac{n}{2}\log(|\Sigma|) - \frac{1}{2}\sum_{i=1}^{n} \delta_{X_i}(\mu, \Sigma) u_i^{(k)} \\ &- \frac{p}{2}\sum_{i=1}^{n}\left[\log(u_i^{(k)}) + \psi\left((\nu^{(k)} + p)/2\right) - \log\left((\nu^{(k)} + p)/2\right)\right] - n\log(\Gamma(\nu/2)) \\ &+ \frac{\nu n}{2}\log(\nu/2) + \frac{\nu}{2}\sum_{i=1}^{n}\left[\log(u_i^{(k)}) - u_i^{(k)}\right] \\ &+ \frac{\nu n}{2}\left[\psi\left((\nu^{(k)} + p)/2\right) - \log\left((\nu^{(k)} + p)/2\right)\right] \\ &- \sum_{i=1}^{n}\left[\log\left(u_i^{(k)}\right) + \psi\left((\nu^{(k)} + p)/2\right) - \log\left((\nu^{(k)} + p)/2\right)\right] \end{aligned} \tag{B.7}$$

where we write

$$u_i^{(k)} = \mathbb{E}_{\theta^{(k)}}[\tau^{(i)} | X^{(i)}] = \frac{\nu^{(k)} + p}{\nu^{(k)} + \delta_{X^{(i)}}(\mu^{(k)}, \Sigma^{(k)})}$$

for the expected value of $\tau^{(i)}$ conditional on $X^{(i)}$ under the distribution corresponding to the current parameter set $\theta^{(k)}$.

We can now perform the M-step by setting the partial derivatives of $Q(\theta; \theta^{(k)})$ to zero. For the location vector μ this gives

$$\frac{\partial}{\partial \mu} Q(\theta, ; \theta^{(k)}) = -\sum_{i=1}^{n} \Sigma^{-1}(\mu - X^{(i)}) u_i^{(k)}$$

and so we obtain

$$\mu^{(k+1)} = \frac{\sum_{i=1}^n X^{(i)} u_i^{(k)}}{\sum_{i=1}^n u_i^{(k)}}. \tag{B.8}$$

Similarly, the derivative with respect to Σ^{-1} is

$$\frac{\partial}{\partial \Sigma^{-1}} Q(\theta; \theta^{(k)}) = \frac{n}{2}\Sigma - \frac{1}{2}\sum_{i=1}^n (X^{(i)} - \mu)(X^{(i)} - \mu)^T u_i^{(k)}$$

which gives

$$\Sigma^{(k+1)} = \frac{1}{n}\sum_{i=1}^n (X^{(i)} - \mu^{(k+1)})(X^{(i)} - \mu^{(k)})^T u_i^{(k)}. \tag{B.9}$$

The new estimate for ν can then be found by setting the corresponding partial derivative to zero and solving for ν:

$$\begin{aligned} \frac{\partial}{\partial \nu} Q(\theta; \theta^{(k)}) = -\psi(\nu/2) + \log(\nu/2) + 1 + \frac{1}{n}\sum_{i=1}^n \left[\log\left(u_i^{(k)}\right) - u_i^{(k)}\right] \\ + \psi((\nu^{(k)} + p)/2) - \log((\nu^{(k)} + p)/2) \overset{!}{=} 0 \end{aligned}$$

Algorithm 4 summarizes the procedure.
In the ECM algorithm for the estimation of the multivariate t-distribution the first CM-step is to maximise the completed log-likelihood function $Q(\theta; \theta^{(k)})$ only with respect to μ and Σ which again gives $\mu^{(k+1)}$ and $\Sigma^{(k+1)}$ as in (B.8) and (B.9). In the second CM-step one then maximises the updated completed log-likelihood function $Q(\nu, \mu^{(k+1)}, \Sigma^{(k+1)}; \theta^{(k)})$ with respect to ν. However, as the degrees of freedom is decoupled from the location and scale matrix these constraint steps are equivalent to the M-step in the regular EM method. However, if we add E-steps between the two CM-steps we obtain a multicycle version of ECM which is not identical to the EM algorithm. The ECME algorithm is now the same as the multicycle ECM besides that in the second CM-step we maximise the true log-likelihood function which amounts to finding a solution of

$$\begin{aligned} 0 \overset{!}{=} -\psi(\nu/2) + \log(\nu/2) + 1 + \frac{1}{n}\sum_{i=1}^n \left[\log\left(u_i^{(k)}(\nu)\right) - u_i^{(k)}(\nu)\right] \\ + \psi((\nu^{(k)} + p)/2) - \log((\nu^{(k)} + p)/2) \end{aligned}$$

where

$$u_i^{(k)}(\nu) = \frac{\nu + p}{\nu + \delta_{X^{(i)}}(\mu^{(k+1)}, \Sigma^{(k+1)})}.$$

The full procedure for both multicycle ECM and ECME is shown in algorithm 5.

Algorithm 4 Expectation Maximisation algorithm for estimating the parameters ν, μ and Σ of a p-dimensional t-distribution.

```
    Input: Samples $X^{(1)}, ..., X^{(n)}$ of a $p$-dimensional random variable,
    error tolerance $\varepsilon > 0$, initial value $\nu^{(0)}$ for the degrees of freedom
    Output: Estimates $\hat{\nu}, \hat{\mu}, \hat{\Sigma}$ of the location $\mu$, the scale matrix $\Sigma$ and the degrees
    of freedom $\nu$
1: Initialize the location and scale matrix with the empirical mean and covariance matrix:
       $$\mu^{(0)} := \frac{1}{n}\sum_{i=1}^{n} X^{(i)}, \quad \Sigma^{(0)} := \frac{1}{n}\sum_{i=1}^{n}(X^{(i)} - \mu^{(0)})(X^{(i)} - \mu^{(0)})^T$$
2: converged ← False
3: while not converged do
4:     E-Step: For $i = 1, ..., n$ set
           $$u_i^{(k)} = \frac{\nu^{(k)} + p}{\nu^{(k)} + \delta_{X^{(i)}}(\mu^{(k)}, \Sigma^{(k)})}.$$
5:     M-Step: Update $\mu^{(k+1)}$ and $\Sigma^{(k+1)}$ using the closed form solutions:
           $$\mu^{(k+1)} = \frac{\sum_{i=1}^{n} X^{(i)} u_i^{(k)}}{\sum_{i=1}^{n} u_i^{(k)}}, \quad \Sigma^{(k+1)} = \frac{1}{n}\sum_{i=1}^{n}(X^{(i)} - \mu^{(k)})(X^{(i)} - \mu^{(k)})^T u_i^{(k+1)}$$
       Set $\nu^{(k+1)}$ to be the root of
           $$0 = -\psi(\nu/2) + \log(\nu/2) + 1 + \frac{1}{n}\sum_{i=1}^{n}\left[\log\left(u_i^{(k)}\right) - u_i^{(k)}\right] + \psi((\nu^{(k)} + p)/2) - \log((\nu^{(k)} + p)/2).$$
6:     if $|\ell(\theta^{(k+1)}) - \ell(\theta^{(k)})| < \varepsilon$ then
7:         converged ← True
8:     end if
9: end while
```

Algorithm 5 ECM and ECME algorithms for estimating the parameters ν, μ and Σ of a p-dimensional t-distribution.

```
Input: Samples $X^{(1)}, ..., X^{(n)}$ of a $p$-dimensional random variable, error tolerance $\varepsilon > 0$, initial value $\nu^{(0)}$ for the degrees of freedom
Output: Estimates $\hat{\nu}, \hat{\mu}, \hat{\Sigma}$ of the location $\mu$, the scale matrix $\Sigma$ and the degrees of freedom $\nu$
 1: Initialize the location and scale matrix with the empirical mean and covariance matrix:
        $$\mu^{(0)} := \frac{1}{n}\sum_{i=1}^{n} X^{(i)}, \quad \Sigma^{(0)} := \frac{1}{n}\sum_{i=1}^{n}(X^{(i)} - \mu^{(0)})(X^{(i)} - \mu^{(0)})^T$$
 2: converged ← False
 3: while not converged do
 4:     E-Step 1: For $i = 1, ..., n$ set $u_i^{(k)} = \frac{\nu^{(k)}+p}{\nu^{(k)}+\delta_{X^{(i)}}(\mu^{(k)},\Sigma^{(k)})}$.
 5:     CM-Step 1: Update $\mu^{(k+1)}$ and $\Sigma^{(k+1)}$ using the closed form solutions:
            $$\mu^{(k+1)} = \frac{\sum_{i=1}^{n} X^{(i)} u_i^{(k)}}{\sum_{i=1}^{n} u_i^{(k)}}, \quad \Sigma^{(k+1)} = \frac{1}{n}\sum_{i=1}^{n}(X^{(i)} - \mu^{(k)})(X^{(i)} - \mu^{(k)})^T u_i^{(k+1)}$$
 6:     E-Step 2: For $i = 1, ..., n$ set $u_i^{(k)} = \frac{\nu^{(k)}+p}{\nu^{(k)}+\delta_{X^{(i)}}(\mu^{(k+1)},\Sigma^{(k+1)})}$.
 7:     CM-Step2:
         ECM: Set $\nu^{(k+1)}$ to be the root of
            $$0 = -\psi(\nu/2) + \log(\nu/2) + 1 + \frac{1}{n}\sum_{i=1}^{n}\left[\log\left(u_i^{(k)}\right) - u_i^{(k)}\right] + \psi((\nu^{(k)} + p)/2) - \log((\nu^{(k)} + p)/2).$$
         ECME: Set $\nu^{(k+1)}$ to be the root of
            $$0 = -\psi(\nu/2) + \log(\nu/2) + 1 + \frac{1}{n}\sum_{i=1}^{n}\left[\log\left(u_i^{(k)}(\nu)\right) - u_i^{(k)}(\nu)\right] + \psi((\nu^{(k)} + p)/2) - \log((\nu^{(k)} + p)/2).$$
        where
            $$u_i^{(k)}(\nu) = \frac{\nu + p}{\nu + \delta_{X^{(i)}}(\mu^{(k+1)}, \Sigma^{(k+1)})}$$
 8:     if $|\ell(\theta^{(k+1)}) - \ell(\theta^{(k)})| < \varepsilon$ then
 9:         converged ← True
10:     end if
11: end while
```

Appendix C

Estimating the DCC-GARCH model

In this appendix we describe the two-step procedure commonly used to fit the DCC-GARCH model whereby we focus on the specific case of multivariate normal standardized residuals. We write $\theta = (\phi, \psi, P_c)$ for the vector of all parameters. Here $\psi = (a_1, ..., a_P, b_1, ..., b_Q)$ denotes the vector containing all parameters of the conditional correlation matrix and $\phi = (\phi_1, ..., \phi_d)$ holds $\phi_k = (\alpha_{k,1}, ..., \alpha_{k,p_k}, \beta_{k,1}, ..., \beta_{k,q_k})$, the parameters of the univariate GARCH components. Given observations $X^{(1)}, ..., X^{(T)}$, the likelihood function is

$$L(\theta) = \prod_{t=1}^{T} \frac{1}{\sqrt{(2\pi)^d |\Sigma_t(\theta)|}} \exp\left(-\frac{X_t \Sigma_t(\theta) X_t}{2}\right)$$

where we explicitely denote the dependence of Σ_t on the parameters θ. Taking logarithms and using $\Sigma_t = \Delta_t P_t \Delta_t$ we can rewrite the above as follows:

$$\begin{aligned}
\log(L(\theta)) &= -\frac{1}{2}\sum_{t=1}^{T} \big(d\log(2\pi) + \log(|\Sigma_t(\theta)|) + X_t^T \Sigma_t(\theta) X_t\big) \\
&= -\frac{1}{2}\sum_{t=1}^{T} \Big(d\log(2\pi) + \log(|\Delta_t(\phi) P_t(\psi, P_c) \Delta_t(\phi)|) \\
&\quad + X_t^T \Delta_t^{-1}(\phi) P_t^{-1}(\psi, P_c) \Delta_t^{-1}(\phi) X_t\Big) \\
&= -\frac{1}{2}\sum_{t=1}^{T} \Big(d\log(2\pi) + 2\log(|\Delta_t(\phi)|) + \log(|P_t(\psi, P_c)|) \\
&\quad + X_t^T \Delta_t^{-1}(\phi) P_t^{-1}(\psi, P_c) \Delta_t^{-1}(\phi) X_t\Big)
\end{aligned}$$

Here we again indicate the dependence of Δ_t and P_t on ϕ, ψ and P_c. Generally, maximizing this function with respect to θ is very complicated and so one resorts to the following two step procedure which under certain conditions can be proven to give consistent and asymptotically normal estimators [16]

First Step: First one replaces $P_t(\psi, P_c)$ with the identity matrix I_d which simplifies the log-likelihood function to

$$\begin{aligned}
\log(L^*(\theta)) &= -\frac{1}{2}\sum_{t=1}^{T}\left(d\log(2\pi) + 2\log(|\Delta_t(\phi)|) + \log(|I_d|) + X_t^T\Delta_t^{-1}(\phi)I_d\Delta_t^{-1}(\phi)X_t\right) \\
&= -\frac{1}{2}\sum_{t=1}^{T}\left(d\log(2\pi) + 2\log(|\Delta_t(\phi)|) + X_t^T\Delta_t^{-1}(\phi)I_d\Delta_t^{-1}(\phi)X_t\right) \\
&= -\frac{1}{2}\sum_{t=1}^{T}\left(d\log(2\pi) + \sum_{i=1}^{n}\left[\log(\sigma_{t,i}^2) + \frac{X_{t,i}^2}{\sigma_{t,i}^2}\right]\right) \\
&= \sum_{i=1}^{n}\left(-\frac{1}{2}\sum_{t=1}^{T}\left[\log(\sigma_{t,i}^2) + \frac{X_{t,i}^2}{\sigma_{t,i}^2}\right] + C\right) \qquad \text{(C.1)}
\end{aligned}$$

with some constant C. We immediately see that this is the sum of the log-likelihood functions of the component processes. Hence, maximizing this modified log-likelihood is equivalent to fitting each univariate model separately. In (C.1) we need to provide values for X_t, $\sigma_{t,i}$ and P_t for times $t < 0$. One usually sets these values to zero.

After the parameter ϕ has been computed we can then extract the conditional volatilities $\sigma_{t,i}$ and compute the standardized residuals $Z_{t,i} = X_{t,i}/\sigma_{t,i}$. From this devolatized process we can then estimate the stationary correlation matrix P_c using classical estimators. After these steps have been performed it only remains to estimate ψ.

Second Step: In the second step we now treat the log-likelihood function as a function of ψ only which, ignoring constants, gives

$$\log(L^{**}(\psi)) = -\frac{1}{2}\sum_{t=1}^{T}\left(\log(|P_t|) + Z_t^T P_t^{-1} Z_t\right).$$

This function can then be mxaimized with respect to ψ using any optimization algorithm.

The above method can now easily be extended to the multivariate t-distribution for which we must additionally estimate the degrees of freedom ν.

Appendix D

Additional material for chapter 4

In this appendix we provide additional material for chapter 4. We start with the unconditional case and then discuss the conditional one.

D.1 Unconditional VaR

Table D.1 shows the expected and actual number of exceptions for the 99% VaR and figure D.1 visualized the backtesting behaviour for the GAN modelling the portfolio returns. Again the GAN modelling the portfolio returns performs considerably better than all of its competitors apart from the kernel density estimation method which performs equally well.

Year	2014	2015	2016	2017	2018	Overall
Trading days	252	252	252	251	251	1258
Expected no. of exceptions	2.5	2.5	2.5	2.5	2.5	12.6
VC	1 (0.27)	9 (0.00)	5 (0.17)	2 (0.74)	16 (0.00)	33 (0.00)
VC-t	0 (0.02)	5 (0.17)	3 (0.77)	0 (0.02)	9 (0.00)	17 (0.23)
HS	0 (0.02)	6 (0.06)	3 (0.77)	0 (0.02)	12 (0.00)	21 (0.03)
KDE	0 (0.02)	5 (0.17)	3 (0.77)	0 (0.02)	7 (0.02)	15 (0.51)
GAN-RF	0 (0.02)	7 (0.02)	4 (0.39)	0 (0.02)	13 (0.00)	24 (0.00)
GAN-P	0 (0.02)	4 (0.39)	3 (0.77)	0 (0.02)	8 (0.01)	15 (0.51)

Table D.1: Number of exceptions of the 99% Value-at-Risk calculated using the classical as well as the GAN methods. The value in brackets is the p-value of Kupiec's POF test. The abbreviations of the methods are as follows: Variance-Covariance (VC), t-distribution (VC-t), Historical Simulation (HS), Kernel Density Estimator (KDE), GAN on Risk-Factor level (GAN-RF), GAN on Portfolio level (GAN-P).

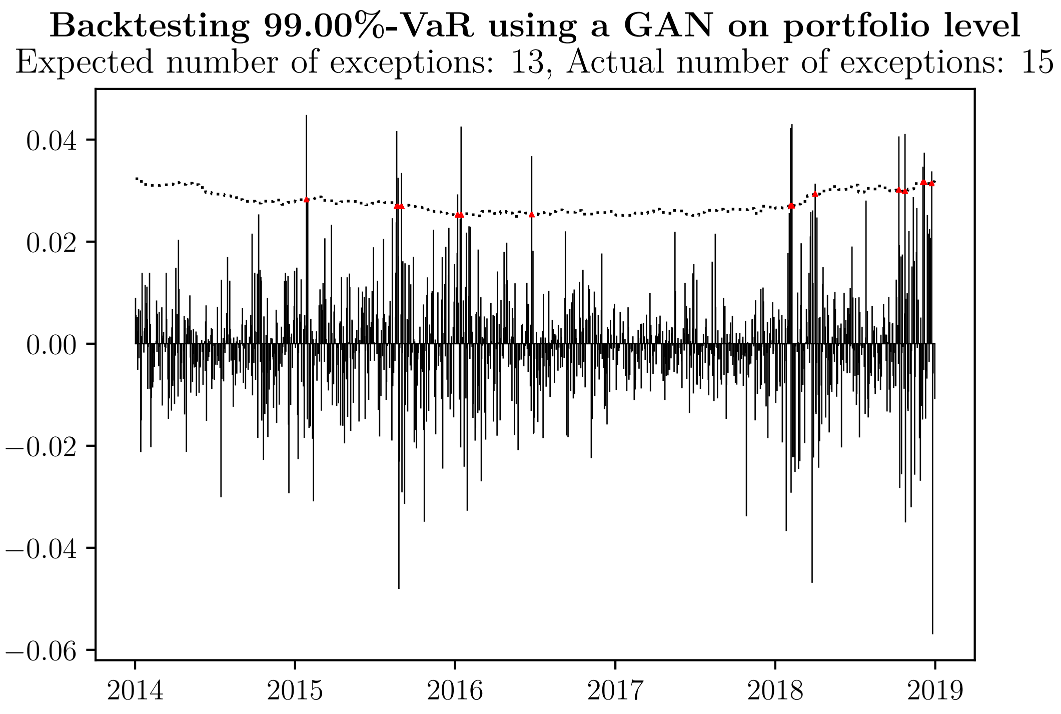

Figure D.1: Backtesting the 99%-VaR forecast performance of the GAN model that samples from the equal-weight portfolio log-returns. The vertical lines indicate the negative daily log-returns and the dotted line gives the corresponding VaR forecast. Exceptions are indicated by red triangles.

D.2 Conditional VaR

In this section we show some plots that have been omitted in the main part of the thesis. Figure D.2 shows the partial autocorrelation function of the time series generated by the RCGAN and the corresponding true time series. Similary, figure D.3 shows the corresponding PACF of the squared time series. Both figures further strengthen the assertion that the GAN has indeed learned the true dynamics. We further show the superiority of the univariate GARCH(1,1) model with t-distributed in 95% VaR backtesting in figure D.4.

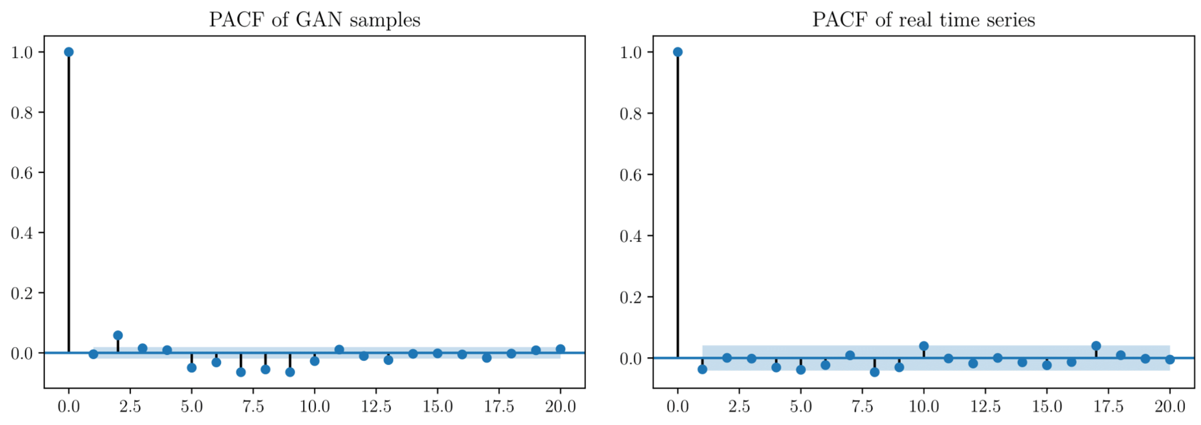

Figure D.2: PACF of the time series generated by the RCGAN (left) and of the real time series (right).

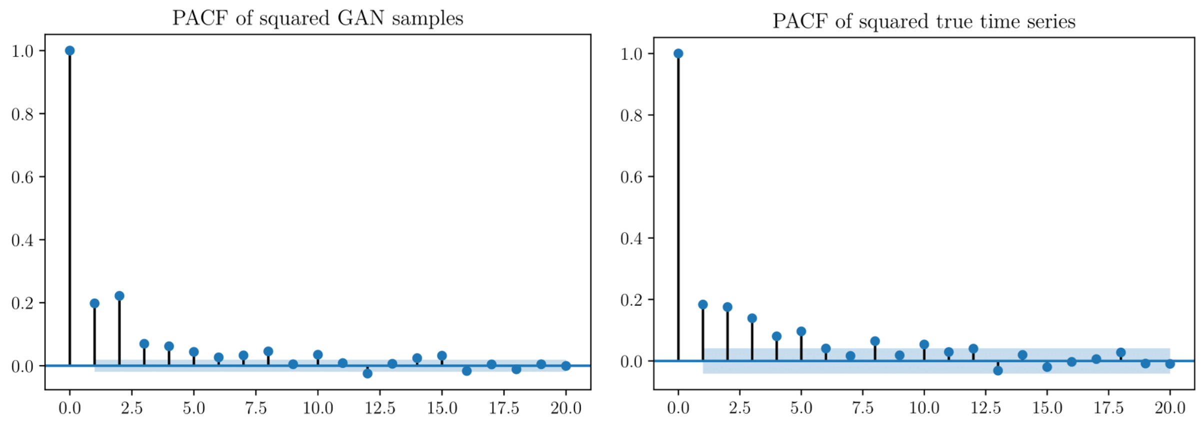

Figure D.3: PACF of the squared time series generated by the RCGAN (left) and of the squared real time series (right).

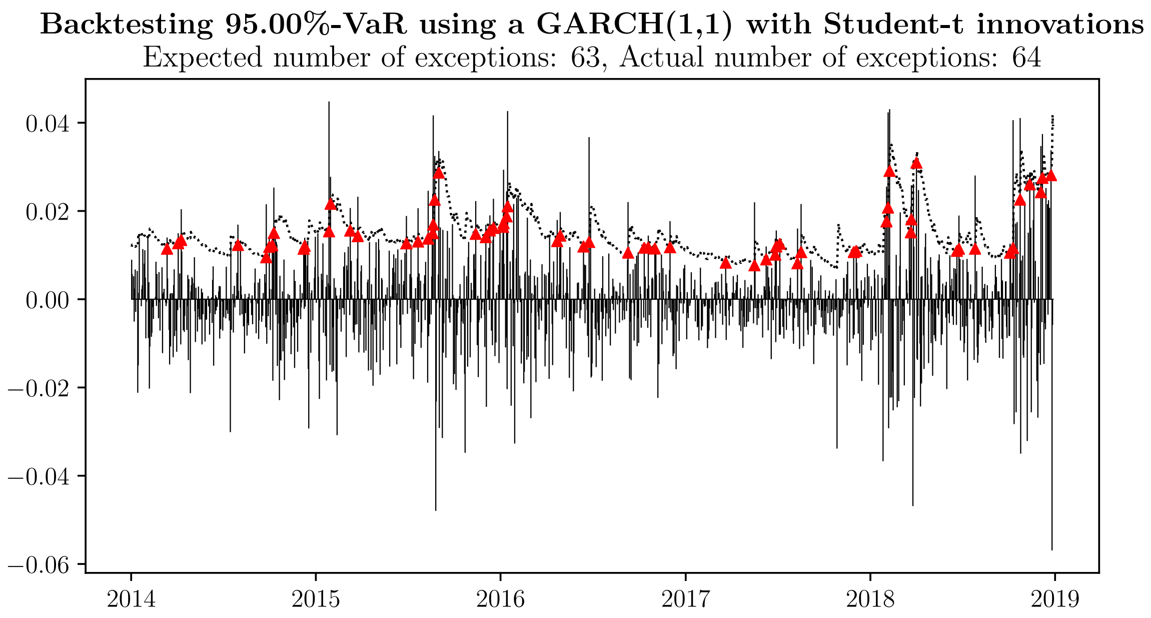

Figure D.4: 95% VaR backtesting performance of the univariate GARCH(1,1) model with *t*-distributed innovations.

References

[1] Apple Inc. (AAPL). Historical Data. Yahoo! Finance, 2019. Available at: https://uk.finance.yahoo.com/quote/AAPL/history/ (Retrieved May 25, 2019, from the Yahoo! Finance database).

[2] Dinesh Acharya, Zhiwu Huang, Danda Pani Paudel, and Luc van Gool. Towards High Resolution Video Generation with Progressive Growing of Sliced Wasserstein GANs. ETH Zurich, pages 1-48, 2018. Available at: http://arxiv.org/pdf/1810.02419v2 (Retrieved June 14, 2019, from the arXiv database).

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. ArXiv, pages 1-32, 2017. Available at: http://arxiv.org/pdf/1701.07875v3 (Retrieved May 09, 2019, from the arXiv database).

[4] Philippe Artzner, Freddy Delbaen, Jean-Marc Eber, and David Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[5] Basel Committee on Banking Supervision. Basel II: International Convergence of Capital Measurement and Capital Standards: A Revised Framework - Comprehensive Version. Bank for International Settlements, pages 1-333, 2006. Available at: https://www.bis.org/publ/bcbs128.pdf (Retrieved May 06, 2019, from the BIS database).

[6] M. Berger, B. Eckmann, P. Harpe, F. Hirzebruch, N. Hitchin, L. Hörmander, A. Kupiainen, G. Lebeau, M. Ratner, D. Serre, and Ya. G. Sinai. *Optimal Transport: Old and New*, volume 338 of *Grundlehren der mathematischen Wissenschaften*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[7] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[8] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. ArXiv, pages 1-10, 2019. Available at: http://arxiv.org/pdf/1809.11096v2 (Retrieved June 02, 2019, from the arXiv database).

[9] Umberto Cherubini, Elisa Luciano, and Walter Vecchiato. *Copula methods in finance*. Wiley finance series. Wiley, Chichester, reprinted. edition, 2006.

[10] Soumith Chintala, Emily Denton, Martin Arjovsky, and Michael Mathieu. How to Train a GAN? Tips and tricks to make GANs work. GitHub, 2016. Available at: https://github.com/soumith/ganhacks (Retrieved June 17, 2019, from GitHub).

[11] Peter F. Christoffersen. Evaluating interval forecasts. *International Economic Review*, 39(4):841, 1998.

[12] R. Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1(2):223–236, 2001.

[13] Cisco Systems Inc. (CSCO). Historical Data. Yahoo! Finance, 2019. Available at: https://finance.yahoo.com/quote/csco/history/ (Retrieved May 25, 2019, from the Yahoo! Finance database).

[14] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial Feature Learning. ArXiv, pages 1-18, 2017. Available at: http://arxiv.org/pdf/1605.09782v7 (Retrieved May 20, 2019, from the arXiv database).

[15] D. M. Endres and J. E. Schindelin. A new metric for probability distributions. *IEEE Transactions on Information Theory*, 49(7):1858–1860, 2003.

[16] Robert Engle and Kevin Sheppard. *Theoretical and Empirical properties of Dynamic Conditional Correlation Multivariate GARCH.* National Bureau of Economic Research, Cambridge, MA, 2001.

[17] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4):987, 1982.

[18] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs. ArXiv, pages 1-13, 2017. Available at: http://arxiv.org/pdf/1706.02633v2 (Retrieved May 27, 2019, from the arXiv database).

[19] William Fedus, Mihaela Rosca, Balaji Lakshminarayanan, Andrew M. Dai, Shakir Mohamed, and Ian Goodfellow. Many Paths to Equilibrium: GANs Do Not Need to Decrease a Divergence At Every Step. ArXiv, pages 1-21, 2017. Available at: http://arxiv.org/pdf/1710.08446v3 (Retrieved June 17, 2019, from the arXiv database).

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT Press, Cambridge, Massachusetts and London, England, 2016.

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. Universit de Montral, 2014, pages 1-8, Montral. Available at: http://arxiv.org/pdf/1406.2661v1 (Retrieved June 08, 2019, from the arXiv database).

[22] Artur Gramacki. *Nonparametric Kernel Density Estimation and Its Computational Aspects*, volume 37 of *Studies in Big Data.* Springer International Publishing, Cham, 2018.

[23] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. ArXiv, pages 1-20, 2017. Available at: http://arxiv.org/pdf/1704.00028v3 (Retrieved June 10, 2019, from the arXiv database).

[24] Marcus Haas. New methods in backtesting. financial engineering research center, pages 1-20, 2001. available at: https://www.ime.usp.br/ rvicente/risco/haas.pdf (retrieved june 09, 2019).

[25] Hamaad Shah. Using Bidirectional Generative Adversarial Networks to estimate Value-at-Risk for Market Risk Management. Towards Data Science, 2018. Available at: https://towardsdatascience.com/using-bidirectional-generative-adversarial-networks-to-estimate-value-at-risk-for-market-risk-c3dffbbde8dd (Retrieved June 21, 2019).

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[27] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[28] Intel Corporation (INTC). Historical Data. Yahoo! Finance, 2019. Available at: https://finance.yahoo.com/quote/intc/history?ltr=1 (Retrieved May 25, 2019, from the Yahoo! Finance database).

[29] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. ArXiv, pages 1-15, 2017. Available at: http://arxiv.org/pdf/1412.6980v9 (Retrieved June 24, 2019, from the arXiv database).

[30] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[31] Paul H. Kupiec. Techniques for verifying the accuracy of risk measurement models. *The Journal of Derivatives*, 3(2):73–84, 1995.

[32] Jiwei Li, Will Monroe, Tianlin Shi, Sébastien Jean, Alan Ritter, and Dan Jurafsky. Adversarial Learning for Neural Dialogue Generation. ArXiv, 1-13, 2017. Available at: http://arxiv.org/pdf/1701.06547v5 (Retrieved June 13, 2019, from the arXiv database).

[33] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael P. Wellman. Generating Realistic Stock Market Order Streams. OpenReview, 2019. Available at: https://openreview.net/forum?id=rke41hC5Km (Retrieved June 05, 2019, from the OpenReview database).

[34] Yujia Li, Kevin Swersky, and Richard Zemel. Generative Moment Matching Networks. ArXiv, pages 1-9, 2015. Available at: http://arxiv.org/pdf/1502.02761v1 (Retrieved June 05, 2019, from the arXiv database).

[35] Chuanhai Liu and Donald B. Rubin. Ml estimation of the t-distribution using em and its extensions, ecm and ecme. statistica sinica, pages 1-21, 1995. available at: www3.stat.sinica.edu.tw/statistica/oldpdf/a5n12.pdf (retrieved may 27, 2019). In *Statistica Sinica*.

[36] Padala Manisha and Sujit Gujar. Generative adversarial networks (gans): What it can generate and what it cannot? *ArXiv*, abs/1804.00140, 2018.

[37] Paul Charles Matthews. *Vector calculus*. Springer undergraduate mathematics series. Springer, London, 9. print edition, 2006.

[38] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*. Wiley series in probability and statistics. Wiley-Interscience, Hoboken, NJ, 2. ed. edition, 2008.

[39] Alexander J. McNeil, Rüdiger Frey, and Paul Embrechts. *Quantitative risk management: Concepts, techniques and tools.* Princeton University Press, Princeton, 2015.

[40] XIAO-LI MENG and Donald B. Rubin. Maximum likelihood estimation via the ecm algorithm: A general framework. *Biometrika*, 80(2):267–278, 1993.

[41] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? arxiv, pages 1-39, 2018. available at: http://arxiv.org/pdf/1801.04406v4 (retrieved june 11, 2019, from the arxiv database). 2018.

[42] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. ArXiv, pages 1-7, 2014. Available at: http://arxiv.org/pdf/1411.1784v1 (Retrieved June 12, 2019, from the arXiv database).

[43] Microsoft Corporation (MSFT). Historical Data. Yahoo! Finance, 2019. Available at: https://uk.finance.yahoo.com/quote/MSFT/history/ (Retrieved May 25, 2019, from the Yahoo! Finance database).

[44] Alfred Müller. Integral probability metrics and their generating classes of functions. *Advances in Applied Probability*, 29(2):429–443, 1997.

[45] Saralees Nadarajah and Stephen Chan. Estimation methods for value at risk. In François Longin, editor, *Extreme Events in Finance*, volume 4, pages 283–356. John Wiley & Sons, Inc, Hoboken, New Jersey, 2016.

[46] Roger B. Nelsen. *An introduction to Copulas.* Springer series in statistics. Springer New York, New York, NY, 2. ed. 2006. corr. 2. pr. softcover version of original hardcover edition 2006 edition, 2010.

[47] Frans A. Oliehoek, Rahul Savani, Jose Gallego-Posada, Elise van der Pol, Edwin D. de Jong, and Roderich Gross. GANGs: Generative Adversarial Network Games. ArXiv, pages 1-9, 2017. Available at: http://arxiv.org/pdf/1712.00679v2 (Retrieved June 15, 2019, from the arXiv database).

[48] Pfizer Inc. (PEE). Historical Data. Yahoo! Finance, 2019. Available at: https://finance.yahoo.com/quote/PFE/history/ (Retrieved May 25, 2019, from the Yahoo! Finance database).

[49] Lillian J. Ratliff, Samuel A. Burden, and S. Shankar Sastry. On the characterization of local nash equilibria in continuous games. *IEEE Transactions on Automatic Control*, 61(8):2301–2307, 2016.

[50] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved Techniques for Training GANs. ArXiv, pages 1-10, 2016. Available at: http://arxiv.org/pdf/1606.03498v1 (Retrieved June 05, 2019, from the arXiv database).

[51] Santanu Khan. Calculate Value-at-Risk Using Wasserstein Generative Adversarial Networks (WGAN-GP) for Risk. Mc.ai, Aggregated news around AI and Co, 2019. Available at: https://mc.ai/calculate-value-at-risk-using-wasserstein-generative-adversarial-networks-wgan-gp-for-risk/ (Retrieved June 21, 2019).

[52] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*, volume v.26 of *Chapman and Hall/CRC Monographs on Statistics and Applied Probability.* Routledge, Boca Raton, 2018.

[53] Gillian Tett. *Fool's gold: The inside story of J.P. Morgan and how Wall Street greed corrupted its bold dream and created a financial catastrophe.* Free Press, New York, 1st free press paperback ed. edition, 2010.

[54] Ruey S. Tsay. *Analysis of financial time series.* Wiley series in probability and statistics. Wiley, Hoboken, N.J, third edition edition, 2010.

[55] 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, 2019.

[56] Y. Zhang and S. Nadarajah. A review of backtesting for value at risk. *Communications in Statistics - Theory and Methods*, 47(15):3616–3639, 2018.

[57] Xingyu Zhou, Zhisong Pan, Guyu Hu, Siqi Tang, and Cheng Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018(1):1–11, 2018.

[58] Gordan Zitkovi. Lecture notes in Theory of Probability I: Lecture 7 - Weak convergence. Texas University, pages 1-9, 2013. Available at: https://web.ma.utexas.edu/users/gordanz/notes/weak.pdf (Retrieved June 05, 2019).