LUND UNIVERSITY

Master's Thesis

Department of Statistics

Value at Risk Estimation with Generative Adversarial Networks

Author

David Tobjörk

2021

15 ECTS

Supervisor: Björn Holmquist

ABSTRACT

Risk is of large importance for financial institutions and there are many different measures that can be used. A popular one is value at risk (VaR), which is the maximum likely loss for a portfolio of financial assets. Different methods of estimating it has been suggested, one often described is the variance-covariance method. Here a method based on a generative model called generative adversarial networks (GANs) was considered. More precisely, a type of model presented as a way of improving training of GANs called WGAN-GP was used. By simulating synthetic returns for the portfolio VaR was estimated. Two types of models using WGAN-GP was considered, a model that can generate returns conditioned on recent returns and unconditional models. The models were backtested and compared to a benchmark model using the variance-covariance approach. When evaluating the models it's shown that the conditional model is very sensitive to changes in the conditioning data. The unconditional models however, show similar estimates to the benchmark model over time. But, only the benchmark model is not rejected for the overall backtesting period by Kupiec's POF test.

CONTENTS

CHAPTER 1

INTRODUCTION

In this chapter the risk measure value at risk and ways of estimating it is introduced. Moreover, the aim of the thesis and delimitations are presented. Finally, the disposition for the rest of the thesis is given.

1.1 Background

The concept of risk is an old one and the importance of it was recognized even before there was much theoretical understanding. Early work to understand the risk of a portfolio of assets was done by Markowitz (1952). Markowitz showed mathematically how diversification effects a portfolio of financial assets. He showed that its not the individual assets that are important but their contribution to the portfolio.

Nowadays there are many different measures of financial risk. One common measure is value at risk (VaR) which was popularized in the 1990s and is a commonly used risk measure by financial institutions. Value at risk is the maximum likely loss over a certain period of time, more precisely VaR is the maximum loss that with a certain confidence level will not be exceeded over a certain time period. For example the VaR over a month with a confidence level of 95% corresponds to the 5th percentile loss of the distribution of profits/losses over a month. One apparent advantage of VaR as a risk measure is its interpretability, for example the following statement is easy to interpret:

With 95% confidence the loss over the next month for the stock portfolio will not exceed 10 MSEK.

In this statement the 95% VaR is 10 MSEK, note that VaR it is expressed in terms of loss. If the the 5th percentile for the distribution of returns corresponds to -10 MSEK the 95% VaR is 10 MSEK.

When estimating VaR the relevant market factors of the asset portfolio need to be identified. Which market factors that are relevant depends on the particular portfolio that should be estimated. An example of market factors and VaR estimation is given by Linsmeier and Pearson (1996). They use a portfolio consisting of a single foreign exchange forward contract, in this case the market factors are the spot exchange rate and interests rates for each currency. If we instead are interested in the VaR of a stock portfolio, the price of each stock can be identified as a risk factor.

Furthermore, we need to decide on the method to be used to estimate VaR. There are many different methods that can be used to estimate VaR; three often described ones are the variance-covariance, historical simulation and Monte Carlo simulation method. The variance-covariance method relies on the assumption that the market factors are multivariate normally distributed and thereby also the profits/losses of the portfolio. By fitting the distribution of the profits/losses the usual properties of the normal distribution can be used to calculate VaR (Linsmeier & Pearson, 1996). The historical simulation method makes use of the empirical distribution of the profits/losses. By ordering historic data of returns we get an empirical distribution, the VaR with $1 - \alpha$ confidence level is the α quantile of the empirical distribution expressed in terms of loss. Finally, using Monte Carlo simulation VaR can be estimated by defining a stochastic process for the market factors and simulating their changes. There are many other methods and extensions; one is the modification to the historical simulation approach by Hull and White (1998). In their method they take into account the changes in volatility by adjusting the variables of the empirical distribution by estimates of a GARCH or exponential moving average model. Another type of methods are those that use extreme value theory to estimate VaR, for example Longin (2000), Danielsson, Hartmann, and de Vries (1998) and Marimoutou, Raggad, and Trabelsi (2009). As noted by Marimoutou et al. (2009) VaR estimates is related to the tail of the distribution and the extreme value approaches focus only on this part of the distribution.

In this thesis a simulation approach using generative adversarial networks (GANs) to estimate VaR is considered. GAN is a type of machine learning algorithm presented by Goodfellow et al. (2014) that learns to generate new data. GANs have two main parts, the generator function trying to learn the underlying distribution of the training data and the discriminator function trying to judge whether data is

real or fake (generated from the generator). The generator and discriminator are adversaries in the sense that the generator tries to fool the discriminator to think the fake data is real. A typical application of GANs is to generate new images by learning the distribution from a training set of images.

Previous applications of GANs to VaR estimation include the internet posts by Shah (2018) and Khan (2019), but they do not backtest their models. Fu et al. (2019) use a conditional GAN to estimate VaR, where the GAN is conditioned on whether the market conditions corresponds to a normal or stressed period. Both the generator and discriminator architectures are built up of feed forward networks and they consider a 99% daily VaR for a portfolio consisting of two stocks. To backtest their model they use data of around 1000 business days of stock prices, they do not present any statistical tests for the backtesting but their GAN model is closer in terms of expected number of breaches of VaR than the historical simulation method. The thesis by Fiechtner (2019) considers unconditional and conditional models using GANs for VaR estimation for stock portfolios. The conditional GAN model is conditional in the sense that it is conditioned on a set of previous returns. For the unconditional model Fiechtner considers feed forward networks. In the conditional model the conditioning data is processed through long short-term memory networks (Hochreiter & Schmidhuber, 1997) before it is fed into the generator and discriminator respectively. Unconditional GAN models modelling the risk factors individually and on the portfolio level are backtested and compared to some classical approaches to VaR estimation. Kupiec's POF test does not reject these on the overall period. However, the conditional model is shown to be unstable and is rejected by Kupiec's POF test on the overall period.

Similarly to Fiechtner both models of the unconditional and conditional type are considered here. However, here convolutional neural networks will be considered for the architectures of the GANs and also a larger backtesting data set was used.

This thesis was inspired by Mariani et al. (2019), who use GANs for a different financial application. They use GANs to find efficient portfolios and compare it to the traditional Markowitz approach. They use their GAN to simultaneously generate time series of prices for the assets in the portfolio. Based on the simulated prices they set up an optimization problem with the aim of maximizing expected return and minimizing variance. They compare this to the traditional Markowitz approach on two test portfolios. They conclude that the method using GAN achieves better performance in terms of maximization of expected return and minimization of risk compared to the Markowitz approach. The conditional GAN model used here is similar to the one of Mariani et al. (2019), but instead used for VaR estimation.

1.2 Problem Statement

In this thesis VaR estimation using simulation through GANs and the traditional variance-covariance approach is performed with the aim of answering the following question:

Is estimation of value at risk using generative adversarial networks suitable?

Some delimitations were made. First, VaR estimations were made for a stock portfolio and portfolios consisting of other asset classes were not considered. Second, the benchmark approach is the common variance-covariance method. Third, in this thesis particular architectures of the GANs were considered.

1.3 Disposition

In the next chapter some theory on VaR is given including estimation through the variance-covariance approach and how to evaluate VaR models through backtesting. Chapter 3 gives an introduction to neural networks before generative adversarial networks are described in chapter 4. In chapter 5 the method to estimate VaR using GANs and the data for the empirical study is presented. In chapter 6 the results are presented and analysed. Finally, chapter 7 concludes.

CHAPTER 2

VALUE AT RISK

In this chapter, VaR is defined, followed by an explanation of how it can be estimated using the variance-covariance approach. Thereafter, a method used to evaluate the VaR estimation methods in this thesis is described.

2.1 Definition

As stated earlier VaR is the maximum likely loss for a given confidence level and time horizon. A formal definition of VaR is given by Saita (2007) as

$$Pr(V - V_0 < -\text{VaR}_{1-\alpha}) = \alpha \tag{2.1}$$

where V_0 is the portfolio value at the beginning of the time period, V the portfolio value at the end of the time period and $1 - \alpha$ the confidence level. Note that VaR is a measure of loss, hence the $-$VaR in (2.1).

2.2 Variance-Covariance Method

In this thesis the variance-covariance (or delta normal) method is used as a benchmark. The variance-covariance method assumes that the risk factors of the portfolio are all normally distributed. Before going in to the details of this approach it should be noted that when estimating VaR the risk factors influencing the value of the portfolio have to be identified. What these are depend on the assets that makes up the portfolio, they can for example be stock returns, exchange rates and interest rates. In this thesis portfolios of stocks are considered and the risk factors are the

stock returns. We let $\boldsymbol{r}$ be a vector with elements r_i [1] being the return for stock i among k stocks in the portfolio, and $\boldsymbol{w}$ be a vector with portfolio weights w_i where $\sum_{i=1}^{k} w_i = 1$

$$\boldsymbol{r} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{bmatrix} \quad , \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \end{bmatrix}$$

then the return for the portfolio is

$$r_p = \boldsymbol{w}^T \boldsymbol{r} \tag{2.2}$$

and the expected return of the portfolio is

$$\mu_p = \boldsymbol{w}^T \boldsymbol{\mu} \tag{2.3}$$

where $\boldsymbol{\mu}$ is the vector with elements μ_i being the expected return for asset i. Moreover, the variance of the portfolio is

$$\sigma_p^2 = \boldsymbol{w}^T \boldsymbol{C} \boldsymbol{w} \tag{2.4}$$

where $\boldsymbol{C}$ is the covariance matrix for the return vector $\boldsymbol{r}$

$$\boldsymbol{C} = \begin{bmatrix} \sigma_1^2 & \sigma_{1,2} & \dots & \sigma_{1,k} \\ \vdots & \ddots & & \\ \sigma_{k,1} & \dots & & \sigma_k^2 \end{bmatrix}.$$

With the assumption that the returns of the individual assets are normally distributed, also the total return of the portfolio r_p is normally distributed. Now, we can use the properties of the normal distribution to calculate VaR. With V_0 being the portfolio value at the beginning of the period, the VaR with $1 - \alpha$ confidence level for the portfolio is

$$\mathrm{VaR}_{1-\alpha} = V_0(z_\alpha \sigma_p - \mu_p) \tag{2.5}$$

where z_α is the z-score for the standard normal distribution, for example $z_\alpha = 1.645$

[1] Here each r_i represents a risk factor.

would give VaR with 95% confidence level.

In practice we can estimate VaR by using the sample counterparts. If we have a portfolio of k assets and n sample returns for each asset $r_{i,j}$, $i = 1, ..., k$, $j = 1, ..., n$ then we have the sample mean vector of returns

$$\bar{\boldsymbol{r}} = \begin{bmatrix} \bar{r}_1 \\ \bar{r}_2 \\ \vdots \\ \bar{r}_k \end{bmatrix} \quad \text{, where } \bar{r}_i = \frac{1}{n} \sum_{j=1}^{n} r_{i,j}$$

and the sample mean return for the portfolio

$$\hat{\mu}_p = \boldsymbol{w}^T \bar{\boldsymbol{r}}. \tag{2.6}$$

Furthermore the sample covariance matrix is

$$\boldsymbol{S} = \begin{bmatrix} s_{1,1} & s_{1,2} & \dots & s_{1,k} \\ \vdots & \ddots & & \\ s_{k,1} & \dots & & s_{k,k} \end{bmatrix} \quad \text{, where } s_{i,m} = \frac{1}{n-1} \sum_{j=1}^{n} (r_{i,j} - \bar{r}_i)(r_{m,j} - \bar{r}_m).$$

Now the portfolio variance can be estimated as

$$\hat{\sigma}_p^2 = \boldsymbol{w}^T \boldsymbol{S} \boldsymbol{w} \tag{2.7}$$

and the VaR as

$$\widehat{\text{VaR}}_{1-\alpha} = V_0(z_\alpha \hat{\sigma}_p - \hat{\mu}_p). \tag{2.8}$$

This presentation on how to estimate VaR is partly based on Jorion (2006) that gives a more in depth presentation of VaR for the interested reader.

2.3 Backtesting

A model to be used to estimate VaR needs to be validated to see if the predictions of the model is adequate. This can be done using backtesting, were statistical methods are used to compare VaR forecasts and actual returns for historical data (Jorion, 2006). In this thesis Kupiec's proportion of failures (POF) test by Kupiec (1995) (as cited in MathWorks, n.d.) will be used. For $\text{VaR}_{1-\alpha}$ the proportion of expected

failures (or exceedences) of VaR is α of the observations, following the notation of MathWorks we call this proportion p. Kupiec's POF test uses likelihood ratio to test if the probability of exceptions for the model differs from p (MathWorks, n.d.). We let N be the number of observations and x the number of exceedences for the VaR model, then the test statistic is

$$LR_{POF} = -2\log\left(\frac{(1-p)^{N-x}p^x}{(1-\frac{x}{N})^{N-x}(\frac{x}{N})^x}\right) \tag{2.9}$$

this test statistic is asymptotically chi-square distributed with 1 degree of freedom. If the test statistic exceeds the critical value the model is rejected by the test (MathWorks, n.d.).

CHAPTER 3

NEURAL NETWORKS

In this chapter a brief introduction to neural networks is given, so that the basics are covered before GANs are explained in chapter 4. To a large extent the notation follows that of Goodfellow, Bengio, and Courville (2016) which is good for a more in depth presentation of neural networks.

3.1 Structure of a Feed Forward Neural Network

A neural network can be used for classification problems, for example to classify borrowers as default or non-default, or regression problems such as predicting house prices. Figure 3.1 illustrates a feed forward neural network with three layers. The first layer is called the input layer and takes p features (independent variables) as inputs. The middle layer is called a hidden layer and consists of d units. The hidden layer takes a linear combination of the input layer and applies an activation function. Finally, the last layer is the output layer with k outputs.

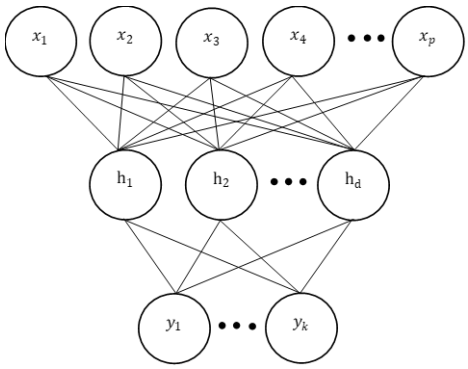


Figure 3.1: A feedforward neural network.

Using similar notation to that of Goodfellow et al. (2016) the hidden layer can be defined as

$$\boldsymbol{h} = r(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{c}) \tag{3.1}$$

where $\boldsymbol{h}$ is a d-dimensional vector representing the units in the hidden layer, r is the activation function, $\boldsymbol{x}$ the p-dimensional vector of inputs, $\boldsymbol{W}_1$ is a $p \times d$ matrix of weights between the input layer and the hidden layer and $\boldsymbol{c}$ is a d-dimensional vector of biases. The activation function r can for example be the rectified linear unit (ReLU)

$$r(z_i) = max(0, z_i) \tag{3.2}$$

were r is applied to each element z_i of $\boldsymbol{z} = \boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{c}$. Applying weights and biases to the above gives the output layer as

$$f(\boldsymbol{x}; \boldsymbol{W}_1, \boldsymbol{c}, \boldsymbol{W}_2, \boldsymbol{b}) = a(\boldsymbol{W}_2^T r(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{c}) + \boldsymbol{b}) \tag{3.3}$$

where $\boldsymbol{W}_2$ is a $d \times k$ matrix of weights between the hidden layer and the output layer, $\boldsymbol{b}$ is a k-dimensional vector of biases and a is the activation function for the output layer. As stated by Friedman, Hastie, and Tibshirani (2001) this function can be the identity function for regression type problems or softmax function for classification into k classes, were the softmax function is

$$a(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{k} e^{y_j}} \tag{3.4}$$

and where the a is applied to each element y_i of $\boldsymbol{y} = \boldsymbol{W}_2^T r(\boldsymbol{W}_1^T \boldsymbol{x} + \boldsymbol{c}) + \boldsymbol{b}$, that is the outputs before applying the activation function.

It should be noted that there is one hidden layer in the neural network illustrated here, but a deeper network with more layers is also possible. The above network is naturally extended by additional layers, weights, biases and activation functions.

3.2 Training of the Network

By minimizing a function called the cost function, or sometimes error function, the weights and biases of the network can be found. Friedman et al. (2001) suggests sum of squared error for regression and for classification either squared error or cross entropy. The cost functions used in this thesis is described in the next chapter that presents GANs. The cost function can be minimized by a step-wise procedure

called gradient descent. For each step the cost function is minimized by calculating the gradient of the function with respect to its parameters, and then updates the parameters by moving in the direction of the negative gradient. Formally, that is for a cost function $J(\boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$ the updated parameters $\boldsymbol{\theta'}$ are

$$\boldsymbol{\theta'} = \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{3.5}$$

where ϵ is the called the learning rate and controls the size of the update. As noted by Goodfellow et al. (2016) commonly the cost function is a sum over a loss function for each training sample, and with larger training data the training becomes computationally more expensive. A variation of the minimization process above is stochastic gradient descent that uses a minibatch of training examples for each update step, that is a sample from the training examples. Given a training example $\boldsymbol{x_i}$ and parameters $\boldsymbol{\theta}$ we have the prediction $f(\boldsymbol{x_i}; \boldsymbol{\theta})$ for the network, and we denote the loss for example $\boldsymbol{x_i}$ by $L(f(\boldsymbol{x_i}; \boldsymbol{\theta}), \boldsymbol{y^{(i)}})$ were $\boldsymbol{y^{(i)}}$ is the target. Now, using a minibatch of size m the gradient is instead estimated as

$$\hat{\boldsymbol{g}} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i}^{m} L(f(\boldsymbol{x_i}; \boldsymbol{\theta}), \boldsymbol{y^{(i)}}) \tag{3.6}$$

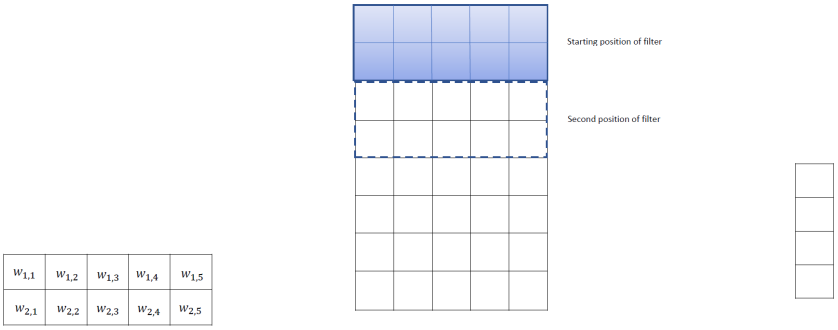
and for each step the parameters are updated according to

$$\boldsymbol{\theta'} = \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}. \tag{3.7}$$

Further extensions to stochastic gradient descent have been made, one such is the Adam optimizer by Kingma and Ba (2014) that has become a popular choice for deep learning. The Adam optimizer applies individual learning rates to the parameters, where the learning rates are adjusted by calculating estimates of the first and second order moments of the gradients. Adam takes four setup parameters stepsize α, exponential decay rate for first order moment estimate β_1, exponential decay rate for second order moment estimate β_2 and a small parameter ϵ used to avoid division by zero in the algorithm. Note here that the stepsize parameter α is not the effective stepsize (which is adaptive in the algorithm). Furthermore, Kingma and Ba show on a number of different applications that the Adam optimizer works well empirically.

3.3 Convolutional Layers

In section 3.1 a neural network where the layers are fully connected is shown (also called dense layers), that is all the neurons in a layer are connected to the neurons in the next layer. In addition to this type of layer another type of layer called convolutional layer will be used, more precisely what is called a one dimensional convolutional layer. An example of a one dimensional convolutional layer is illustrated in Figure 3.2. The convolutional layer takes the input data as a matrix and another matrix called the filter (Figure 3.2 (a)), that contains the weights, is slid over as illustrated in Figure 3.2 (b).



(a) Filter of size 2 (b) Convolution with stride 2 (c) Output

Figure 3.2: One dimensional convolutional operation with stride 2.

For each step we take the sum of the element wise product of the filter and the corresponding subset of the input data (blue region in Figure 3.2 (b)). In addition to the weights we can also include a bias for the filter. Note that in this example the size of the filter is 2 as it has 2 rows.

From the convolution we get one scalar output per step, the convolution results in an output illustrated in Figure 3.2 (c). The number of rows that the filter moves down in each step is the so called stride, which is 2 in Figure 3.2 (b). Furthermore we can apply more than one filter, the output of the convolutional operation will then have depth > 1 corresponding to the number of filters. More precisely the input has dimensions $rows_{input} \times width_{input} \times depth_{input}$ and the output $rows_{output} \times width_{output} \times depth_{output}$. For example if the input is a RGB image $depth_{input} = 3$, we have three channels; reed, green and blue.

We can also add rows of zeros in the beginning and end of the input data, which is called padding. This is used in this thesis to control the size of the output of a convolutional layer, more precisely whats called same padding was used. Same padding for 1D convolution means adding zeros such that with a stride of 1, $rows_{input}$

is the same as $rows_{output}$ (when convolution is preformed along the row dimension). More generally, the dimension of $rows_{output}$ will be $\lceil \frac{rows_{input}}{stride} \rceil$ if we are convolving along the row dimension (where $\lceil \ \rceil$ is the ceiling operator). By using a stride of 2 the number of rows will be halved if the input has an even number of rows.

3.4 Batch Normalization

Ioffe and Szegedy (2015) proposed a technique called batch normalization to accelerate training of deep neural networks. Batch normalization aims at handling the problem with changing distributions of the inputs to layers in the network. This is done by applying normalization to each activation before the next layer takes the normalized ones as inputs. Consider a layer in a network with p activations (outputs). Lets call these activations $x^{(1)}, x^{(2)}, ..., x^{(p)}$ then each of these are normalized to have a mean of 0 and variance of 1. This is done by estimating mean and variance from the data of a minibatch of size m, for each of the $x^{(k)}$ we have m activations per minibatch. Now, leaving out the superscript k for simplicity, we have activations $x_1, x_2, ..., x_m$ for a minibatch. For the minibatch the mean is calculated as

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i \tag{3.8}$$

and the variance as

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \tag{3.9}$$

and then normalized values are calculated as

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{3.10}$$

where ϵ is a constant for numerical stability. However, as noted by Ioffe and Szegedy the normalized input values can alter what the layer can represent, therefore they for each $x^{(k)}$ include two trainable parameters scale $\gamma^{(k)}$ and shift β^k. Again leaving out the superscript the scaled and shifted normalized value is

$$y_i = \gamma \hat{x}_i + \beta. \tag{3.11}$$

This step might seem odd at first as we can adjust back the mean by β. However, as noted by Goodfellow et al. (2016) before normalization the mean was decided by a complicated interaction of parameters from the previous layers instead of only β.

CHAPTER 4

GENERATIVE ADVERSARIAL NETWORKS

In this chapter the algorithm used to simulate future stock returns, generative adversarial network (GAN), is explained. In the first section the overall idea of GANs is presented, and thereafter objective functions and possible improvements are discussed.

4.1 Introduction to GANs

Generative Adversarial Networks (GANs) is a type of a generative model that was introduced by Goodfellow et al. (2014). GANs are made up of two main components, a generator function G and a discriminator function D. The discriminator's goal is to identify whether data is coming from the training set or from the generator by classifying it as fake or real. The generator however tries to fool the discriminator by finding the distribution of the training data. The generator does this by trying to minimize a cost function $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ which depends on both the parameters of the discriminator $\boldsymbol{\theta}^{(D)}$ and its own parameters $\boldsymbol{\theta}^{(G)}$, but it can only control its own parameters. Similarly the discriminator tries to distinguish real from fake data by minimizing a cost function $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ by only controlling its own parameters $\boldsymbol{\theta}^{(D)}$. To generate the new data the generator takes a latent variable $\boldsymbol{z}$ as input, typically represented by some simple probability distribution, and produces an output $G(\boldsymbol{z})$ from the probability distribution of the model. Further, a real data sample is denoted by $\boldsymbol{x}$. The generator G and the discriminator D are typically represented by neural networks, details on the architectures used for these are given in section 5.2. An illustration of a GAN is shown in Figure 4.1.

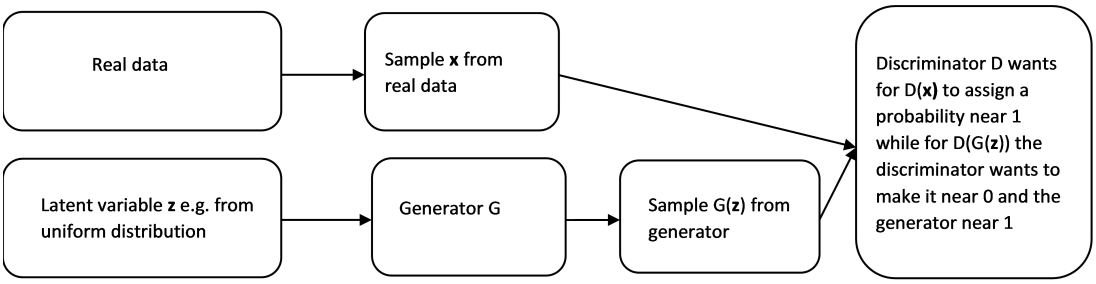


Figure 4.1: Illustration of a generative adversarial network.

4.2 Objective Functions

We will now go into some more detail on the objective functions. Note that, as the term cost function would be limited to a function that should be minimized, and we also can express an optimization problem in terms of maximization we use the more general term objective function. First, starting with the objective function for the discriminator $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$. The original presentation of GANs by Goodfellow et al. (2014) maximizes the following objective function for the discriminator (corresponding to minimizing $-J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ if expressed in terms of cost)

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = \mathbb{E}_{\mathbf{x}\sim p_{data}} \log D(\mathbf{x}) + \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))). \tag{4.1}$$

Note that that the discriminator is trained using two sets of data, one is real data and the other generated from the generator. The real data is coded as 1 and the generated data as 0. From (4.1) we can see that the discriminator maximizes its objective function by giving probabilities as close to 1 as possible for the real data and as close to 0 as possible for the generated data. For the generator Goodfellow et al. (2014) minimizes the function

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = \mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z}))). \tag{4.2}$$

Here the generator will minimize its objective function by making the discriminator assign probabilities as close as possible to 1, that is $D(G(\mathbf{z}))$ close to 1, trying to fool the discriminator.

4.3 Wassersterstein GAN

A popular variation to improve stability, making the model less sensitive to architectural choices is the Wasserstein GAN (WGAN) by Arjovsky, Chintala, and Bottou

(2017). Their approach differs in terms of choice of objective function. What we previously called discriminator they call critic, as it is now not bounded to give a value between 0 and 1. The objective function is derived from what is called the Earth-mover distance (or Wasserstein-1). Furthermore, for their objective function the critic should be a 1-Lipschitz continuous function. Under the Lipschitz constraint the following function is minimized (or maximizing $-J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$)

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = \mathbb{E}_{\mathbf{z}}\, D(G(\mathbf{z})) - \mathbb{E}_{\mathbf{x}\sim p_{data}}\, D(\mathbf{x}). \tag{4.3}$$

Furthermore, for the generator we simply minimize

$$J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = -\,\mathbb{E}_{\mathbf{z}}\, D(G(\mathbf{z})). \tag{4.4}$$

To enforce the Lipschitz constraint, they use weight clipping on the critic. This means that the weights of the critic is clipped so that they are within some range $[-c, c]$. However, Arjovsky et al. notes that there are problems with using weight clipping and the choice of the parameter c. To improve on the WGAN Gulrajani, Ahmed, Arjovsky, Dumoulin, and Courville (2017) suggest to instead use gradient penalty to enforce the constraint (WGAN-GP). Gulrajani et al. state that:

> "A differentiable function is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere"

They use this fact and introduce a penalty term, so the critics objective function becomes

$$J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)}) = \mathbb{E}_{\mathbf{z}}\, D(G(\mathbf{z})) - \mathbb{E}_{\mathbf{x}\sim p_{data}}\, D(\mathbf{x}) + \lambda\, \mathbb{E}_{\hat{\mathbf{x}}}(\nabla_{\hat{\mathbf{x}}}||D(\hat{\mathbf{x}})||_2 - 1)^2 \tag{4.5}$$

where the penalty term $\lambda\, \mathbb{E}_{\hat{\mathbf{x}}}(\nabla_{\hat{\mathbf{x}}}||D(\hat{\mathbf{x}})||_2 - 1)^2$ takes randomly weighted averages of real and generated data $\hat{\mathbf{x}}$. The objective function is penalized when the gradient deviates from 1, rather than only when it is above. Gulrajani et al. empirical results shows that this does not constrain the critic much. The penalty coefficient λ, they set to 10 with good empirical results on different architectures and image data sets.

CHAPTER 5

DATA AND METHODOLOGY

This chapter describes how the suitability of using GANs for VaR estimation was evaluated. Training GANs can be difficult and there are many possible setups, after some initial experimentation on the data a number of models were considered for VaR estimation and to be backtested. This chapter describes these models and how they were tested. First, the data and software tools used are discussed. Second, the different types of GAN models considered for VaR estimation are presented. Finally, the training and backtesting procedure for the GANs and benchmark model is described.

5.1 Data Preparation and Software

Programming was done using Python 3 and the machine learning libraries TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2015). The implementation of the WGAN-GP was based on the Keras code example by Nain (2020).

To test the suitability of GANs for VaR estimation, stock price data was downloaded from Yahoo! Finance (*Yahoo! Finance*, 2020). A portfolio consisting of five stocks from the Nasdaq OMX Stockholm large cap list with good data availability was selected. The five stocks selected were: Hennes & Mauritz B, Elekta B, Tele2 B, SEB A and Investor B. For each stock the daily adjusted close price was used to calculate the daily returns according to

$$r_t = \frac{s_t - s_{t-1}}{s_{t-1}} \tag{5.1}$$

where, s_t is the stock price at time t and s_{t-1} the stock price at time $t-1$. Moreover, the portfolio considered is an equally weighted portfolio of long positions, that is a

weight of 0.2 for each of the five stocks.

5.2 GANs for Value at Risk

In this thesis VaR was modeled by training the generator with the aim of learning the distribution of the returns for a portfolio of stocks. The generator can be used to run simulations of future portfolio returns, and the $a\%$ VaR estimated as the ath percentile times -1 (note VaR is in terms of loss). The VaR considered was the 95% daily VaR. This was done to get a fairly large number of observations and expected number of failures for backtesting, and thereby a more reliable result than if a VaR for a longer time period and higher confidence level was used. Furthermore, for each estimate of VaR using GANs, 1000 simulations from the generator were used.

Two different types of models were considered. 1) A conditional model, where the generator G learns a distribution conditional on previous returns. 2) Unconditonal models, where the generator learns the unconditional distribution. In the first case the generator takes previous returns as inputs in addition to the latent input. Models of both these types are backtested using Kupiec's test described in section 2.3. As a benchmark also a traditional variance-covariance model was considered. The types of GANs considered for VaR estimation will now be discussed.

5.2.1 Conditional Model

Mariani et al. (2019) use a conditional GAN for portfolio optimization. A similar GAN is used here for VaR estimation. In their case the generator G takes b days of previous prices in addition to the latent variable. The aim is for G to learn the distribution of the portfolios prices the following f days. The data is represented by matrices $\mathbf{M}$ containing a total of w days of prices, split in two parts, the matrix $\mathbf{M_b}$ containing prices for the b previous days and $\mathbf{M_f}$ that contains the prices for the f following days. So the generator generates simulations $\hat{\mathbf{M}}_\mathbf{f}$ taking previous prices $\mathbf{M_b}$ and latent input $\mathbf{z}$

$$\hat{\mathbf{M}}_\mathbf{f} = G(\mathbf{M_b}, \mathbf{z}). \tag{5.2}$$

For the generator Mariani et al. split it into two parts. First, in what they call conditioning, they process the conditioning data $\mathbf{M_b}$ by a number of 1D convolutional layers and a dense layer. This is then concatenated with the latent vector (and a vector of analysis values they calculate). Second, the output from the above is taken as inputs to what they call the simulator, to generate the simulations $\hat{\mathbf{M}}_\mathbf{f}$.

The discriminator takes as input either real data $\mathbf{M}$ or fake data, that is the concatenation of $\mathbf{M_b}$ and $\hat{\mathbf{M}}_\mathbf{f}$. A similar approach is taken here, were the data is split into $w \times k$ matrices $\mathbf{M}$ of stock returns. Each matrix containing w days (rows) and k stocks (columns). A backwards window of $b = 40$ is used for conditioning, a forwards window of $f = 1$ as one day VaR is estimated and a portfolio of $k = 5$ stocks. The details of the generator's architecture for the conditional model is shown in Table 5.1.

Table 5.1: Generator conditional model.

Layer Number	Type of Layer	Output Shape	Description
1	Input layer	40×5	Takes conditional matrix $\mathbf{M_b}$
2	1D convolutional layer	20×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
3	Batch normalization	20×10	
4	1D convolutional layer	10×10	Number of filters = 10, stride = 2, filter size = 5, padding same, activation ReLU
5	Batch normalization	10×10	
6	1D convolutional layer	5×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
7	Batch normalization	5×10	
8	1D convolutional layer	3×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
9	Batch normalization	3×10	
10	Flatten	30	Flattening the output
11	Dense layer	5	Fully connected layer, activation ReLU
12	Batch normalization	5	
13	Latent Input	10	Latent input vector $\mathbf{z}$ of length 10
14	Concatenation	15	concatenation of 12 and Latent input vector $\mathbf{z}$ (13)
15	Dense layer	20	Fully connected layer, activation ReLU
16	Batch normalization	20	
17	Reshape	4×5	Reshaping to 4×5
18	1D convolutional layer	2×5	Number of filters = 5, stride =2, filter size = 5, padding same, activation ReLU
19	Batch normalization	2×5	
20	1D convolutional layer	1×5	Number of filters = 5, stride =2, filter size = 5, padding same

The generator takes the conditioning input $\mathbf{M_b}$ followed by four 1D convolutional layers with batch normalization, where convolution is done over the rows (time dimension). This output is then flattened and followed by a fully connected layer and batch normalization. The generator then takes the latent input and concatenates it with the previous layers input. This is followed by a fully connected layer and batch normalization before it is reshaped and followed by two 1D convolutional layers (with batch normalization after the first) to shrink it down to the desired output shape. The output of the generator $\hat{\mathbf{M}}_\mathbf{f}$ is a vector with simulated returns for the five stocks in the portfolio. The discriminator's architecture is shown Table 5.2.

Table 5.2: Discriminator conditional model.

Layer Number	Type of Layer	Output Shape	Description
1	Input layer	41×5	Takes a matrix $\mathbf{M}$ or the concatenation of $\mathbf{M_b}$ and $\hat{\mathbf{M}}_\mathbf{f}$
2	1D convolutional layer	21×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation leaky ReLU
3	1D convolutional layer	11×20	Number of filters = 20, stride =2, filter size = 5, padding same, activation leaky ReLU
4	1D convolutional layer	6×40	Number of filters = 40, stride =2, filter size = 5, padding same, activation leaky ReLU
5	1D convolutional layer	3×80	Number of filters = 80, stride =2, filter size = 5, padding same, activation leaky ReLU
6	1D convolutional layer	2×160	Number of filters = 160, stride =2, filter size = 5, padding same, activation leaky ReLU
7	1D Flatten	320	Flattening the output
8	Dense layer	1	Fully connected layer

The discriminator takes a 41×5 matrix, either real data $\mathbf{M}$ or fake data, that is the concatenation of $\mathbf{M_b}$ and $\hat{\mathbf{M}}_\mathbf{f}$. This is processed by five 1D convolutional layers, flattened and finally there is a fully connected output layer. The discriminator gives a critic score, where it wants to assign a larger value if it believes the data comes from the true distribution, and a smaller if it believes it comes from the distribution learned by the generator.

5.2.2 Unconditional Model

The second type is an unconditional model, where the generator only takes the latent input $\mathbf{z}$ for a simulation $\hat{\mathbf{M}}_\mathbf{f}$

$$\hat{\mathbf{M}}_\mathbf{f} = G(\mathbf{z}). \tag{5.3}$$

As we do not condition on previous stock returns, an observation $\mathbf{M}$ is simply a 5-dimensional vector of returns. The discriminator takes either true returns $\mathbf{M}$ or fake returns from the generator $\hat{\mathbf{M}}_\mathbf{f}$. The architecture of the generator for the unconditional model is shown in Table 5.3.

Table 5.3: Generator unconditional model.

Layer Number	Type of Layer	Output Shape	Description
1	Input layer	20	Takes a latent input vector $\mathbf{z}$ of size 20
2	Dense layer	160	Fully connected layer, activation ReLU
3	Batch normalization	160	
4	Reshape	32×5	Reshaping to 32×5
5	1D convolutional layer	16×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
6	Batch normalization	16×10	
7	1D convolutional layer	8×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
8	Batch normalization	8×10	
9	1D convolutional layer	4×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
10	Batch normalization	4×10	
11	1D convolutional layer	2×10	Number of filters = 10, stride =2, filter size = 5, padding same, activation ReLU
12	Batch normalization	2×10	
13	1D convolutional layer	1×5	Number of filters = 5, stride =2, filter size = 5, padding same

The input layer of the generator takes the latent input $\mathbf{z}$ followed by a dense layer and batch normalization. This is then reshaped and followed by convolutional layers with batch normalization. The generator generates a simulation of one day of returns for the five stocks. The structure of the discriminator for the unconditional model is built up by a number of fully connected layers as shown in Table 5.4.

Table 5.4: Discriminator unconditional model.

Layer Number	Type of Layer	Output Shape	Description
1	Input layer	1×5	Takes real data $\mathbf{M}$ or fake data $\hat{\mathbf{M}}_\mathbf{f}$
2	Dense layer	1×100	Fully connected layer, activation Leaky ReLU
3	Dense layer	1×50	Fully connected layer, activation Leaky ReLU
4	Dense layer	1×25	Fully connected layer, activation Leaky ReLU
5	Dense layer	1×1	Fully connected layer

5.3 Training and Backtesting of Models

To test the suitability of GANs as a way of estimating VaR, five VaR models based on GANs are tested as well as a benchmark model using the traditional variance-covariance approach described in section 2.2. All the models were backtested on a data set of daily returns for the period 2010-12-22 to 2019-12-30. When performing Kupiec's POF test a significance level of 5% was used, meaning a model with a p-value lower than 5% is rejected by the test.

GAN models of both types are trained on data of returns from dates prior the backtesting period. In addition to this training the conditional models are updated during backtesting by additional training on recent dates to the day VaR is estimated for. This is similar to the approach used by Fiechtner (2019), that updates the VaR estimates for backtesting every 10 days by training for 100 additional epochs. The models considered are: 1) a conditional model as described in section 5.2.1, trained for 2000 epochs on a training data set of daily return matrices $\mathbf{M}$, where returns are from the period 2000-01-04 to 2009-12-18. The return matrix $\mathbf{M}$ is 41×5 and the first 40 rows (days) correspond to the conditioning matrix $\mathbf{M_B}$ (40 days of returns). VaR is estimated for the backtesting data set by running 1000 simulations according to 5.2 were $\mathbf{M_b}$ is the matrix with returns for the previous 40 days. 2) An unconditional model, first pretrained for 2000 epochs using returns from the period 2000-02-29 to 2009-12-18. When calculating the VaR estimates for the backtesting data the pretrained model was trained for 100 additional epochs every five days using the previous 252 days of returns. Here a trade-off between the computational

cost and the ability of the model to adjust to changes in the market conditions was made. 3) An unconditional model, pretrained for 1000 epochs on returns from the period 2005-12-22 to 2009-12-18. As in model 2 a new VaR estimate was calculated every five days using the 252 most recent days of returns, but additional training was done for 200 epochs from the most recent parameters of the model rather than from the parameters of the original pretrained model. 4) As model 2, but training for 20 additional epochs and updating every second day. 5) As model 4, but updating every day and using a larger value for the step size parameter. 6) A model using the traditional variance-covariance approach where each estimate is based on the 252 previous days of returns.

All the GAN models are of the WGAN-GP type, and as done by the paper introducing WGAN-GP (Gulrajani et al., 2017), the discriminator is trained for five steps for every step of the generator. For all GAN models a batch size of 36 and the Adam optimizer were used for training. The step size parameter α was set to 2×10^{-6} for all GAN models except for model 5 where it was set to 2×10^{-5}. As this model is updated every day, a larger value was allowed due to computational cost. The penalty term in the objective function of the discriminator (4.5) was set to 10 and in the Adam optimizer β_1 was set to 0.5 and β_2 to 0.9 as done by Gulrajani et al. (2017). Furthermore for all GAN models the data was standardized, this was done as to make the pretraining data have a mean of 0 and standard deviation of 1. That is for each stock the returns are standardized as

$$r_t^s = \frac{r_t - \mu}{\sigma} \tag{5.4}$$

where r_t is the unstandardized return, μ the mean in pretraining data and σ the standard deviation in the pretraining data. Furthermore, for the latent variable $\boldsymbol{z}$ of the GAN models the standard multivariate normal distribution was used.

CHAPTER 6

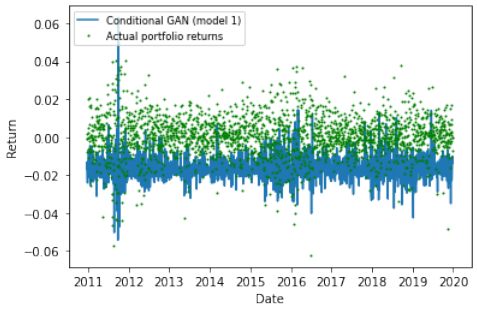
RESULTS AND ANALYSIS

In this chapter the results from the empirical analysis on the backtesting data is presented and discussed. First the considered models are compared against the actual returns visually, thereafter the results of performing Kupiec's POF test are discussed.

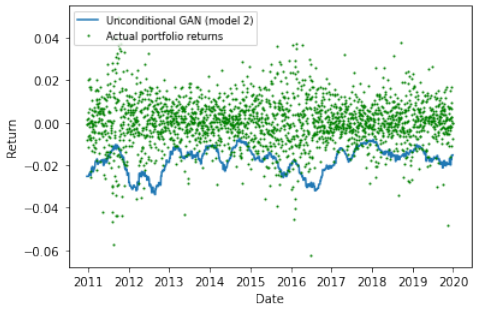
6.1 Plots of Actual Returns Versus VaR Models

To begin, we look at plots of the actual returns over the test period (2010-12-22 to 2019-12-30) versus the $-$VaR estimates of the models considered in Figure 6.1. As the models are for 95% VaR, we expect close to 5% of the actual returns to be below the line indicating the $-$VaR estimates for a good model. For all the models it is at least clear that most of the actual returns are above the estimates given by the models. But by looking at Figure 6.1a, it shows that model 1 (conditional GAN) differs allot from the other five models. The hope for this model was that it would capture changes in market conditions by conditioning the model on recent returns for the underlying assets, modelling the conditional distribution of returns. The plot shows a very jagged line for the VaR estimates and it seems unrealistic that the large changes over short time periods would simply reflect changes in the market. The other models are not directly conditioned on previous returns as model 1, however they are through the training process where the models are trained on recent returns. The other GAN models (2-5) show similar changes in the VaR estimates over time. The fact that model 5 is updated frequently is evident in Figure 6.1e from the jagged line for the $-$VaR estimate. But, overall the different training setups of the models don't seem to effect the overall pattern of the VaR

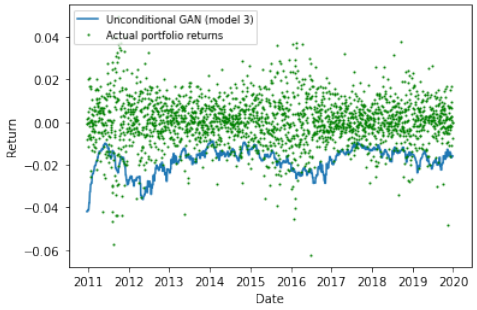
estimates much. Furthermore, the benchmark (model 6) in Figure 6.1f shows quite similar estimates to the unconditional GAN models.



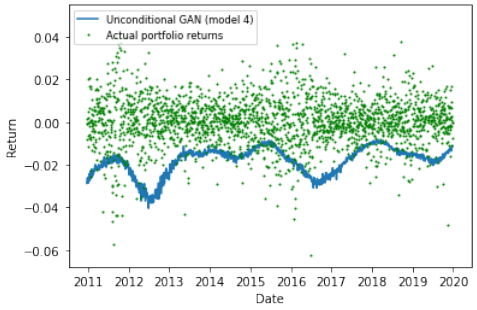
(a) Model 1 (conditional GAN) and actual returns.



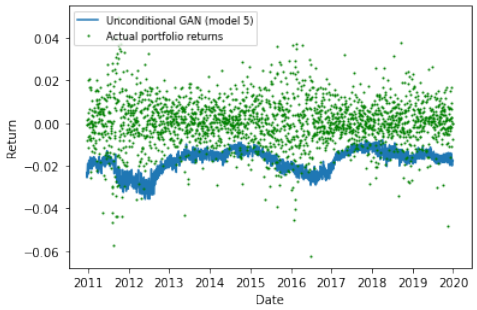
(b) Model 2 and actual returns.



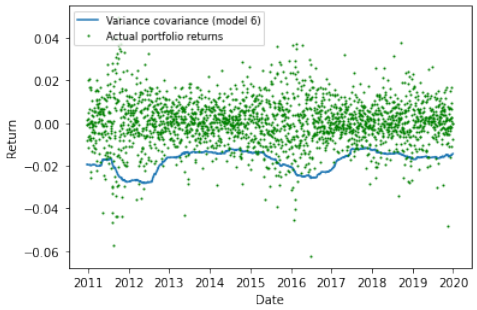
(c) Model 3 and actual returns.



(d) Model 4 and actual returns.



(e) Model 5 and actual returns.



(f) Model 6 (variance-covariance model) and actual returns.

Figure 6.1: 95% one day −VaR estimates and actual returns for the backtesting period.

6.2 Kupiec's POF Test

The results of the backtesting of the six models using Kupiec's POF test are shown in Table 6.1. All the models show a higher proportion of failures on the backtesting data than the expected 5%. That is all models underestimates VaR for the backtesting period. The model closest to 5% of failures is the variance-covariance model. Out

of the GAN models the model pretrained on a shorter time period (model 3) is the closest, with 6.14% of failures. However, all models except the variance-covariance model (model 6) are rejected at the 5% level by Kupiec's POF test.

Table 6.1: Backtesting results for VaR models.

	Number of Failures	Proportion Failures	P-value for Kupiec's Test
Model 1	173	0.0764	7.9e-08
Model 2	149	0.0658	0.00097
Model 3	137	0.0605	0.02609
Model 4	144	0.0636	0.00430
Model 5	139	0.0614	0.01613
Model 6	128	0.0565	0.16174

Some VaR models might perform better during some periods and market conditions. Table 6.2 and Table 6.3 show how the models performed during the first half and second half of the period separately. For the first half, model 4, 5 and 6 are not rejected by Kupiec's test, while for the second half only the conditional GAN model (model 1) is rejected. The GAN models not rejected for the first period are those that are updated more frequently. Model 3 performs much better in the second period than the first and is doing almost as well as the benchmark model. However, non of the GAN models are better than the others in both periods and only the benchmark model can not be rejected overall. Further, with a larger number of observations the rejection region for Kupiec's POF test gets smaller and its therefore not surprising that some models are not rejected in neither period separately, but are overall. For example model 5 which can not be rejected in neither period at the 5% significance level.

Table 6.2: Backtesting results for the first period (2010-12-22 to 2015-07-02).

	Number of Failures	Proportion Failures	P-value for Kupiec's Test
Model 1	82	0.0724	0.00113
Model 2	79	0.0698	0.00385
Model 3	78	0.0689	0.00565
Model 4	68	0.0600	0.13130
Model 5	69	0.0610	0.06095
Model 6	70	0.0618	0.07754

Table 6.3: Backtesting results for the second period (2015-07-03 to 2019-12-30).

	Number of Failures	Proportion Failures	P-value for Kupiec's Test
Model 1	91	0.0804	1.5e-05
Model 2	70	0.0618	0.07754
Model 3	59	0.0521	0.74507
Model 4	76	0.0671	0.06714
Model 5	70	0.0618	0.07754
Model 6	58	0.0512	0.84916

There is a trade-off for the unconditional GAN models between training the models on enough data so they generalize well and at the same time having the models reflect current market conditions. Therefore how we train the models on the previous data is important.

CHAPTER 7

CONCLUSION

Using GANs as a way of estimating VaR has been studied. Both a model where the generator is modelling a conditional distribution of returns as well as unconditional models were considered. It was shown that the conditional model is very sensitive to the conditioning data and the backtesting shows that its not a suitable model for VaR. The unconditional models showed more promise, the development of the VaR estimates over time are similar to the ones of the variance-covariance model. However, looking at the results of Kupiec's POF test over a total test period of 2164 observations, shows that only the variance-covariance model is not rejected. A challenge with the unconditional models is to capture current market conditions which might not be reflected in data far back in time. The results do not show that the more complicated GAN models are preferred as the variance-covariance model is simpler and also shows better performance in the backtesting. The more traditional approach therefore seems like a better choice, but GANs are fairly new and especially for financial applications. GANs can be difficult to train and problems such as instability can be evident. Recent improvements such as the development of WGAN-GP has been developed to improve training. WGAN-GP was applied in this thesis for stability, still in future research other architectures of the GANs could be interesting to consider. For example, a different architecture of the generator of the conditional model that would result in more realistic changes in the VaR estimates over time.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Chollet, F., et al. (2015). *Keras.* `https://keras.io`.

Danielsson, J., Hartmann, P., & de Vries, C. (1998). The cost of conservatism. *Risk*, *11*(1), 101–103.

Fiechtner, L.-B. (2019). *Risk management with generative adversarial networks* (Unpublished master's thesis). University of Oxford.

Friedman, J., Hastie, T., & Tibshirani, R. (2001). *The elements of statistical learning* (2nd ed.). New York: Springer Series in Statistics.

Fu, R., Chen, J., Zeng, S., Zhuang, Y., & Sudjianto, A. (2019). Time series simulation by conditional generative adversarial net. *arXiv preprint arXiv:1904.11419*.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. (`http://www.deeplearningbook.org`)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems,* (pp. 2672–2680).

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. In *Advances in neural information processing systems* (pp. 5767–5777).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hull, J., & White, A. (1998). Incorporating volatility updating into the historical simulation method for value-at-risk. *Journal of risk*, *1*(1), 5–19.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jorion, P. (2006). *Value at risk: The new benchmark for managing financial risk* (3rd ed.). New York: McGraw-Hill.

Khan, S. (2019). *Calculate value-at-risk using wasserstein generative adversarial networks (wgan-gp) for risk management system.* Retrieved from `https://chatbotslife.com/calculate-value-at-risk-using-wasserstein-generative-adversarial-networks-wgan-gp-for-risk-2b1d320fde59` (accessed 2020-11-18)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Linsmeier, T. J., & Pearson, N. D. (1996). *Risk measurement: An introduction to value at risk* (Tech. Rep.). University of Illinois at Urban-Champaign.

Longin, F. M. (2000). From value at risk to stress testing: The extreme value approach. *Journal of Banking & Finance*, *24*(7), 1097–1130.

Mariani, G., Zhu, Y., Li, J., Scheidegger, F., Istrate, R., Bekas, C., & Malossi, A. C. I. (2019). Pagan: Portfolio analysis with generative adversarial networks. *arXiv preprint arXiv:1909.10578*.

Marimoutou, V., Raggad, B., & Trabelsi, A. (2009). Extreme value theory and value at risk: application to oil market. *Energy Economics*, *31*(4), 519–530.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91.

MathWorks. (n.d.). *Overview of var backtesting.* Retrieved from `https://se.mathworks.com/help/risk/overview-of-var-backtesting.html` (accessed 2020-12-28)

Nain, A. K. (2020). *Wgan-gp overriding model.train_step.* https://keras.io/examples/generative/wgan_gp/. (accessed 2021-01-18)

Saita, F. (2007). *Value at risk and bank capital management* (1st ed.). Amsterdam: Elsevier.

Shah, H. (2018). *Using bidirectional generative adversarial networks to estimate value-at-risk for market risk management.* Retrieved from https://towardsdatascience.com/using-bidirectional-generative-adversarial-networks-to-estimate-value-at-risk-for-market-risk-c3dffbbde8dd (accessed 2020-11-18)

Yahoo! finance. (2020). https://finance.yahoo.com/.