UNIVERSITY OF TURKU

# ABSTRACT

| | |
|---|---|
| | Bachelor's thesis |
| x | Master's thesis |
| | Licentiate's thesis |
| | Doctoral dissertation |

| | | | |
|---|---|---|---|
| Subject | Quantitative Methods in Management | Date | 9.6.2020 |
| Author | Akseli Leino | Number of pages | 58 |
| Title | Using generative adversarial network as a value-at-risk estimator | | |
| Supervisors | Prof. Luis Alvarez Esteban, Ph.D. Mika Hannula | | |


Abstract

Value-at-risk (VaR) estimation is a critical task for modern financial institution. Most methods to estimate VaR rely on classical statistical methods. They produce reliable estimates but there is demand for ever more accurate estimates. Recently there has been major breakthroughs for machine learning models in other fields. This has led to increasing interest in applying machine learning for financial applications. This thesis applies new data-driven machine learning method, generative adversarial network (GAN), for (VaR) estimation.

GAN was proposed for fake image generation. Since then it has found applications in multiple domains, such as finance. Estimating the true underlying distribution of financial time series is notoriously difficult task. GAN doesn't explicitly estimate the underlying distribution but tries to generate new samples from the distribution. This thesis applies a basic GAN model to simulate stock market returns and then estimate the VaR from these. The experiments are conducted on S&P500-index.

The GAN model is compared to a simple historical simulation baseline. In the experiments it becomes evident that the GAN model lacks robustness and responds poorly to changes in market. The GAN is unable to fully capture the statistical properties of stock market returns. It can replicate a little of the excess kurtosis present in stock market returns and some of the volatility clustering. The results show that the GAN model has tendency to estimate the VaR between a fairly narrow range. This is in contrast to historical simulation, which can respond to changes in the stock market.

Machine learning models, especially neural networks like GANs, present challenges to financial practitioners. Although they provide sometimes more accurate estimates than traditional methods, they lack transparency. GANs have shown promise in the literature but suffer from being unstable to train. It is difficult to guess will a trained GAN work as it is meant to work. Regardless of these shortcomings, it is worthwhile to study GANs and other neural networks in finance. They have performed exceptionally in other fields. Researchers must try to open the black-box nature of the models. Interpretability of the models will allow their use in the financial industry. This thesis shows that more research is needed to provide robust estimates that can be relied on.



| | |
|---|---|
| Key words | Value-at-risk, generative adversarial network, neural network |

TURUN YLIOPISTO

# TIIVISTELMÄ

| | |
|---|---|
| | Kandidaatintutkielma |
| x | Pro gradu -tutkielma |
| | Lisensiaatintutkielma |
| | Väitöskirja |

| | | | |
|---|---|---|---|
| Oppiaine | Taloustieteen kvantitatiiviset menetelmät | Päivämäärä | 9.6.2020 |
| Tekijä | Akseli Leino | Sivumäärä | 58 |
| Otsikko | Generatiivisen kilpailevan verkon käyttö value-at-risk arvon estimoinnissa | | |
| Ohjaajat | Prof. Luis Alvarez Esteban, KTT Mika Hannula | | |


Tiivistelmä

Value-at-risk (VaR) arvon estimointi on kriittinen tehtävä modernille finanssialan instituutiolle. Suuri osa VaR:n estimointimenetelmistä on perinteisiä tilastotieteellisiä menetelmiä. Nämä menetelmät tuottavat luotettavia estimaatteja, mutta tarkemmille menetelmille on aina tarvetta. Viime vuosina koneoppiminen on ottanut harppauksia muilla aloilla. Tämä on johtanut lisääntyneeseen kiinnostukseen soveltaa koneoppimista rahoitusalan sovelluksiin. Tässä tutkielmassa sovelletaan uutta dataan pohjautuvaa koneoppimismallia, nimeltään generatiivinen kilpaileva verkko (GAN), VaR arvon estimointiin.

GAN mallia sovellettiin ensin kuvien luontiin. Sen jälkeen mallia on sovellettu monilla aloilla mm. rahoituksessa. Rahoituksen aikasarjan allaolevan todellisen jakauman estimointi on äärimmäisen haastava tehtävä. GAN ei suoraan estimoi allaolevaa jakaumaa vaan yrittää generoida uusia otoksia jakaumasta. Tämä tutkielma käyttää normaalia GAN mallia simuloimaan osakemarkkinatuottoja ja estimoi näistä VaR:n. Testeissä käytetään S&P500-indeksiä.

GAN mallia verrataan yksinkertaiseen historialliseen simulaatioon. Testeistä käy ilmi, että GAN malli ei ole robusti ja reagoi heikosti muutoksiin markkinoilla. GAN malli ei pysty täysin replikoimaan osakemarkkinatuottojen tilastollisia ominaisuuksia. Malli replikoi vain osan tuottojen huipukkuudesta ja volatiliteetin klusteroinnista. Tulokset osoittavat, että GAN mallilla on taipuvuus estimoida VaR arvo hyvin kapealle välille. Tämä on päinvastoin kuin historiallinen simulaatio, joka reagoi osakemarkkinan muutoksin.

Koneoppimismallit, erityisesti neuroverkot kuten GAN, aiheuttavat haasteita niiden soveltajille rahoitusalalla. Vaikka ne tuottavat joskus tarkempia estimaatteja kuin perinteiset menetelmät, ne eivät ole läpinäkyviä. GAN mallit ovat saavuttaneet lupaavia tuloksia tutkimuskirjallisuudessa, mutta kärsivät epästabiilista koulutuksesta. On vaikeaa tietää koulutetun GAN mallin toimivuutta. Puutteista huolimatta on tärkeää tutkia GAN mallien ja muiden neuroverkkojen käyttöä rahoituksessa. Nämä mallit ovat toimineet erityisen hyvin toisilla aloilla. Tutkijoiden on yritettävä avata mallien läpinäkymätöntä luonnetta. Mallien tulkittavuus antaa mahdollisuuden soveltaa niitä rahoitusalalla. Tässä tutkielmassa osoitetaan, että lisätutkimusta tarvitaan, jotta GAN mallien voi luottaa tuottavan hyviä estimaatteja.



| | |
|---|---|
| Avainsanat | Value-at-risk, generatiivinen kilpaileva verkko, neuroverkko |

UNIVERSITY OF TURKU
Turku School of Economics

# USING GENERATIVE ADVERSARIAL NETWORK AS A VALUE-AT-RISK ESTIMATOR

Master's Thesis
in Quantitative Methods in Management

Author:
Akseli Leino

Supervisors:
Prof. Luis Alvarez Esteban
Ph.D. Mika Hannula

9.6.2020
Helsinki

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

# CONTENTS

# FIGURES



# TABLES

# 1 INTRODUCTION

Realistic estimations for risk measures are needed throughout finance industry. Banks' risk management departments calculate different measures to assess the risks the banks are facing daily. The single most important measure is Value-at-Risk (VaR) which tells what is the maximum amount of money the bank can lose with given probability. (Hull, 2015, 255) Banks and other financial institutions use VaR for internal planning, but also need to report it to regulators. Since the 1996 amendment to the 1988 Basel Accord, banks have been required to keep capital for market risks generated from trading. Sophisticated banks were allowed to use internal models, which meant calculating VaR and basing the market risk capital on that. (Hull, 2015, 333-334)

VaR estimation boils down to estimating the distribution of returns. That is because VaR is the $\alpha$-quantile of future portfolio return distribution. The exact value of $\alpha$ is arbitrary. Normally it is some some small percentage, such as 1%. Because estimating future returns is a complex task, many different methods for simulating the distribution of returns have been proposed. The methods can roughly be divided to two categories: parametric and nonparametric. Parametric methods make assumptions about the distribution of returns and nonparametric do not. Sometimes the best methods are combination of these two approaches as shown in chapter 4.

Modeling financial time series presents multiple problems. The underlying data generating process is typically unknown. There can be regime shifts that change the process in future. When observing financial time series, one can detect universal properties which are sometimes called stylized facts (Cont, 2001). These stylized facts include but are not limited to heavy-tails, linear unpredictability, return asymmetry, volatility clustering and leverage effect. The stylized facts of financial time series are presented in chapter 3.

One main method to simulate financial time series has been stochastic processes. These parametric models try to imitate the underlying data generating process. Two widely used methods mentioned in econometric literature are ARCH (Engle, 1982) and GARCH (Bollerslev, 1986). Stochastic processes typically capture some of the universal properties. However it is challenging to recover all of the properties given how dependent the models are on assumptions about the underlying distribution. Furthermore the complex dynamics of financial markets makes it demanding for explicit mathematical models to describe the true data generating process.

Nonparametric models rely on historical returns to produce estimates for the future. Machine learning models have been applied recently for the task. (Takahashi, Chen and Tanaka-Ishii, 2019; Fu, Chen, Zeng, Zhuang and Sudjianto, 2019; Wiese, Knobloch, Korn and Kretschmer, 2019). One particular model, namely generative adversarial network (GAN), is well suited for the task. GAN is a type of neural network. GAN aims to learn to generate new data with the same properties as the given training set. GAN includes two neural networks: generator $G$ and discriminator $D$. The generator attempts to learn a mapping from random noise to training data space. The discriminator tries to classify whether given example $\boldsymbol{x}$ came from the original training set or from the generator $G$. $D$ outputs the probability that $\boldsymbol{x}$ came from the original training set. $D$ is trained to maximize the probability of predicting the correct label for examples from the training set and the generator. Simultaneously $G$ is trained to minimize $\log(1 - D(G(\boldsymbol{z})))$, where $\boldsymbol{z}$ is random noise.

Looking at the definition of GAN above one can easily see the potential of applying them to the financial domain. They are designed to produce new data with the same properties as the training data. In other words, given the empirical time series, GAN aims to generate fake time series with the same universal properties as the real time series. If this fake time series is obtained from training on trading portfolio returns, one can calculate the VaR from the time series. The GAN model is discussed in more detail at section 2.5 and the specific architecture of the model used in this thesis can be found at section 5.2.2.

## 1.1 Research questions

This thesis aims to study how well a GAN model can estimate the daily VaR of a stock index. GAN is compared to a simple historical simulation baseline. Three research questions arised from planning of this thesis:

- Can GAN based method beat the simple baseline?
- Which architecture is most suitable for VaR estimation?
- How well does the GAN model replicate universal properties of stock returns?

## 1.2 Structure of the thesis

The thesis is structured as follows. In the second chapter neural networks and GANs are presented. The theory behind them is explained to give the reader basic understanding of the concepts. Third chapter introduces the universal properties of financial time series. In the fourth chapter some widely used methods for VaR estimation are introduced. The methodology is described in the fifth chapter. This includes the backtesting methodology and the specific architecture of the GAN model. Sixth chapter examines and analyses the results of the experiments. First the data used in the experiments is defined. Then I will examine the statistical properties replication capabilities of the GAN model. Finally the results of VaR backtesting are presented. In the seventh chapter conclusion and discussion for future work is made.

# 2 NEURAL NETWORKS

Artificial neural networks are layers of perceptrons. The idea of perceptrons was conceived by Rosenblatt (1958). At the time there was only single layer of perceptrons in the model. Minsky and Papert (1969) showed in their book that the single layer perceptron had limitations. They demonstrated the incapability of single layer perceptron to learn simple boolean XOR function. This was due to the nonlinearity of the XOR. Minsky and Papert (1969) argued that to learn nonlinear functions there must be several layers of perceptrons. Rosenblatt's algorithm for learning only works with a single layer.

The answer to learning nonlinear functions with the multilayer perceptron (MLP) came with backpropagation. It was first proposed as the learning algorithm for multilayer perceptrons by Werbos (1974). It wasn't until 1986 when Rumelhart et al. popularized the method. Backpropagation enabled the flow of gradients by utilizing the chain rule of calculus. MLP is also known as a feedforward neural network.

Hornik, Stinchcombe and White (1989) demonstrated that multilayer perceptrons with at least one hidden layer could approximate any Borel measurable function from finite dimensional space to another given enough perceptrons or neurons in the layer. This result has lead to the notion of describing multilayer perceptrons or feedforward neural networks as universal approximators.

These major developments in the history of neural networks were followed in the next decades with several improvements. According to Goodfellow, Bengio and Courville (2016) two of the main reasons for the popularity and performance of neural networks are larger datasets and more powerful computers. Larger datasets have helped with the issue of statistical generalization of neural networks. Computing power enables the neural networks to have more layers and units per layer.

## 2.1 MLP

Multilayer perceptron or feedforward neural network is the most basic deep learning model. Feedforward network tries to approximate some function $f^*$. This function is typically a mapping from the input $\boldsymbol{x}$ to output $\boldsymbol{y}$. It thus defines a mapping $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$. To learn the mapping the network needs to estimate the parameters $\boldsymbol{\theta}$ that produce the best approximation. (Goodfellow et al., 2016, 164.)

The network is called feedforward because information flows from the input to output. The name network comes from the fact that the model is usually constructed from multiple functions. This way the feedforward neural network can be associated with directed acyclic graph which describes how the functions are composed together. If the network has $n$ functions in a chain we will have

$$f(\boldsymbol{x}) = f^{(n)} \circ f^{(n-1)} \circ \cdots \circ f^{(1)}. \tag{1}$$

This chain structure is commonly used in neural network models. Each of the functions are called layers. Thus equation 1 describes an $n$-layer feedforward neural network. The number of layers in the network determines the depth of the model. The first layer is commonly known as the input layer and the last layer as the output layer. All layers between those two are called hidden layers. The name hidden comes from the fact that the training data does not explicitly show the desired output for these layers. (Goodfellow et al., 2016, 164–165.)

Neural networks are called neural because they are inspired by neuroscience. The hidden layers consists of multiple units which are analogous to neurons. Since the hidden layers are normally vector valued, each unit or neuron of the layer is an element of the vector. Neuroscience has inspired the use multiple layers of vector valued representations and the choice of functions $f^{(i)}(\boldsymbol{x})$. Despite all these connections to neuroscience modern research in neural networks is guided mostly by mathematical sciences. After all the goal of neural networks is to approximate some function, not to model how brains function. (Goodfellow et al., 2016, 165.)

The basic 2-layer feedforward neural network with input $\boldsymbol{x}$ can be written as

$$\begin{aligned} \boldsymbol{h} &= a(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) \\ \boldsymbol{y} &= \sigma(\boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2) \end{aligned} \tag{2}$$

or alternatively

$$\boldsymbol{y} = \sigma(\boldsymbol{W}_2 a(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are called weight matrices and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are called biases. The function $a$ is called activation function and it is applied elementwise. This function is usually some nonlinear function like hyperbolic tangent function. The activation function makes the network nonlinear. Function $\sigma$ maps the values from hidden layer to final output. This function is typically the identity function for regression

tasks and softmax

$$\sigma(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{3}$$

for K-class classification. (Goodfellow et al., 2016, 168, 181, 188.)

In figure 1 is an example of 2-layer feedforward neural network. Next I will examine the learning algorithm for neural networks.

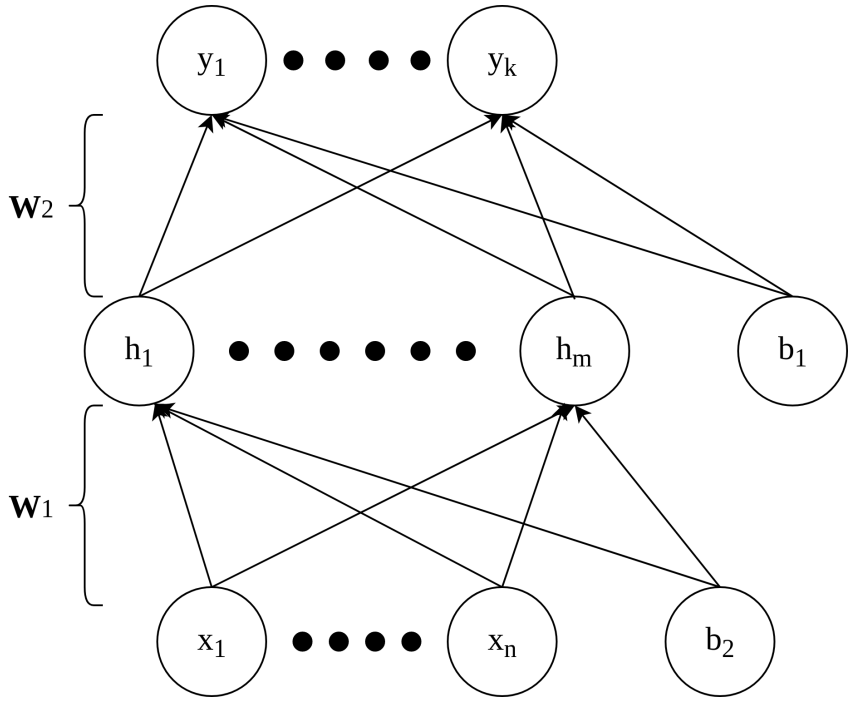


Figure 1: Feedforward neural network. There is $n$ elements in the input vector $\boldsymbol{x}$, $m$ elements in the hidden vector $\boldsymbol{h}$ and $k$ elements in the output vector $\boldsymbol{y}$. Therefore the input weight matrix $\boldsymbol{W}_1$ is $m \times n$-matrix and the hidden layer weight matrix $\boldsymbol{W}_2$ is $k \times m$-matrix.

## 2.2 Estimating parameters

So far the presented feedforward neural network has had known parameters $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$. But given a dataset these parameters need to be estimated to fit neural network to the data. In machine learning language estimating parameters means *training* the model. Often it is also known as the *learning* step. Estimating parameters happens typically by defining a cost or a loss function for the task and then minimizing that. For most cost functions the nonlinearity of

neural networks means that they become nonconvex. Therefore convex optimization algorithms that guarantee a global minimum cannot be used. Gradient-based methods are used instead, such as stochastic gradient descent (SGD). SGD has no convergence guarantee for nonconvex cost functions. It will depend on the initial values of the parameters to determine how good the reached local minimum is. Swirszcz, Czarnecki and Pascanu (2016) explore the local minima problem of neural networks. Their hypothesis is that neural network learning is well behaved only conditioned on the structure of the data. Despite the impossibility of finding the global minimum, neural networks produce incredible results in practice. There is no consensus why local minimum is enough for practical problems.

Stochastic gradient descent differs from gradient descent by taking the average gradient over a minibatch of i.i.d examples from the training dataset. This iterative method gives the following update rule for the parameters:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L_i(\boldsymbol{\theta}),$$

where $m$ is the size of the i.i.d sample from training dataset, $L_i(\boldsymbol{\theta})$ the value of the cost function at example $i$ from the sample and $\eta$ the step size or *learning rate*. (Goodfellow et al., 2016, 290–291.)

Neural networks are usually trained with maximum likelihood. The cost function is then the negative log-likelihood. Negative log-likelihood is the same as cross-entropy between the training data and the model distribution. The cross-entropy is

$$L(\boldsymbol{\theta}) = -\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim\hat{p}_{data}} \log p_{model}(\boldsymbol{y}|\boldsymbol{x}). \tag{4}$$

The $p_{model}$ determines the specific form of the cost function. Deriving the cost function from maximum likelihood eliminates the need to design a different cost function for every model. If the model is $p(\boldsymbol{y}|\boldsymbol{x})$ then the cost function is $\log p(\boldsymbol{y}|\boldsymbol{x})$. (Goodfellow et al., 2016, 174–175.)

Choosing cost functions depends on the choice of the output activation functions. Because usually the cost function is the cross-entropy from equation (4), the activation function of the output layer defines the specific form of the cross-entropy function. (Goodfellow et al., 2016, 173–174.)

The most simple kind of activation is just a linear transformation of the values. This is typically used in regression problems. From equation (2) we get the hidden vector and apply linear transformation

$$\hat{\boldsymbol{y}} = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2$$

to get the output. In other words the output activation function $\sigma(\cdot)$ is just the identity function. These linear output layers can be used to yield the mean of a conditional Gaussian distribution

$$p(\boldsymbol{y}|\boldsymbol{x}) = N(\boldsymbol{y}; \hat{\boldsymbol{y}}, \boldsymbol{I}).$$

Now the maximum likelihood is the same as minimizing the mean squared error. Covariance can also be estimated with the maximum likelihood approach, but the covariance matrix must be positive definite for all inputs. This constraint makes it hard for linear transformation and usually other transformations are used. (Goodfellow et al., 2016, 178.)

For binary classification tasks the maximum likelihood method defines a Bernoulli distribution $\mathbb{P}(y = 1|\boldsymbol{x})$. Neural network will thus predict a probability $\mathbb{P}(y = 1|\boldsymbol{x})$ and this number must lie in the range $[0, 1]$. To meet this constraint the output activation used is the logistic sigmoid function. The output is then

$$\hat{y} = \sigma(\boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2)$$

where $\sigma$ is

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

(Goodfellow et al., 2016, 178–180.)

Multi-class classification utilizes the softmax function. The softmax defines a discrete probability distribution over the classes. The softmax function is presented in equation 3. Both the sigmoid and softmax have the useful property of having the exponential function. The maximum likelihood training undoes the exponential function because the cost function is $-\log p(\boldsymbol{y}|\boldsymbol{x})$. (Goodfellow et al., 2016, 180–181.)

When using gradient based learning algorithms, such as SGD, gradients are needed. Backpropagation is the method to calculate the gradient. In the context of neural networks, backpropagation means the retrieval of the gradient of the cost function respect to the parameters. This means evaluating the expression $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. (Goodfellow et al., 2016, 200.)

At the core of backpropagation is the chain rule of calculus. The chain rule states that if there is functions composed of other functions we can calculate the derivative of this composition. In the most simple case there is functions $y = g(x)$ and $z = f(g(x))$. The chain rules states that

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}. \tag{5}$$

To generalize this to multivariable case with $\boldsymbol{y} = g(\boldsymbol{x})$ and $z = f(g(\boldsymbol{x}))$ we get

$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z, \tag{6}$$

where $\left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)$ is the Jacobian of $g$. (Goodfellow et al., 2016, 201, 203.)

The algorithm 1 demonstrates the calculation of backpropagation for feedforward neural network. In software implementations of the backpropagation the

**Algorithm 1:** Backpropagation in feedforward neural network

Let $\boldsymbol{a} = \{a_1, \ldots, a_d\}$ be the activations before the activation function is applied

Let $f_i$ be the activation function in layer $i$

$\boldsymbol{g} \leftarrow \nabla_{\hat{y}}$

**for** *k = d, d-1, . . . , 1* **do**

Get the gradient before the nonlinear activation with elementwise multiplication:

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{a}_k} L = \boldsymbol{g} \odot f'(\boldsymbol{a}_k)$

Get the gradients of weights and biases:

$\nabla_{\boldsymbol{b}_k} L = \boldsymbol{g}$

$\nabla_{\boldsymbol{W}_k} L = \boldsymbol{g}\boldsymbol{h}_{k-1}^{\top}$

Update the weights and biases using the SGD

$\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{h}_{k-1}} L = \boldsymbol{W}_k^{\top} \boldsymbol{g}$

**end**

algorithm is usually more complicated to cater for the inefficiencies of the naive method. When calculating the gradient there can be multiple subexpressions that are repeated in the calculation. It is therefore computationally cheaper to store the values of these subexpressions to some variables for following calculations. (Goodfellow et al., 2016, 217.)

## 2.3 Optimizers

Because training a neural network succesfully is a difficult problem and it is computationally expensive, many sophisticated optimization techniques have been developed. One main difference between pure optimization and optimization in ma-

chine learning is the indirect approach in machine learning. Normally there is some cost function $L(\boldsymbol{\theta})$ that the optimization procedure minimize. However, in machine learning the end goal is not to optimize this cost function directly. There is typically some external statistic that is computed on the test set. This statistic tends to be intractable for optimization. In machine learning the generalization of the model to unseen examples is more important than to get the absolute minimum cost function value in training. (Goodfellow et al., 2016, 272.)

The basic SGD algorithm is used in practice but some modifications are also used to make training easier. One method to make SGD faster is the idea of momentum (Polyak, 1964). It keeps moving to the direction of past gradients by aggregating them in exponentially decaying moving average. Formally the update rule is

$$\boldsymbol{v}_{k+1} = \alpha \boldsymbol{v}_k - \eta \nabla_{\boldsymbol{\theta}} \frac{1}{m} \sum_{i=1}^{m} L_i(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \boldsymbol{v}_{k+1}$$

where $\boldsymbol{v}_k$ is a velocity that is set to some initial value at the start and $\alpha$ is a hyperparameter which determines how rapidly the previous gradients decay.The decaying parameter $\alpha$ lies in the interval $[0, 1)$. (Goodfellow et al., 2016, 292–293.)

Researchers in the neural network field have discovered that the learning rate is one of the most difficult hyperparameters to choose. This is because the learning rate has substantial effect on the performance of the model. To make the learning rates better, adaptive methods have been developed. These methods change the learning rate during the model training. Next I will present some adaptive algorithms. (Goodfellow et al., 2016, 302–303.)

AdaGrad (Duchi, Hazan and Singer, 2011) adapts the learning rates of the model parameters by inversely scaling them proportional to the square root of the sum of all previous squared values of the gradient. The AdaGrad has some nice properties for convex optimization but in practice it can suffer from the accumulation of squared gradients that can decrease the learning rate too much (Goodfellow et al., 2016, 303). Algorithm 2 presents the AdaGrad algorithm.

RMSProp (Hinton, Srivastava and Swersky, 2012) changes the AdaGrad to accumulate gradients in exponentially weighted moving average. While AdaGrad is designed to converge quickly in convex functions, the RMSProp makes it better for nonconvex functions by discarding the distant past of gradients (Goodfellow et al., 2016, 303–304). The RMSProp is widely used in practice because of its

**Algorithm 2:** AdaGrad

Let $\epsilon$ be some very small constant

Let $\boldsymbol{r} = 0$

Let $\eta$ be the global learning rate

**while** *model not converged* **do**

Calculate the gradient:

$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L_i(\boldsymbol{\theta})$

Accumulate the squared gradients:

$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

Calculate the update and apply to the parameters:

$\Delta\boldsymbol{\theta} \leftarrow -\frac{\eta}{\epsilon + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

**end**

effectiveness (Goodfellow et al., 2016, 305). The RMSProp algorithm is shown in algorithm 3.

Adam (Kingma and Ba, 2014), meaning adaptive moment estimation, combines the advantages of AdaGrad and RMSProp. According to Kingma and Ba (2014) the benefits of Adam are that size of parameter updates are invariant to rescaling of the gradient, learning rates are bounded by a hyperparameter, it can handle sparse gradients, objective doesn't need to be stationary and it does learning rate annealing. Adam is demonstrated in algorithm 4.

All the presented algorithms try to focus on certain challenges of optimizing neural networks. But there is no consensus on what algorithm is the best. Choi, Shallue, Nado, Lee, Maddison and Dahl (2019) discovered that more general optimizers never underperform their special cases in empirical tests. They suggest tuning all the metaparameters of the popular adaptive methods by training the model several times. This of course can be computationally expensive, if the models are large. Goodfellow et al. (2016, 306–307) also give credit to the robustness of adaptive methods, but acknowledge the nonexistence of the single best method.

## 2.4 Regularization

The end goal of a machine learning model, such as a neural network, is to have a good performance on a given dataset. This means the training data and new

**Algorithm 3:** RMSProp

Let $\epsilon$ be small constant

Let $\rho$ be the decay rate

Let $\boldsymbol{r} = 0$

Let $\eta$ be the global learning rate

**while** *model not converged* **do**

Calculate the gradient:

$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L_i(\boldsymbol{\theta})$

Accumulate the squared gradients:

$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Calculate the update and apply to the parameters:

$\Delta \boldsymbol{\theta} \leftarrow -\frac{\eta}{\sqrt{\epsilon + \boldsymbol{r}}} \odot \boldsymbol{g}$

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

**end**

**Algorithm 4:** Adam

Let $\eta$ be the learning rate

Let $\beta_1, \beta_2 \in [0, 1)$ be the exponential decay rates for the moment estimates

Let $\epsilon$ be small constant

Let $t = 0$

**while** *model not converged* **do**

$t \leftarrow t + 1$

$\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{m} L_i(\boldsymbol{\theta})$

First moment estimate:

$\boldsymbol{m} \leftarrow \beta_1 \boldsymbol{m} + (1 - \beta_1) \boldsymbol{g}$

Second moment estimate:

$\boldsymbol{v} \leftarrow \beta_2 \boldsymbol{v} + (1 - \beta_2) \boldsymbol{g} \odot \boldsymbol{g}$

Compute bias-corrected moments:

$\hat{\boldsymbol{m}} \leftarrow \frac{\boldsymbol{m}}{1 - \beta_1^t}$

$\hat{\boldsymbol{v}} \leftarrow \frac{\boldsymbol{v}}{1 - \beta_2^t}$

Update parameters:

$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\hat{\boldsymbol{m}}}{\sqrt{\hat{\boldsymbol{v}}} + \epsilon}$

**end**

unseen data. Methods designed for reducing the test error are called *regularization* methods. Reducing the test error causes the model generalize better, but comes at the expense of sometimes increasing training error. (Goodfellow et al., 2016, 224.)

Several distinctive regularization strategies have been developed. Some strategies add constraints to the parameter values. Some introduce extra terms to the cost function. These constraints can express different purposes, such as prior knowledge or preference for simpler models. Then there are ensemble methods that group together several models and combine their hypotheses. (Goodfellow et al., 2016, 224–225.)

Regularization of a neural network is typically based on regularizing the estimator. Regularizer increases bias and decreases variance. For good results the bias must not increase too much. The goal of regularization is to prevent overfitting. Overfitting happens when a complex model is trained, but the trained model doesn't reflect the true data generating process. Because in practice the true data generating process is unattainable, regularization is needed to prevent the model from fitting the training data too well. Next I will introduce few regularization methods used in training of neural networks. (Goodfellow et al., 2016, 225.)

Parameter norm penalties limit the capacity of the model by introducing an extra term to the cost function. Regularized cost function is then

$$\tilde{L}(\boldsymbol{\theta}) = L(\boldsymbol{\theta}) + \alpha\Omega(\boldsymbol{\theta}),$$

where $\alpha \in [0, \infty)$ is a hyperparameter adjusting the weight of the norm penalty $\Omega$. Larger values of $\alpha$ will result to more regularization. Minimizing the regularized cost function will decrease the original cost function and also the norm of the parameters. Some commonly used norms are the $L^1$ and $L^2$ norms. (Goodfellow et al., 2016, 226–227, 230.)

Augmenting the dataset with more data is one way to regularize the model. It will usually generalize better. Obtaining new data can be daunting task and therefore methods to create fake data have emerged. This method of creating fake data is a challenging problem and depending on the task it doesn't necessarily improve generalization of the model (Goodfellow et al., 2016, 236). In this thesis the data used is a financial time series. The task is to generate new data from some distribution. To do that the model needs to estimate the density. But to augment the dataset, knowledge of the density is needed. Because the task itself is estimating the density, it is difficult to generate fake data (Goodfellow et al., 2016, 236).

During the training of models, which are large enough to overfit, the training cost tends to decline at every iteration or *epoch* over the training dataset. However the validation cost can start to rise again, hurting the generalization of the model. To overcome this obstacle a rather simple method of storing the parameter values $\boldsymbol{\theta}$ at every epoch and returning the parameter setting where the validation cost is the lowest has been established. This regularization method is called early stopping. It is implemented by terminating the training step, if the validation cost hasn't decreased for some predefined amount of epochs. Then it returns the parameter setting with the lowest cost in the validation dataset. Early stopping is quite popular due to its simplicity and effectivity. (Goodfellow et al., 2016, 241, 243.)

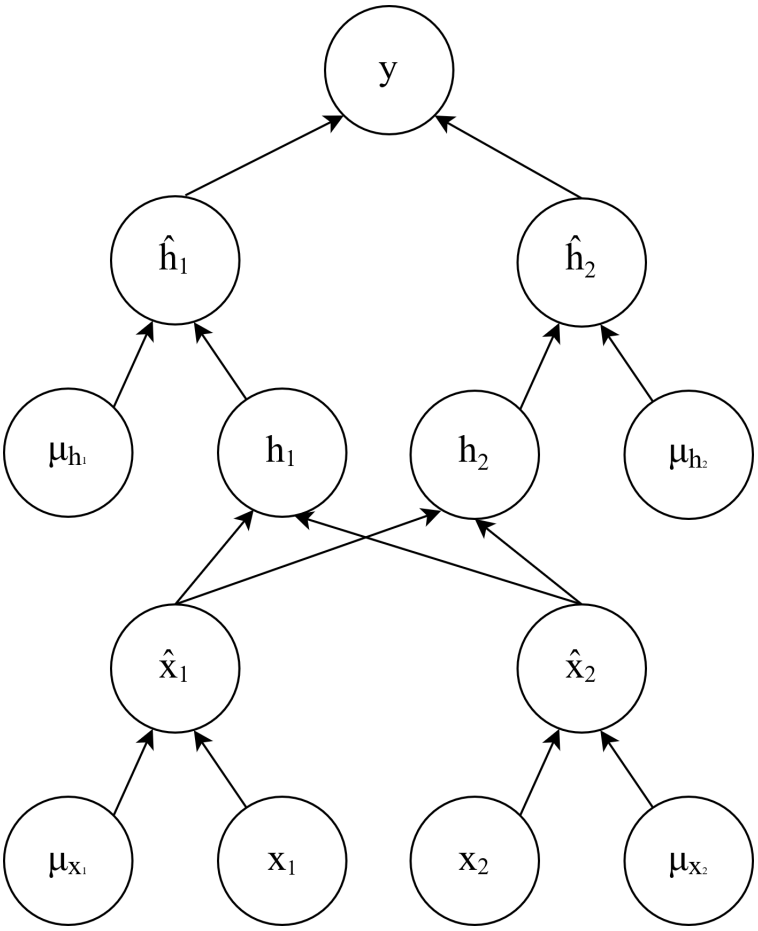


Figure 2: Dropout applied to forward propagation of feedforward neural network with two input and two hidden units. The vector $\boldsymbol{\mu}$ is the randomly sampled binary mask having 0 or 1 for every input and hidden unit. Each nonoutput unit is multiplied with the mask vector.

Another effective regularization method is dropout. Dropout enables combining predictions from multiple subnetworks (Srivastava, Hinton, Krizhevsky, Sut-

skever and Salakhutdinov, 2014). It is quite similar to bagging, where multiple models are trained and their predictions combined. Training large neural network can take a lot of time making it impractical to train several models independently. Dropout is a method to address this problem. Dropout regularization trains ensemble of subnetworks, which are formed by deleting units from the network excluding output units. This can be accomplished easily by multiplying the output of the unit with zero. At each step in the stochastic gradient descent -based algorithm a random binary mask is applied to the input and hidden units. Then the forward- and backpropagation and parameter updates are carried on as usually. The probability of a mask value 1 for some unit is a hyperparameter that is fixed. Figure 2 illustrates dropout. (Goodfellow et al., 2016, 255, 257.)

The difference between bagging and dropout comes from the fact that all the models trained in bagging are independent. In dropout the model shares the parameters between different steps in the training process.

## 2.5 Generative adversarial networks

Generative adversarial networks, known as GANs, were first proposed by Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville and Bengio (2014). They are generative models which have two components. There is a generator $G$ and discriminator $D$. The generator estimates the data distribution. The discriminator estimates the probability that a sample came from training data not generator $G$. During the training the objective of $G$ is to maximize the probability of mistake for $D$. This setup is equivalent to two-player minimax game. If both $G$ and $D$ are feedforward neural networks, the whole model can be trained with backpropagation. After the training, new fake samples can be obtained by passing random noise with forward propagation through the generator.

When both the $G$ and $D$ are feedforward networks the two-player minimax game with value function $V(G, D)$ can be written as

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

A prior $p_z(z)$ is defined on the input noise variables. Implementing the game involves alternating between optimizing $D$ and optimizing $G$. Algorithm 5 demonstrates the basic procedure for training a GAN. (Goodfellow et al., 2014.)

In their paper Goodfellow et al. (2014) show that the probability distribution $p_G$ defined by the generator $G$ has a global optimum $p_g = p_{data}$ in the minimax game,

**Algorithm 5:** GAN training with stochastic gradient descent

Let $k$ be a hyperparameter determining the number of steps to optimize discriminator

Let $\eta$ be the learning rate

**for** *number of training iterations* **do**

  **for** *k steps* **do**

    Sample minibatches of priors $\{z_1, \ldots, z_m\}$ from $p_G(z)$ and $\{x_1, \ldots, x_m\}$ from $p_{data}(\boldsymbol{x})$

    Update the parameters of $D$ by ascending the gradient:

    $\boldsymbol{\theta}_D \leftarrow \boldsymbol{\theta}_D + \eta \nabla_{\boldsymbol{\theta}_D} \frac{1}{m} \sum_{i=1}^{m} [\log D(x_i) + \log(1 - D(G(z_i)))]$

  **end**

  Sample minibatch of priors $\{z_1, \ldots, z_m\}$ from $p_G(z)$

  Update the parameters of $G$ by descending the gradient:

  $\boldsymbol{\theta}_G \leftarrow \boldsymbol{\theta}_G - \eta \nabla_{\boldsymbol{\theta}_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i)))$

**end**

if the model is large enough and training time is unrestricted. They also discuss the advantages and disadvantages of GANs. Some of the advantages they mention are that backpropagation can be used, no need for inference during learning and sharper distributions can be represented. The disadvantages are that there is no explicit formulation of $p_G(x)$ and the training must be synchronized well in order to avoid $G$ to map too many $z$ to the same value of $x$. This scenario, named "Helvetica scenario", happens when $G$ is trained too much without updating $D$ and results in having too little diversity in $p_G$ to model $p_{data}$.

GANs have been applied succesfully to many applications such as image generation (Radford, Metz and Chintala, 2015), music generation (Uricar, Krizek, Hurych, Sobh, Yogamani and Denny, 2019) and even the study of dark matter (Mustafa, Bard, Bhimji, Lukić, Al-Rfou and Kratochvil, 2019).

### 2.5.1 Financial applications

Recently GANs have gathered also the interest of researchers in the financial domain. Several problems in finance need simulated synthetic data. The simulated data needs to be of good quality and represent the complex real world. Because methods for generating financial data, especially time series data, have mainly

relied on parametric stochastic processes, the need for more realistic generative models without assumptions about the distribution of the data generating process have emerged.

Wiese et al. (2019) introduce a data driven financial time series generator which they have named Quant GAN. Their GAN model modifies the original GAN by replacing the feedforward neural networks with temporal convolutional networks. They claim that the temporal convolutional network catches the long range dependencies of the time series better than a vanilla feedforward neural network. Using S&P 500 daily log returns they demonstrate the capabilities of their model. Their model outperforms GARCH(1,1) in four metrics. The four metrics are Wasserstein-1 distance, DY metric, ACF score and leverage effect score.

Takahashi et al. (2019) compare the reproducibility of certain stylized facts of financial time series using GANs with feedforward neural networks, convolutional neural networks and the combination of those two. The stylized facts they test are linear unpredictability, heavy-tails, volatility clustering, leverage effect, coarse-fine volatility correlation and gain-loss asymmetry. In their study only the feedforward neural network generator can reproduce all the stylized facts.

Fu et al. (2019) apply a particular GAN model called conditional GAN (CGAN) (Mirza and Osindero, 2014) to time series generation. They test the performance of CGAN compared to historical simulation in value-at-risk estimation. They also apply the model to economic modeling. By utilizing the CGAN they can condition the model to normal and stressed periods in the market. They use two stocks, JPM and WFC, as examples in the empirical test. Then comparison between historical simulation and CGAN method is conducted. They find that the CGAN method doesn't underestimate the VaR and expected shortfall, in contrast to the historical simulation.

Marti (2019) proposes GANs for financial correlation matrix generation. The GAN model he employs is deep convolutional GAN. He shows that his approach is capable of replicating the main stylized facts of correlation matrices. The data for experiments is daily returns of S&P 500.

Mariani, Zhu, Li, Scheidegger, Istrate, Bekas and Malossi (2019) address the problem of portfolio optimization. They utilize a GAN model to simulate several realistic paths for future asset prices and optimize the portfolio based on the simulations. Thus their GAN models the future probability distribution of returns. For empirical tests several US and European stocks and ETFs are used. Then they apply their portfolio optimization approach to the stocks and compare that with the traditional Markowitz optimization. In out-of-sample experiments the

GAN portfolios outperform Markowitz portfolios, with respect to Sharpe ratio, in almost all risk settings.

In addition to all these studies Zhou, Pan, Hu, Tang and Zhao (2018) and Zhang, Zhong, Dong, Wang and Wang (2019) applied GANs to predict stock prices. Koshiyama, Firoozye and Treleaven (2019) used a CGAN model to fine tune trading strategies.

# 3 PROPERTIES OF FINANCIAL TIME SERIES

Financial time series have certain characteristics that present problems to the researcher. The future of the time series contains significant amount of uncertainty. Financial markets are complex systems with millions of participants. This makes it difficult for statistical models to estimate the data generating process. When estimating the data generating process, the models have been traditionally parametric. Parametric models need a lot of calibration to replicate all the stylized facts of financial time series. Nonparametric data-driven models, such as generative adversarial networks, rely only on the empirical data and can represent highly nonlinear functions.

Cont (2001) presents three problems facing the application of statistics to financial asset returns. The first is stationarity. Let $S_t$ be the price of an asset at time $t$. Stationarity means that the joint distribution of the returns

$$r_{t_1}, \ldots, r_{t_k},$$

where $r_{t_i} = \log S_{t_i+\Delta t_i} - \log S_{t_i}$ and $t_i$ is time instant, is equal to the joint distribution of returns

$$r_{t_1+\tau}, \ldots, r_{t_k+\tau},$$

where $\tau$ is some time interval. This means that the unconditional joint distribution stays the same, when returns are shifted in time. Most statistical analysis relies on the existence of the statistical properties to be stable over time. Depending on the time frame asset returns are calculated, the stationarity of the returns doesn't hold. There are so-called seasonality effects, such as January effect and weekend effect.

The second problem arises from the fact that the empirical averages need to converge to the values they are trying to estimate. Some sample moment is defined by

$$\widehat{f(r_t)} = \frac{1}{N} \sum_{t=1}^{N} f(r_t).$$

Stationarity is needed to ensure that the distribution of the asset returns doesn't depend on $t$. However this is not sufficient to guarantee the convergence of the sample moment to its expectation $\mathbb{E}[f(r_t)]$. There needs to be an ergodic property to ensure the convergence. Typically this is met with i.i.d observations, but asset returns are not i.i.d. (Cont, 2001, 225.)

The third problem is the properties of finite samples. The statistical estimator isn't necessarily equal to the quantity the estimator estimates. Because daily

asset returns correspond to only some thousands of data points the error of the estimator becomes significant. That is why confidence intervals are necessary. But computing the confidence interval presents new problem. The residuals or noise terms of the return process must be i.i.d and their higher order moments well defined. The heavy tails and nonlinear dependency of empirical asset returns don't support this. (Cont, 2001, 225.)

Cont (2001, 224) lists multiple statistical properties common to financial assets:

**1**) **No autocorrelation**. Asset returns don't have significant autocorrelations except on small intraday time intervals.

**2**) **Heavy tails**. The distribution of returns exhibits a power-law type of tail behaviour with a finite tail index. This means that normal distribution is not a suitable for expressing the tails of the distribution.

**3**) **Asymmetry of gains and losses**. Large declines in asset prices are more frequent than equally large increases in prices.

**4**) **Aggregational Gaussianity**. The distribution of returns looks increasingly similar to Gaussian distribution when the time scale of the return calculation is increased.

**5**) **Intermittency**. Asset returns at all time scales have a lot of variability. Time series of volatility of the returns have irregular bursts.

**6**) **Volatility clustering**. Although the returns don't have significant autocorrelation, the volatility of returns does. Especially periods of high volatility tend to cluster.

**7**) **Conditional heavy tails**. When returns have been corrected for volatility clustering, the remaining time series still displays heavy tails although not as heavy as the unconditional distribution.

**8**) **Slow decay of autocorrelation in absolute returns**. The autocorrelation of absolute returns decays like power law with an exponent $\beta \in [0.2, 0.4]$.

**9**) **Leverage effect**. Returns of an asset is negatively correlated with the volatility of the asset.

**10**) **Correlation of volume and volatility**. The volume of transactions with an asset is correlated with the volatility of the asset.

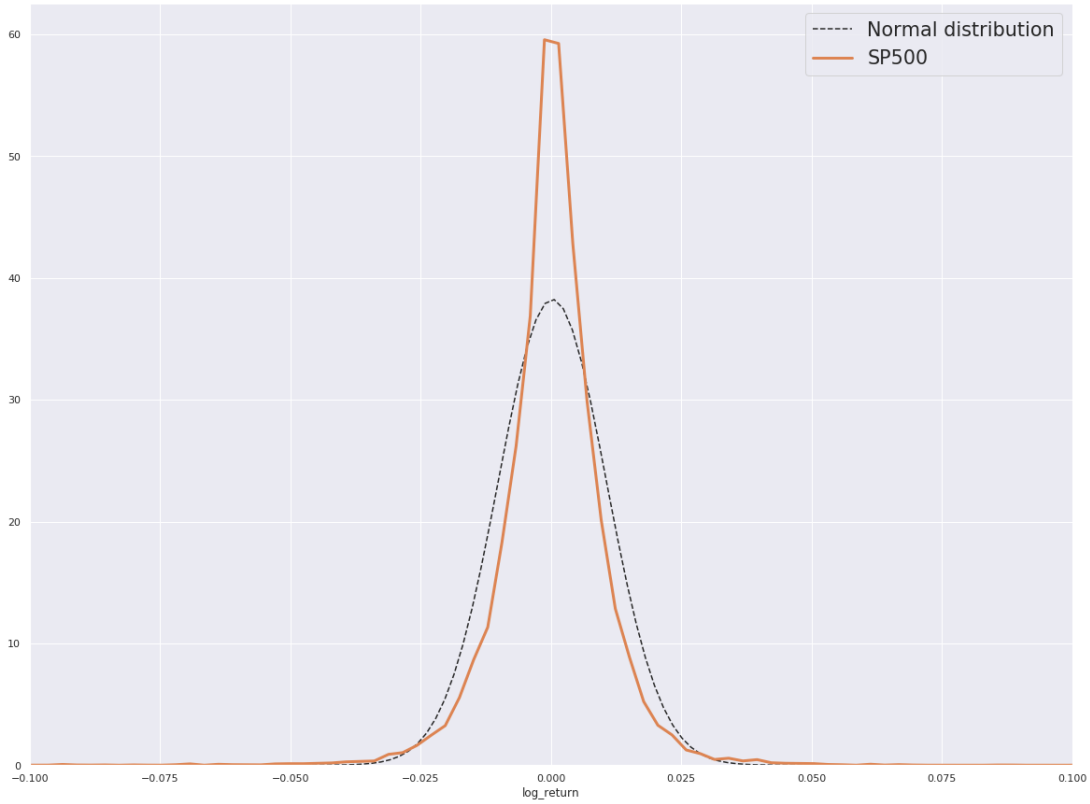


Figure 3: S&P500 log return distribution

11) **Asymmetry of volatility time scales**. Measures of volatility that are coarser, predict the finer volatility measures better than the finer predict the coarser measures.

Next I will delve into some of the stylized facts that Cont (2001) also studies in more detail.

## 3.1 Heavy tails

Already in 1963, Mandelbrot demonstrated the inadequacy of the normal distribution for modeling financial asset returns and their heavy tails. Cont (2001) measures the deviation of the empirical distribution from normal distribution with excess kurtosis. The excess kurtosis of the normal Gaussian distribution equals to zero. Empirical distributions have excess kurtosis that is positive, which indicates heavy tails. According to Cont (2001, 226), the excess kurtosis of five minute asset returns for two currency futures and one stock index future lies between around 16 and 60. It can be said that the distribution of returns is more heavy tailed and

sharp peaked than Gaussian distribution. This can be seen from figure 3. As Cont (2001, 226) points out, the features outlined in the previous sentence are insufficient to determine a fitting distribution. Many distributions have been proposed in the literature: stable distributions (Mandelbrot, 1963), hyperbolic distributions (Eberlein, Keller and Prause, 1998), the Student distribution (Blattberg and Gonedes, 1974), normal inverse Gaussian distributions (Barndorff-Nielsen, 1997) and countless others.

## 3.2 Autocorrelation

A different statistical dispersion measure than the standard deviation is needed to capture the variability of asset returns because the distribution is non-Gaussian. Higher-order moments can be used as measures of dispersion. To use them, it must be known if the moments are well defined. The order of highest absolute finite moment can be determined with the tail index $\xi$ of the distribution. For Gaussian distributions $\xi = \infty$, meaning that all of its moments are finite. For power law distributions $\xi = \gamma$, where $\gamma$ is the exponent of the distribution. (Cont, 2001, 227.)

To estimate if the theoretical moment is finite the sample moment can be represented as a function of the sample size (Mandelbrot, 1963). This way the sample moment should converge close to the theoretical value and fluctuate around it if the theoretical moment is finite. Cont (2001, 227) argues that higher than fourth order moments are not stable measures of risk because of their high fluctuations. He also points out that the distribution of returns is similar to power law distribution, but the exponent $\gamma$ contains a lot of uncertainty.

Another property of asset returns is the absence of autocorrelation. Autocorrelation function is

$$C(\tau) = \mathrm{corr}(r_t, r_{t+\tau}).$$

When $\tau \geq 15$ minutes, the autocorrelation function is in practice zero. This is because at these time frames autocorrelation is arbitraged away. On shorter time intervals autocorrelation can exist because it takes time for the market to react to new information. Returns on a very short time intervals, namely high-frequency transaction price returns, display a negative autocorrelation. This effect might happen because the transaction prices bounce between bid and ask prices or a quick mean reversion of the prices. In contrast to high-frequency prices, weekly and monthly returns show positive autocorrelation. The size of the data is smaller on these frequencies and the statistical evidence varies from sample to sample. In

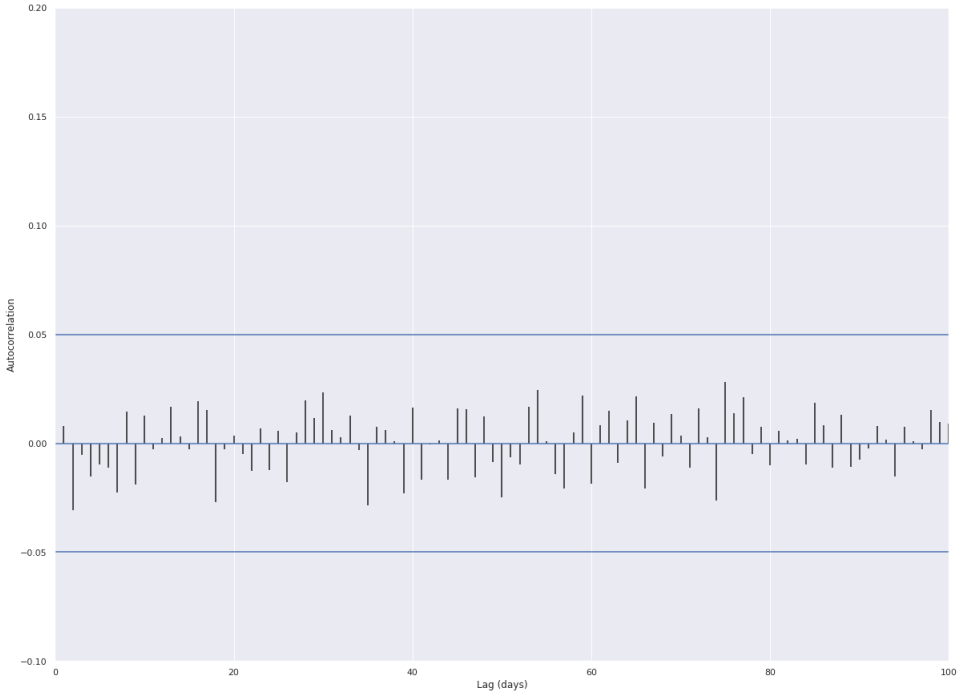


Figure 4: Autocorrelation of daily S&P500 log returns

figure 4 the autocorrelation of S&P500 daily log returns is displayed. It is clear that the autocorrelation hovers very closely around zero. (Cont, 2001, 229–230.)

## 3.3 Volatility clustering

Although the absence of autocorrelation supports the independence of asset returns, it is not enough to imply the independence. Independence demands that any nonlinear function of returns must also have autocorrelation equal to zero. This is in contradiction with the fact that for example absolute returns exhibit positive autocorrelation. The phenomenon of absolute or squared returns having positive autocorrelation is known as volatility clustering. Figure 5 shows the autocorrelation of absolute and squared daily returns of S&P500 stock index. (Cont, 2001, 230.)

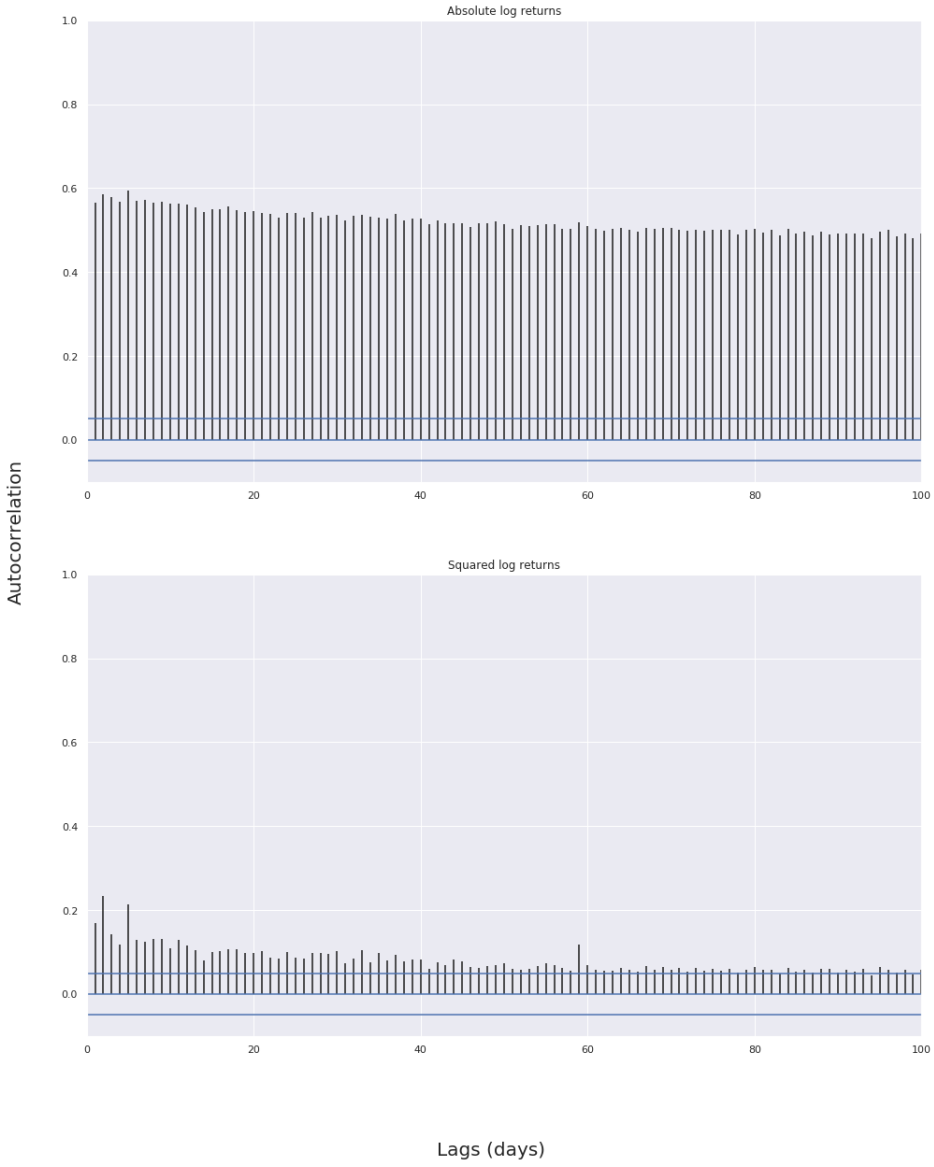


Figure 5: S&P500 autocorrelations of nonlinear transformations of returns.

# 4 VALUE-AT-RISK ESTIMATION METHODS

Baumol (1963) noted that standard deviation is not always a good measure of risk in portfolio optimization. Since then the idea of measuring tail risk has been central to risk management. Value at Risk (VaR) is a risk measure developed in late 1980s to early 1990s at J.P. Morgan (Hull 2015, 256; Nadarajah and Chan 2016, 286). As Hull (2015, 255) explains, VaR is the highest loss a portfolio endures in time $T$ with given confidence level. More formally it is

$$\mathrm{VaR}_\alpha = \inf\{u : F(u) \geq \alpha\}$$

where $1-\alpha$ is the confidence level and $F(\cdot)$ is some cumulative distribution function of portfolio returns in time $T$ (Nadarajah and Chan 2016, 287). In other words $\mathrm{VaR}_\alpha$ is the $\alpha$-quantile of the return distribution. Sometimes VaR is defined as the $1-\alpha$-quantile of negative returns. This way the VaR is a positive value.

Nadarajah and Chan (2016, 287–288) divide the applications of VaR to three major purposes. First is its usage for reporting the aggregate risk in a simple way. Secondly it can be used to set risk limits for different units and traders. Thirdly it helps managing capital across the organization. They also give an extensive list of specific application areas of VaR.

VaR has faced criticism over its properties. It is not coherent risk measure. Artzner, Delbaen, Eber and Heath (1999, 210) defines a coherent risk measure with four mathematical properties. A coherent risk measure should be translation invariant, subadditive, monotonic and positively homogeneous. VaR however is not subadditive. This means that the total VaR of a portfolio can exceed the individual VaRs of its constituent assets. This goes against the diversification principle. In addition to that VaR doesn't tell anything about the size of the loss that exceeds it. In spite of these problems it is still the standard risk measure for financial organizations. Although coherent risk measures exist, such as conditional VaR (expected shortfall), many of them depend on the estimation of VaR. (Chen and Lu 2012, 308.)

Because the estimation of VaR is important and challenging problem, many methods have been proposed. Nadarajah and Chan (2016) give a very thorough overview of different parametric, nonparametric and semiparametric methods. Because the quantity of VaR depends heavily on the cumulative distribution function, it is clear that the problem of estimating VaR boils down to estimating the probability distribution of portfolio returns. As can be seen from chapter 3, the

estimated probability distribution should replicate numerous stylized facts. In the next sections I will introduce few widely used VaR estimation methods.

## 4.1 Historical simulation

Historical simulation method is the simplest nonparametric estimation method. The estimated VaR is simply the $\alpha$-quantile of historical portfolio returns. Given $N$ past portfolio return observations $r_1, r_2, \ldots, r_N$, the order statistic of the returns is $r_{(1)} \leq r_{(2)} \leq \cdots \leq r_{(N)}$. Now the VaR estimate is

$$\widehat{\mathrm{VaR}}_\alpha = r_{(i)},$$

where $\alpha \in \left(\frac{(i-1)}{N}, \frac{i}{N}\right]$. (Nadarajah and Chan 2016, 326–327.)

The main benefit of the historical simulation is its simplicity. It is easy to compute and explain. According to Pritsker (2001, 3) historical simulation might have an advantage over methods that rely on assumptions of normally distributed returns. As seen in chapter 3 of this thesis the empirical properties of asset returns cannot be replicated with normal distribution.

There are however some limitations regarding the naive implementation of the method. The main disadvantage of the method is that the past returns are equally weighted when constructing the empirical cumulative distribution function. This is the same as assuming that the returns are i.i.d. through time. The volatility clustering of asset returns makes this assumption unrealistic. (Pritsker 2001, 3.)

To overcome the limitation of equally weighted method, Boudoukh, Richardson and Whitelaw (1998) introduced a exponentially decaying weighting for the past returns. Their method assigns the most recent returns more weight. In their method the past $N$ returns are weighted by

$$\frac{1-\lambda}{1-\lambda^N}\lambda^{N-i},$$

where $\lambda \in (0, 1]$. Now it is trivial to derive the equally weighted method from this by setting $\lambda = 1$. The normal historical simulation is thus a special case of their method.

Barone-Adesi, Giannopoulos and Vosper (1997) extend the historical simulation method with filtering. They combine the nonparametric historical simulation with parametric GARCH model. Their approach is to scale the historical returns with the ratio of current over past conditional volatility. To not rely on any assumptions about the return distribution, they calibrate the GARCH model to the historical

returns. This gives ARMA-GARCH(1,1) model

$$r_t = \mu r_{t-1} + \phi \epsilon_{t-1} + \epsilon_t, \qquad \epsilon_t \sim N(0, h_t)$$
$$h_t = \omega + \alpha \epsilon_{t-1}^2 + \beta h_{t-1},$$

where $\mu$ is the autoregressive term, $\phi$ the moving average term and $h_t$ the variance of residual $\epsilon_t$ at time $t$. Then to convert the residuals close to a stationary i.i.d. distribution, the residual estimates are divided by corresponding volatility estimate. This gives the standardised residual return

$$e_t = \frac{\hat{\epsilon}_t}{\sqrt{\hat{h}_t}}.$$

These standardised residual returns are then randomly drawed to form a pathway of variances. The first drawn standardised residual return is scaled with one timestep ahead volatility forecast to give an innovation forecast

$$z_{t+1} = \hat{e}_1 \sqrt{\hat{h}_{t+1}}.$$

Now the asset price estimate in the next timestep is

$$\hat{S}_{t+1} = S_t + S_t(\mu r_t + \phi z_t + z_{t+1}).$$

The $\text{VaR}_\alpha$ estimate for one timestep ahead can be obtained by replicating these procedures $N$ times and computing the $\alpha$-quantile of these $N$ returns.

## 4.2 Extreme value theory

Extreme value theory (EVT) is a theory describing the estimation of the tails of a probability distribution. It is helpful in situations where very high confidence levels for VaR estimates is wanted. EVT smooths and extrapolates the tails of an empirical probability distribution. (Hull 2015, 289–290.)

If $F(v)$ is the cumulative distribution function for some variable $v$ and $u$ is a value of $v$ in the right tail of the distribution, the probability of $v$ being between $u$ and $u + y$ $(y > 0)$ is $F(u + y) - F(u)$. The probability that $v$ is between $u$ and $u + y$ conditional on $v > u$ is

$$F_u(y) = \mathbb{P}\{v - u \leq y | v > u\} = \frac{F(u + y) - F(u)}{1 - F(u)}$$

This variable defines the right-side tail of the probability distribution. For many

distributions $F(v)$, the distribution of $F_u(y)$ converges to a generalized Pareto distribution as $u$ is increased. The generalized Pareto distribution is

$$G_{\xi,\beta}(y) = 1 - \left[1 + \xi\frac{y}{\beta}\right]^{-1/\xi}, \qquad \xi \neq 0 \tag{7}$$

The two parameters $\xi$ and $\beta$ need to be estimated from the data. The parameter $\xi$ dictates the shape of the distribution and $\beta$ determines the scale. (Hull 2015, 290.)

Maximum likelihood method can be used to estimate the parameters $\xi$ and $\beta$. First the probability density function needs to be calculated from the cumulative distribution function. Differentiating (7) with respect to $y$ we get

$$g_{\xi,\beta}(y) = \frac{1}{\beta}\left(1 + \xi\frac{y}{\beta}\right)^{-1/\xi-1}, \qquad \xi \neq 0$$

Next the threshold value $u$ is chosen. Then the observations on $v$ are ranked in descending order. Only the observations for which $v > u$ are considered. If there are $N_u$ such observations $v_i$ such that $i \in [1, N_u]$, the likelihood function is

$$\prod_{i=1}^{N_u} \frac{1}{\beta}\left(1 + \xi\frac{v_i - u}{\beta}\right)^{-1/\xi-1}$$

Now the log-likelihood

$$\sum_{i=1}^{N_u} \log\left[\frac{1}{\beta}\left(1 + \xi\frac{v_i - u}{\beta}\right)^{-1/\xi-1}\right]$$

can be maximized with numerical methods to obtain estimates for $\xi$ and $\beta$. (Hull 2015, 291.)

Because the conditional probability $\mathbb{P}(v > u + y | v > u) = 1 - G_{\xi,\beta}(y)$ and the probability $\mathbb{P}(v > u) = 1 - F(u)$, the unconditional probability that $v > x$ is

$$\mathbb{P}(v > x) = [1 - F(u)][1 - G_{\xi,\beta}(x - u)], \qquad x > u$$

Given $N$ number of total observations, $1 - F(u)$ can be estimated with $\frac{N_u}{N}$. The unconditional probability becomes

$$\mathbb{P}(v > x) = \frac{N_u}{N}\left[1 + \xi\frac{x - u}{\beta}\right]^{-1/\xi}, \qquad x > u$$

VaR can now be calculated with

$$F(\mathrm{VaR}_\alpha) = 1 - \mathbb{P}(v > \mathrm{VaR}_\alpha) = 1 - \alpha$$

Solving this equation gives the VaR estimate

$$\mathrm{VaR}_\alpha = u + \frac{\beta}{\xi}\left\{\left[\frac{N}{N_u}\alpha\right]^{-\xi} - 1\right\}$$

An important note is to remember that the EVT models the right tail behaviour. To get the left tail one needs to change $v$ for $-v$. (Hull 2015, 291–292.)

As Chen and Lu (2012, 315) point out, the selection of the threshold value $u$ plays an important part in the estimation. Choosing high $u$ reduces bias because the approximation in (7) is only suitable for the tails. But a high value also means, that only few values $v$ can be observed where $v > u$. This leads to high variance in the estimator. However, choosing low value for $u$ generates the opposite: higher bias and lower variance.

McNeil and Frey (2000) introduce a filtered approach to EVT estimation of VaR. The vanilla EVT method doesn't reflect the volatility clustering of financial time series. To overcome this limitation they use GARCH model to compute estimates for the conditional volatility. Then they apply EVT to estimate the distribution of the residuals, which are roughly i.i.d., from the estimated GARCH model. After that the conditional return distribution can be constructed from the residual distribution and the estimates of conditional mean and volatility.

They assume that the past negative log-returns are realizations from a AR(1)-GARCH(1,1) process. Let $\{V_t\}$ be a time series of negative log-returns. The behaviour of $V$ is defined by

$$V_t = \mu_t + \sqrt{h_t}Z_t,$$

where $Z_t$ are i.i.d. innovations with zero mean and unit variance. The marginal distribution of $Z_t$ is $F_Z(z)$. The conditional variance and mean of the mean-adjusted series $\epsilon_t = V_t - \mu_t$ are

$$\begin{aligned} h_t &= \omega + \alpha\epsilon_{t-1}^2 + \beta h_{t-1} \\ \mu_t &= \phi V_{t-1}. \end{aligned}$$

Pseudo-maximum likelihood method is used to maximize the likelihood of the GARCH model. This method obtains the estimates for parameters $\hat{\theta} = (\hat{\phi}, \hat{\omega}, \hat{\alpha}, \hat{\beta})^T$. (McNeil and Frey 2000, 5–6.)

The innovations are computed from the estimated means and variances to get

$$z_t = \frac{v_t - \hat{\mu}_t}{\sqrt{\hat{h}_t}}$$

These innovations should be i.i.d. if the model is fitted properly. Now the next timestep predictions for conditional mean and variance are

$$\hat{\mu}_{t+1} = \hat{\phi} v_t$$
$$\hat{h}_{t+1} = \hat{\omega} + \hat{\alpha}\hat{\epsilon}_t^2 + \hat{\beta}\hat{h}_t$$

where $\hat{\epsilon}_t = v_t - \hat{\mu}_t$. (McNeil and Frey 2000, 6.)

After these calculations the EVT method is applied to the distribution $F_Z(z)$ of innovations. McNeil and Frey (2000, 8) fix the number of data in the tail to be $k$ where $k \ll N$. Now the $(k+1)$th order statistic is the threshold $u$. Given the ordered innovations $z_{(1)} \geq, z_{(2)}, \ldots, \geq z_{(N)}$, the generalized Pareto distribution is fitted to the excess over the threshold $(z_{(1)} - z_{(k+1)}, \ldots, z_{(k)} - z_{(k+1)})$. With this the tail estimator for $F_Z(z)$ is

$$\widehat{F_Z(z)} = 1 - \frac{k}{N}\left[1 + \hat{\xi}\frac{z - z_{(k+1)}}{\hat{\beta}}\right]^{-1/\xi}$$

Inverting this gives the quantile or VaR estimate

$$\widehat{\mathrm{VaR}}_{\alpha,k} = z_{(k+1)} + \frac{\hat{\beta}}{\hat{\xi}}\left\{\left[\frac{N}{k}\alpha\right]^{-\hat{\xi}} - 1\right\},$$

where $\alpha < \frac{k}{N}$.

# 5 METHODOLOGY

## 5.1 VaR backtesting

Chen and Lu (2012, 319) use three tests in their VaR backtesting framework and I have chosen to follow their approach. In the next sections I will present these three tests.

### 5.1.1 Conditional coverage test

Given a sample path $\{r_t\}_{t=1}^N$ of a portfolio return series $r_t$, the indicator variable $I_t$ is defined as

$$I_t = \begin{cases} 1, & \text{if} \quad r_t < \text{VaR}_{\alpha,t} \\ 0, & \text{if} \quad r_t \geq \text{VaR}_{\alpha,t}, \end{cases}$$

where $\text{VaR}_{\alpha,t}$ means the $\text{VaR}_\alpha$ at time $t$. The failure rate can now be defined as

$$\mathbb{E}[I_t] = \frac{n_e}{N},$$

where $n_e$ is the exceedances over the VaR. Now the unconditional coverage test is

$$H_0 : \mathbb{E}[I_t] = \alpha \qquad H_1 : \mathbb{E}[I_t] \neq \alpha$$

Because the sequence $I_t$ is Bernoulli distributed with the parameter $\alpha$, the likelihood under the null hypothesis is

$$L(\alpha; I_1, I_2, \ldots, I_N) = (1-\alpha)^{N-n_e}\alpha^{n_e}$$

and under the alternative

$$L\left(\frac{n_e}{N}; I_1, I_2, \ldots, I_N\right) = \left(1 - \frac{n_e}{N}\right)^{N-n_e}\left(\frac{n_e}{N}\right)^{n_e}$$

Using these likelihoods the unconditional coverage is the likelihood ratio test statistic

$$\text{LR}_{uc} = -2\log\left[\frac{(1-\alpha)^{N-n_e}\alpha^{n_e}}{\left(1-\frac{n_e}{N}\right)^{N-n_e}\left(\frac{n_e}{N}\right)^{n_e}}\right] \overset{\text{asy}}{\sim} \chi^2(1)$$

(Christoffersen 1998, 843–845; Chen and Lu 2012, 319–320.)

This unconditional coverage has a limitation of not taking into account the clustering of exceedances. Christoffersen (1998) extends the coverage by introducing a test for the independence and a joint test for independence and correct

coverage. The test for independence will be against a first-order Markov chain. Let's assume a binary first-order Markov chain $\{I_t\}$ with transition probability matrix

$$\mathbf{\Pi} = \begin{pmatrix} 1-\pi_{01} & \pi_{01} \\ 1-\pi_{11} & \pi_{11} \end{pmatrix},$$

where $\pi_{ij} = \mathbb{P}(I_t = i | I_{t-1} = j)$. For this chain the approximate likelihood function is

$$L(\mathbf{\Pi}; I_1, I_2, \ldots, I_N) = (1-\pi_{01})^{n_{00}} \pi_{01}^{n_{01}} (1-\pi_{11})^{n_{10}} \pi_{11}^{n_{11}}$$

The $n_{ij}$ is a number of observations where $i$ is followed by $j$.

Maximizing the log-likelihood function and then solving for parameters yields

$$\hat{\mathbf{\Pi}} = \begin{pmatrix} \frac{n_{00}}{n_{00}+n_{01}} & \frac{n_{01}}{n_{00}+n_{01}} \\ \frac{n_{10}}{n_{10}+n_{11}} & \frac{n_{11}}{n_{10}+n_{11}} \end{pmatrix}$$

If the sequence of indicator variables is independent the transition probability matrix is

$$\mathbf{\Pi} = \mathbf{\Pi}_\alpha = \begin{pmatrix} 1-\alpha & \alpha \\ 1-\alpha & \alpha \end{pmatrix}$$

Now the likelihood function for this null hypothesis is

$$L(\mathbf{\Pi}_\alpha; I_1, I_2, \ldots, I_N) = (1-\alpha)^{(n_{00}+n_{10})} \alpha^{(n_{01}+n_{11})}$$

The likelihood ratio test of independence is then

$$\mathrm{LR}_{ind} = -2 \log \left[ \frac{(1-\alpha)^{(n_{00}+n_{10})} \alpha^{(n_{01}+n_{11})}}{(1-\pi_{01})^{n_{00}} \pi_{01}^{n_{01}} (1-\pi_{11})^{n_{10}} \pi_{11}^{n_{11}}} \right] \overset{\text{asy}}{\sim} \chi^2(1)$$

(Christoffersen 1998, 845–847.)

The unconditional coverage and the independence tests can be combined to get the conditional coverage test statistic

$$\mathrm{LR}_{cc} = \mathrm{LR}_{uc} + \mathrm{LR}_{ind} \overset{\text{asy}}{\sim} \chi^2(2)$$

(Christoffersen 1998, 847.)

### 5.1.2 Dynamic quantile test

As Chen and Lu (2012, 321) note the conditional coverage test only tests the temporal dependency of order one. In addition the test doesn't measure the influence of any explanatory variables other than past exceedances. To address this Engle

and Manganelli (2004, 370) proposed a dynamic quantile test. Their test models the process of the hit function

$$H_t = I_t - \alpha = \begin{cases} 1-\alpha, & \text{if} \quad r_t < \text{VaR}_{\alpha,t} \\ -\alpha, & \text{else} \end{cases}$$

with a linear regression. This way the regression model is

$$H_t = \beta_0 + \sum_{j=1}^{p} \beta_j H_{t-j} + \sum_{k=1}^{K} \gamma_k g_k(z_t) + \epsilon_t,$$

where $g(\cdot)$ is a function of past exceedances and of variable $z_t$ and $\epsilon_t$ is an i.i.d. process with mean of zero.

If the VaR estimation method can provide accurate VaR estimates and there is no correlation between $p$ consecutive exceedances, the null hypothesis of the dynamic quantile test will be

$$H_0 : \zeta = (\beta_0, \beta_1, \ldots, \beta_p, \gamma_0, \gamma_1, \ldots, \gamma_K)^\top = 0$$

The dynamic quantile test statistic can now be written as

$$\text{DQ} = \frac{\hat{\zeta}^\top \mathbf{\Sigma}^\top \mathbf{\Sigma} \hat{\zeta}}{\alpha(1-\alpha)}$$

where $\Sigma$ is the covariance matrix of the regression model. Chen and Lu (2012, 322) choose the values $p = 4$ and $K = 1$ and the function $g(z_t) = \widehat{\text{VaR}}_{\alpha,t}$.

### 5.1.3 Ljung-Box test

The third test that Chen and Lu (2012) suggest is the Ljung-Box test. The Ljung-Box test tests the absence of autocorrelation in a time series. Given the same hit function process as in previous section, $\{H_t\}$, the null hypothesis is

$$H_0 : \rho_1(H_t) = \cdots = \rho_p(H_t) = 0,$$

where $\rho_j$ is the $j$th autocorrelation of the process. The Ljung-Box test statistic is now

$$\text{LB} = N(N+2) \sum_{j=1}^{p} \frac{\hat{\rho}_j^2}{N-j} \overset{\text{asy}}{\sim} \chi^2(p),$$

where $p$ is the order of the autocorrelation for $\{H_t\}$. Chen and Lu (2012, 322) recommend $p = \log(N)$. The Ljung-Box test overcomes the limitation of temporal dependency of order one of the conditional coverage test.

## 5.2 VaR estimators

### 5.2.1 Baseline

As a baseline for the VaR backtesting I will use the historical simulation (HS) method introduced in section 4.1. This choice is due to the simple nature of the method and robust performance. In addition to that, the HS method is widely adopted in the industry. Pérignon and Smith (2010, 367) discovered that 73% of banks that disclosed their VaR estimation method used historical simulation. It is out of the scope of this thesis to compare multiple different VaR estimators. For interested reader, Chen and Lu (2012) backtests several different estimators.

### 5.2.2 GAN model

From recent studies applying the GAN model to generate realistic financial time series I chose the simple MLP architecture for both the generator and discriminator. Takahashi et al. (2019, 8) show the effectiveness of a few different architectures. They conclude that the MLP based model can replicate all the stylized facts of financial time series they measured. Their MLP model has four hidden layers with a hyperbolic tangent activation function. As the training of GANs can be sensitive to slight modifications in the model parameters, it is unfortunate that they don't provide the full details of the model like the width of the hidden layers. Fu et al. (2019) support the choice of MLP for the generator and discriminator. They build the model from three hidden layers with a varying width. The width is always over 100, but they don't specify the exact numbers.

Other architectures mentioned in the literature were also considered. de Meer Pardo (2019) adopts convolutional neural networks (CNN) as the generator and discriminator. He also alters the training algorithm. The model is trained with WGAN-GP (Gulrajani, Ahmed, Arjovsky, Dumoulin and Courville, 2017) objective. The cost function in the WGAN-GP setup is

$$L = \mathbb{E}[D(G(z))] - \mathbb{E}[D(x)] + \lambda \mathbb{E}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2],$$

where $z$ is the input noise variable and $x$ comes from the training data. The distribution of $\hat{x}$ is defined by uniformly sampling from straight lines between pairs of points from distributions $p_{data}$ and $p_G$. The gradient penalty coefficient $\lambda$ determines how much to penalize the gradient norm. The point of the WGAN-GP objective is to stabilize the training process.

In my experiments the results with CNNs and WGAN-GPs weren't reasonable. The CNN architecture was also considerably larger than the MLP architecture. With limited computational resources, I narrowed my focus to the MLP model. As a cost function I chose the same function as proposed in the original GAN setup. (Goodfellow et al., 2014). In figures 6 and 7 the chosen MLP architectures are presented.

The generator has as an input a random noise vector of length 50. The noise vector is sampled randomly from the standard normal distribution. The input vector is then fed to three hidden blocks consisting of fully connected linear layer, dropout and LeakyReLU activation. The width of the block is 100 neurons. Dropout probability is 0.8. The LeakyReLU activation was first proposed by Maas, Hannun and Ng (2013) and is defined as

$$\text{LeakyReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ bx, & \text{if } x < 0 \end{cases}$$

where $b$ is some small slope coefficient. In this thesis $b = 0.2$. The output after applying the three hidden blocks is finally fed to fully connected linear layer with width ranging of 500 or 1000. After this the hyperbolic tangent function is used to get the final generated fake returns.

The discriminator takes as an input the generated or real returns of length 500 or 1000. These are transformed with two fully connected hidden layers followed by LeakyReLU activation to the input for the last layer. The first hidden layer has 100 neurons and the second 50 neurons. The final layer outputs the binary classification probability going through the sigmoid function.

Both components utilize the Adam optimizer presented previously in algorithm 4. Learning rate for the generator is 0.0002 and for the discriminator 0.00001. The decay rate $\beta_1$ for the first moment estimate is 0.5 for the generator and 0.1 for the discriminator. The second moment decay rate $\beta_2$ is 0.999 for both networks.

Three parameter retrieval setups during training are used. All the setups are trained for 2000 epochs. First is the minimum kurtosis score method. In this method, the parameters from the epoch that minimizes the norm between the kurtosis of the 10000 training log returns and mean kurtosis of generator generated

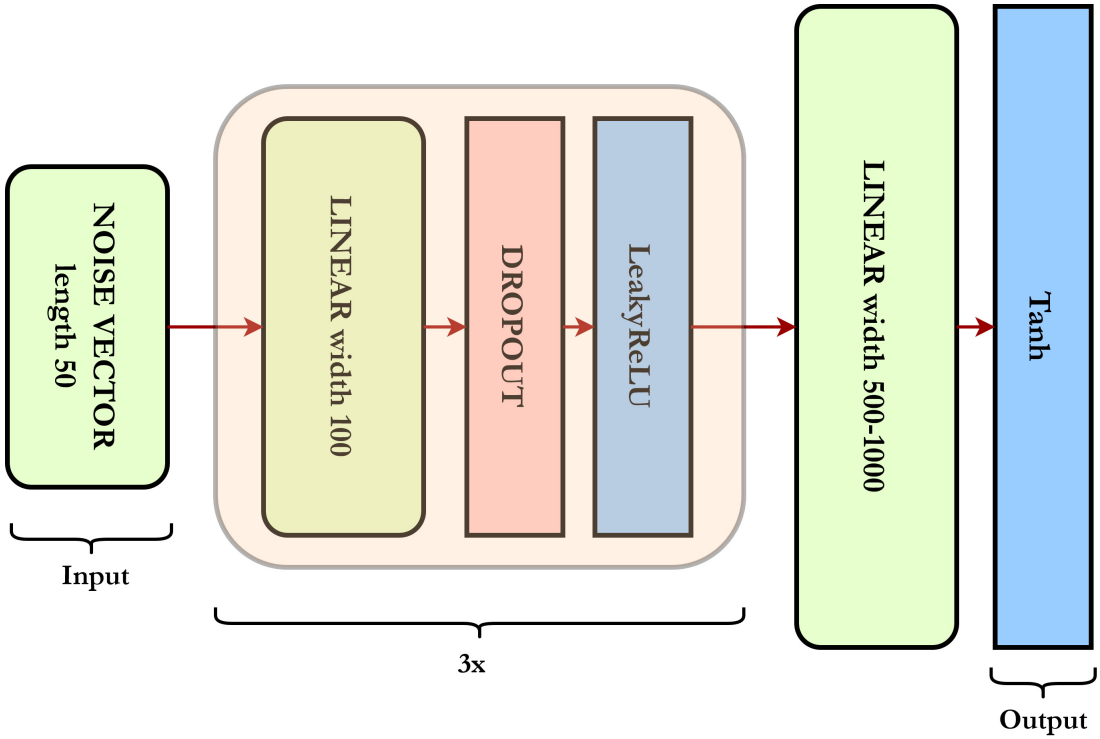


Figure 6: GAN generator architecture.

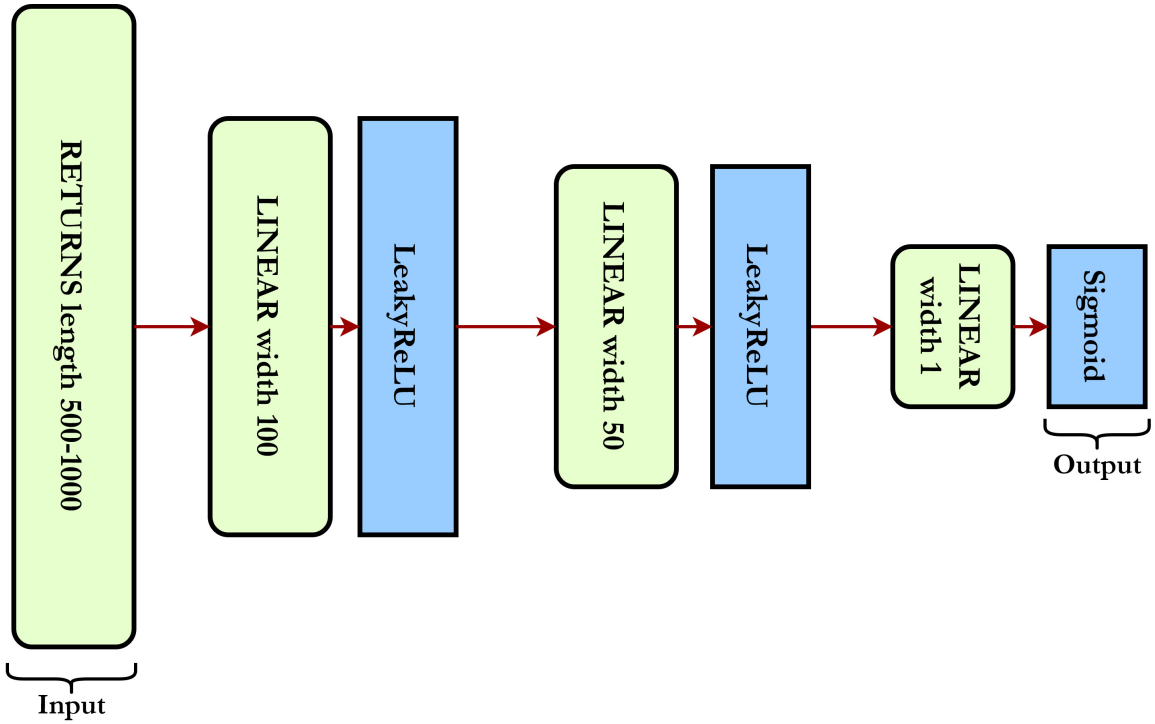


Figure 7: GAN discriminator architecture.

log returns, are saved. More formally

$$\text{KurtScore}(\boldsymbol{\theta}_i) = \left\| \text{kurt}(\boldsymbol{x}_{\text{train}}) - \frac{1}{n}\sum_{k=1}^{n} \text{kurt}(G_{\boldsymbol{\theta}_i}(\boldsymbol{z})) \right\|_2$$
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}_i} \text{KurtScore}(\boldsymbol{\theta}_i)$$

where $\boldsymbol{\theta}$ are the model parameters, $i$ is the epoch and $G_{\boldsymbol{\theta}_i}$ is the generator with parameters at epoch $i$. The kurtosis of generated log returns are averaged over $n$ runs. In this thesis $n = 100$. The minimum kurtosis score training is conducted with generator output size of 500 and 1000 days.

The other method for optimal parameter search is minimum autocorrelation function score (ACFScore). The idea is similar to the minimum kurtosis score. Only the kurtosis calculation is replaced with autocorrelation function. The lags used in the ACF is 100. The minimum ACFScore didn't provide as good results as KurtScore so I only took the 1000 day variant into further investigation.

These two aforementioned approaches to optimal parameter search were based on the notion, that the GAN training didn't converge. These methods were then hypothesized to produce best results in the generation of realistic financial asset return distributions. Further optimization of the parameter search method is out of the scope of this thesis and left to further research.

# 6 EXPERIMENTS

Because the VaR estimation problem can casted as a future portfolio returns distribution estimation problem, I will first demonstrate the performance of the GAN model to reproduce the stylized facts of stock index returns. After this the VaR backtesting procedure is conducted. The experiments were programmed with Python. The code is available at https://github.com/akeele/GANVaR.

## 6.1 Data

Data is gathered from Thomson Reuters Eikon database. I chose S&P 500 price index for my experiments. I calculated log returns for daily prices between 1.1.1970 - 29.11.2017. This choice was made to have a dataset consisting of 12500 days. The choice of 12500 days was arbitrary. It was based on the simple heuristic that neural networks need a lot of data and the test set should also be suffieciently large to calculate reliable statistics. The dataset is divided to training and test set. Training set consists of the first 10000 observations. The test set is the following 2500 observations. In the fitting of the GAN discriminator, the training set returns are chunked into 500 or 1000 days long chunks with rolling step of 50 days. As more data becomes "available", when the testing is conducted daily, every 50 days the GAN model is trained from scratch using the previous 10000 days as the training data.

As my portfolio consists of only one asset, namely S&P 500 index, the results in this thesis cannot be applied trivially to financial institution's vast portfolio. Banks and other institutions have normally thousands of assets in their portfolio, including stocks, bonds, commodities and derivatives. Especially the very different nature of the bond and derivative instruments compared to stocks needs further investigation and refinement of the methods used in this thesis.

Despite all of the above, I believe that using only stock index data provides enough information to judge the capabilities of the GAN model used in this thesis.

## 6.2 Results

### 6.2.1 Replication of statistical features

In chapter 3 I introduced common statistical properties of financial time series. Three main properties were presented in more detail, namely heavy tails, autocorrelation and volatility clustering. Now I will demonstrate how well the GAN model can replicate these three stylized facts of asset returns.

Producing heavy tailed distributions are typically done with some fitted parametric distribution. Few of these distributions were presented in section 3.1. In figure 8 the GAN produced distributions are presented. From the plot it is easy to see that none of the GAN models could replicate the heavy tails properly. When measuring the fit with kurtosis, the min kurtosis GAN model with output size of 500 performs the best. The min kurtosis means that the parameters from the generator, that minimizes the euclidean distance of kurtoses between real and fake returns, are used. This is due to the unstable training process, where the loss function does not converge smoothly. At first I tried to train the model longer to see if it converges but this wasn't the case. The cause of this unstable training process is probably the nonconvexity of the cost function and the sensitive equilibrium between the generator and the discriminator. Barnett (2018, 12–14) has a good summary of the main problems related to GAN convergence issues.

Figure 9 shows the autocorrelations of the different models. All of the models seem to produce reasonable autocorrelations. The autocorrelations stay mostly within the range $[-0.05, 0.05]$. All of the GAN models have low autocorrelations across lags, just as real returns do. It seems that producing the lack of autocorrelation is fairly easy for the GAN models.

Last of the three main stylized facts is volatility clustering. This is usually measured by computing the autocorrelations of a nonlinear transformation of log returns. I used absolute log returns as the nonlinear transformation. Figure 10 depicts the results for the autocorrelations of absolute log returns. There can be seen an interesting result. The surprising thing is that the min ACF model isn't better. This was the model that minimized the euclidean distance between the absolute autocorrelations of generated and real returns. It is hard to say which of models is the best. The min ACF model has the level of autocorrelation close to real returns at the shorter lags. However the min kurtosis with output size 500 has more downward sloping shape and is close to the real returns at the longer lags. When considering all of the main stylized facts, the min kurtosis with output size

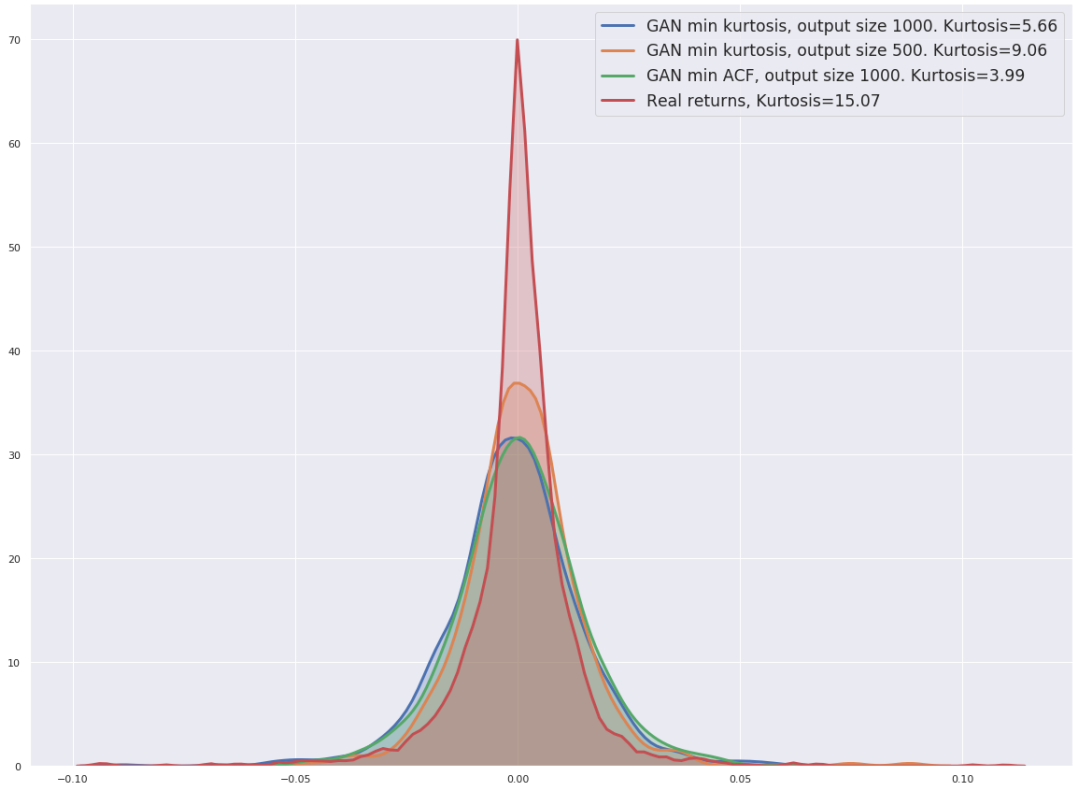


Figure 8: Kernel density plots of fake and real log returns

500 performs the best. It performs as good as the others with autocorrelation and volatility clustering and significantly outperforms them in producing heavy tailed distribution.

### 6.2.2 VaR backtesting

Using the three backtesting procedures introduced in section 5.1, I will demonstrate the performance of the GAN models compared to a simple baseline. The simple baseline is historical simulation with look back window of 250 previous trading days. The choice of 250 trading days was made because it corresponds to one calendar year and performed better than other variations. It was chosen because of its interpretability, fast computation and wide adoption in the financial industry. Results of the backtests are in table 1.

Looking at the Christoffersen's test first, the HS and the GAN model fail to reject the null hypothesis, which is that the sequence of VaR violations is independent. Both methods reject the dynamic quantile null hypothesis. This means that the weights in the dynamic quantile regression model significantly differ from

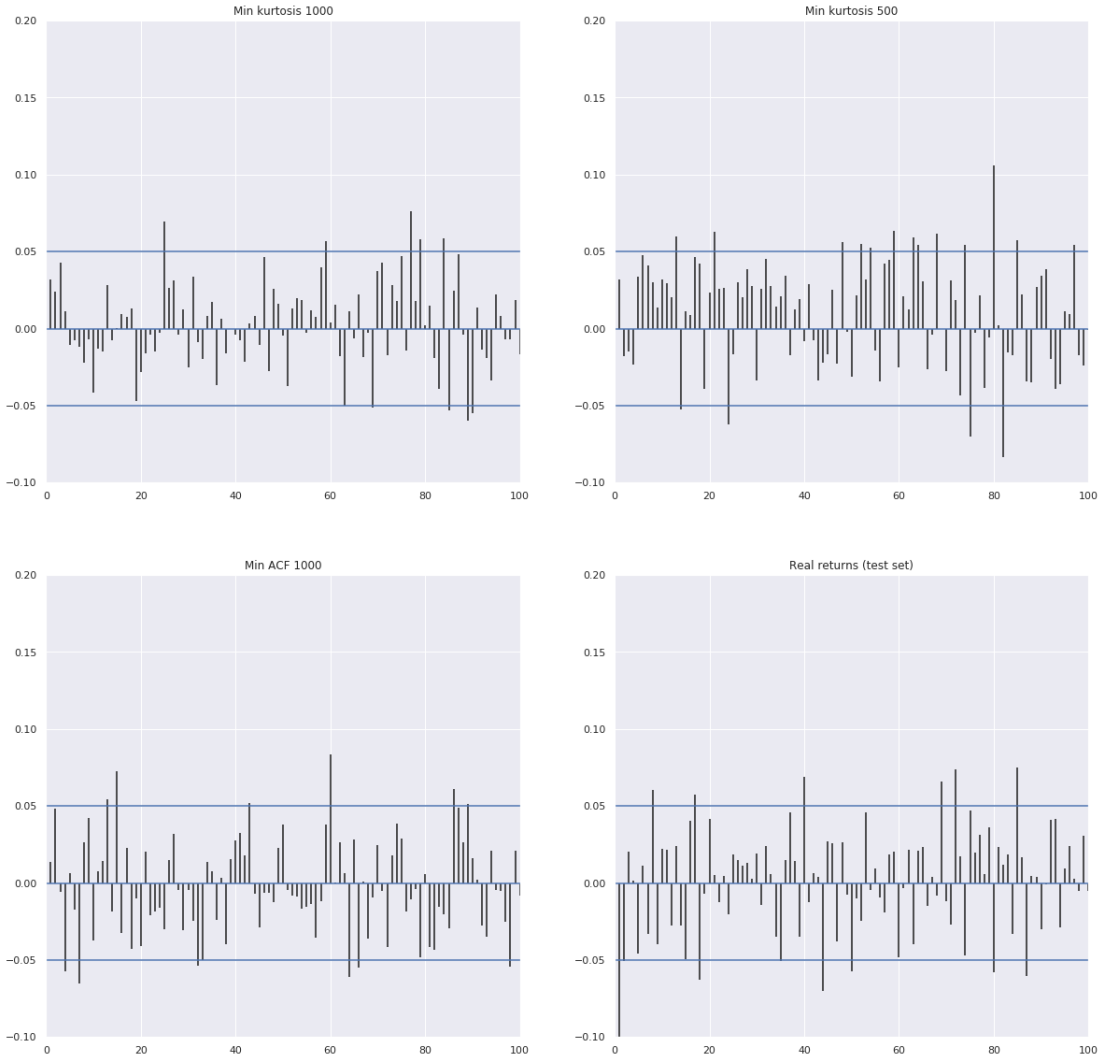


Figure 9: Autocorrelations of GAN generated and real log returns.

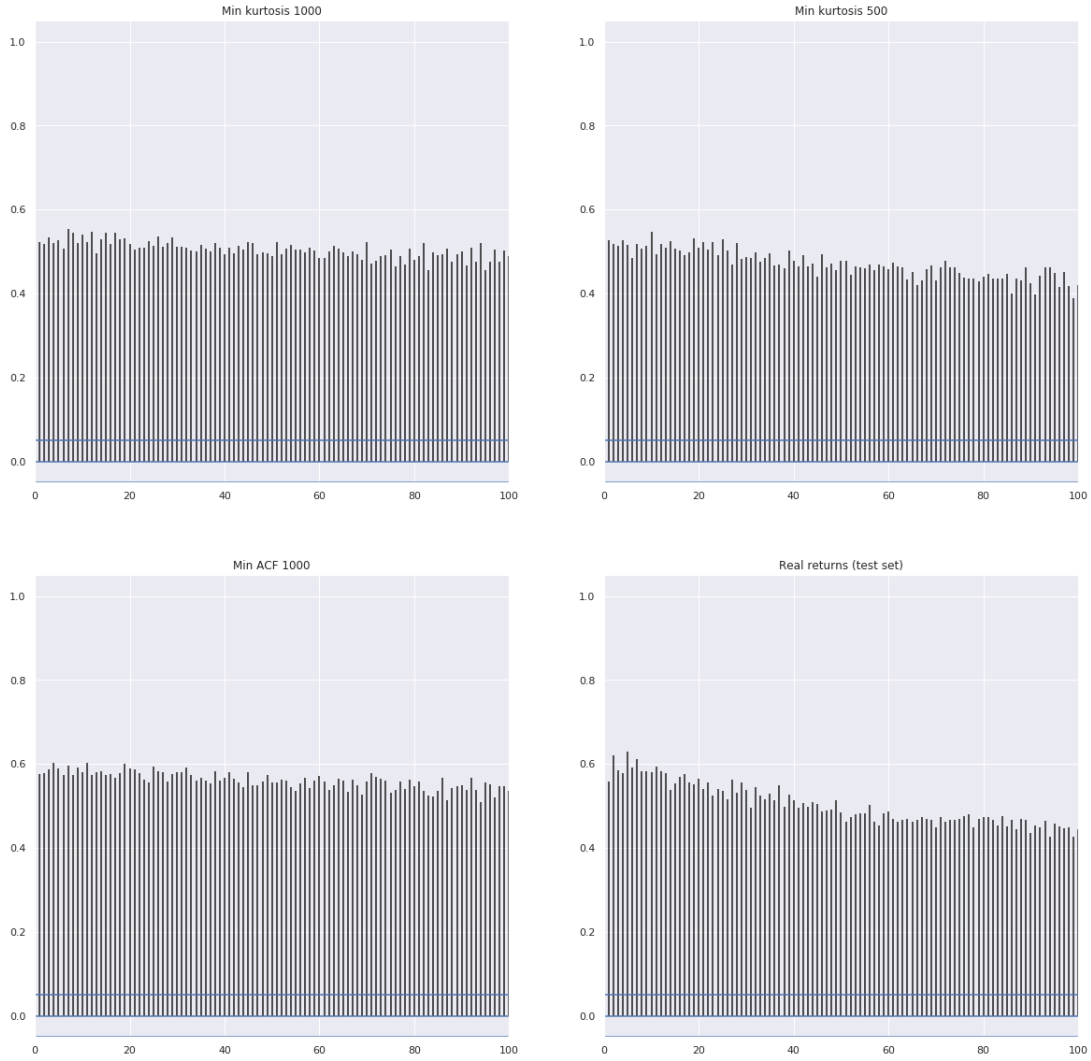


Figure 10: Autocorrelations of absolute log returns.

| | HS 250 days | GAN min kurtosis 500 |
|---|---|---|
| **Christoffersen** | 0.0119 (8.86) | 0.011 (9.03) |
| **Dynamic quantile** | <0.001 (129.58) | <0.001 (181.09) |
| **Ljung-Box** | <0.001 (194.68) | <0.001 (374.90) |

Table 1: p-values of VaR backtests. Test statistics in parentheses. Significance threshold of 0.01 is used.

zero. The same occurs in the Ljung-Box test. The autocorrelations of the violation sequence differ from zero. According to these backtests, both models fare poorly. For example Chen and Lu (2012, 329-330) report robust performance of filtered HS and EVT methods using the same tests. They apply the methods to the DAX, Dow Jones Industry Average 30 and Singapore Straight Time Index over a period from 18.8.2003-31.7.2009.

I think that the backtests didn't provide enough information to decide if the GAN model performs better than the baseline. Looking at the graph of the VaR estimates plotted against daily log returns gives more insight of the performance of the models. This graph is depicted in figure 11. It is clear from the graph, that the HS method is more robust to changes in the underlying return process. This might be due the fact of using the same GAN model for 50 consecutive days before retraining. But taking into account that still doesn't explain the fairly narrow range of VaR estimates for the GAN model. There is much more wider range in the HS estimates. The complexity of the GAN model together with poor performance in the backtests and inability to respond to recent market regime changes makes it unsuitable for VaR estimation. It needs to be further developed and researched if robust performance compared to existing simpler methods is desired.

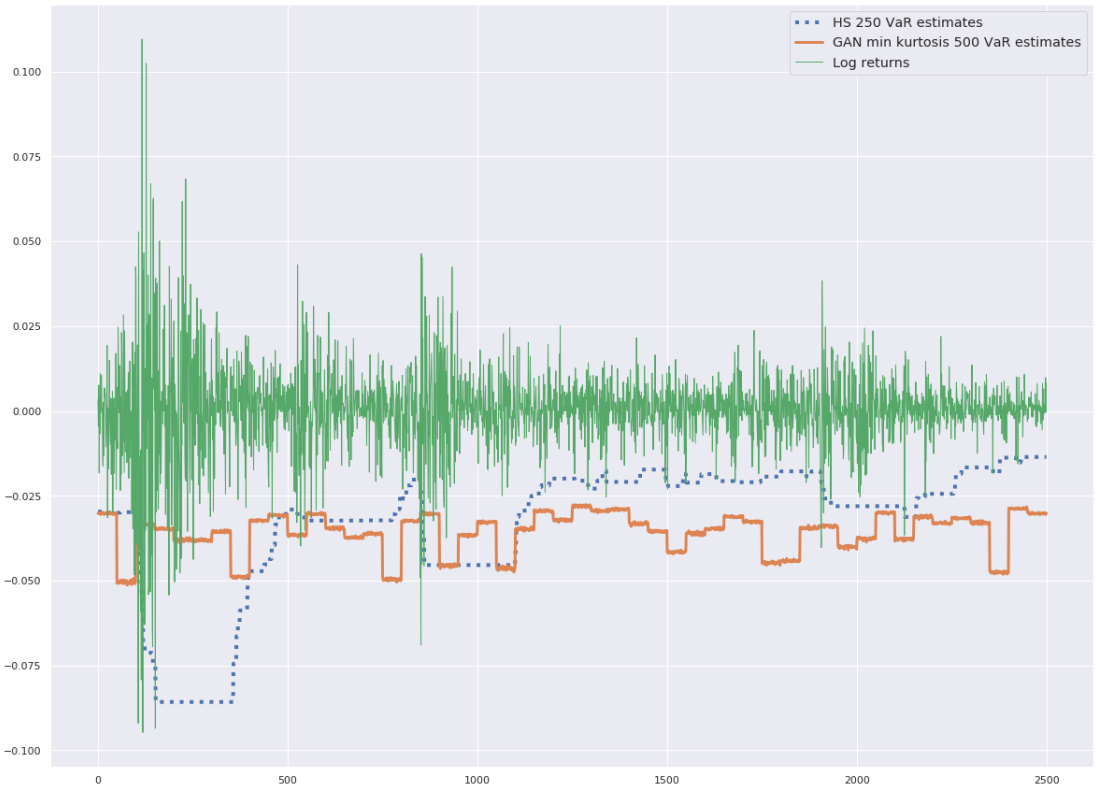


Figure 11: VaR estimates of S&P500 log returns.

# 7 CONCLUSION

## 7.1 Thoughts on results

The GAN model didn't provide more accurate VaR estimates than the baseline HS method. Although there have been promising results from applying it to financial time series, the process of achieving as good results on your own is rocky and slippery road. It is known from the literature that the successful training of GANs can be a serious act of hyperparameter juggling. (see e.g. Lucic, Kurach, Michalski, Bousquet and Gelly, 2018). Research has been done on understanding the inner behaviour of GAN training. (Barnett 2018;Li, Madry, Peebles and Schmidt 2017).

Due to time constraints, I was not able to test extensively different architectures for the generator and discriminator networks. Therefore it is still open question whether the simple feedforward neural network provides the best estimates. Previous research has shown good results with feedforward neural networks (Takahashi et al., 2019), convolutional neural networks (de Meer Pardo, 2019) and temporal convolutional networks (Wiese et al., 2019). Intuitively recurrent neural networks would produce the best results, because they are designed to carry information from previous returns. Recurrent neural networks are however computationally expensive. Zhang et al. (2019) achieve promising results by using LSTM network for stock market prediction.

VaR estimates are important for financial institutions when positioning themselves in the market and reporting to regulators. They need explainable models. The estimates are especially crucial in market crises which can affect the whole financial system. There is thus demand for a robust, understandable and responsive method for VaR estimation. Neural networks are seen as black-box models because analyzing the weights of the network doesn't provide information about the structure of the function it is approximating. The black-box nature makes it very hard to guess what the model will predict in different market regimes. This can lead to catastrophic situations in market turmoil. This thesis shows that more work is needed for GAN based solutions. This concerns both the accuracy of the model as well as the explainability, although explainability wasn't studied.

## 7.2 Further research

GAN based and other data-driven nonparametric methodologies look promising. They rely on past returns as the true data generating process. This is in contrast to parametric models which make assumptions about the distribution of future returns. However these data-driven models need a lot of data and careful fitting to get accurate estimates. Here are few topics to consider when doing further research:

- shorter than daily time intervals
- training same GAN model with multiple assets
- extensive hyperparameter searching
- applying GAN on portfolio wide estimation

More granular level of data means that there are more data points between two timestamps. This can help with the data hungry neural networks inside GAN. This also enables the model to utilize more recent data, because it doesn't have to use decades old returns.

Training with multiple assets follows the previous in that it allows more data to the model. This method can also lead to a more generalizable model. Using for example returns of individual constituents of a stock index as the training data instead of only the index, might produce a model that "knows" more about the real distribution of the index as a whole.

Searching the best hyperparameters for the GAN model is on avenue to check, but it needs sufficient computational resources. This approach is also questionable since it can lead to significant overfitting. It introduces data mining bias, which arises due to the extensive search for parameters that produce good results.

This thesis presented a simple setting with one asset portfolio. To aid financial institutions in their risk management, the GAN model must be able to estimate VaR for a portfolio consisting of thousands of different assets. This can mean multiple GAN models for separate asset classes or more detailed models. These are then combined to calculate the portfolio wide estimate.

# REFERENCES

Artzner, P., Delbaen, F., Eber, J.-M. – Heath, D. (1999) Coherent measures of risk. *Mathematical Finance*, vol. 9 (3), 203–228.

Barndorff-Nielsen, O. E. (1997) Normal inverse gaussian distributions and stochastic volatility modelling. *Scandinavian Journal of Statistics*, vol. 24 (1), 1–13.

Barnett, S. A. (2018) Convergence problems with generative adversarial networks (GANs). *arXiv preprint.*

Barone-Adesi, G., Giannopoulos, K. – Vosper, L. (1997) VaR without correlations for nonlinear portfolios. *Journal of Futures Markets.*

Baumol, W. J. (1963) An expected gain-confidence limit criterion for portfolio selection. *Management Science*, vol. 10 (1), 174–182.

Blattberg, R. C. – Gonedes, N. J. (1974) A comparison of the stable and student distributions as statistical models for stock prices. *The Journal of Business*, vol. 47 (2), 244–280.

Bollerslev, T. (1986) Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, vol. 31 (3), 307–327.

Boudoukh, J., Richardson, M. – Whitelaw, R. (1998) The best of both worlds. *Risk*, vol. 11 (5), 64–67.

Chen, Y. – Lu, J. (2012) *Value at Risk Estimation*, 307–333. Springer Berlin Heidelberg, Berlin, Heidelberg.

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J. – Dahl, G. E. (2019) On empirical comparisons of optimizers for deep learning. *arXiv preprint.*

Christoffersen, P. F. (1998) Evaluating interval forecasts. *International Economic Review*, vol. 39 (4), 841–862.

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

Duchi, J., Hazan, E. – Singer, Y. (2011) Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, vol. 12, 2121–2159.

Eberlein, E., Keller, U. – Prause, K. (1998) New insights into smile, mispricing, and value at risk: The hyperbolic model. *The Journal of Business*, vol. 71 (3), 371–405.

Engle, R. F. (1982) Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, vol. 50 (4), 987–1007.

Engle, R. F. – Manganelli, S. (2004) CAViaR. *Journal of Business & Economic Statistics*, vol. 22 (4), 367–381.

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

Goodfellow, I., Bengio, Y. – Courville, A. (2016) *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. – Bengio, Y. (2014) Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, eds. Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence – K. Q. Weinberger, 2672–2680, Curran Associates, Inc.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V. – Courville, A. (2017) Improved training of asserstein GANs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, 5769–5779, Curran Associates Inc., Red Hook, NY, USA.

Hinton, G., Srivastava, N. – Swersky, K. (2012) *Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural networks for machine learning.

Hornik, K., Stinchcombe, M. – White, H. (1989) Multilayer feedforward networks are universal approximators. *Neural Networks*, vol. 2 (5), 359–366.

Hull, J. C. (2015) *Risk Management and Financial Institutions*. John Wiley Sons, 4 edn.

Kingma, D. – Ba, J. (2014) Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Koshiyama, A. S., Firoozye, N. – Treleaven, P. C. (2019) Generative adversarial networks for financial trading strategies fine-tuning and combination. *arXiv preprint*.

Li, J., Madry, A., Peebles, J. – Schmidt, L. (2017) On the limitations of first-order approximation in GAN dynamics. *arXiv preprint*.

Lucic, M., Kurach, K., Michalski, M., Bousquet, O. – Gelly, S. (2018) Are GANs created equal? a large-scale study. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 698–707, Curran Associates Inc., Red Hook, NY, USA.

Maas, A. L., Hannun, A. Y. – Ng, A. Y. (2013) Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 30, 3.

Mandelbrot, B. (1963) The Variation of Certain Speculative Prices. *The Journal of Business*, vol. 36, 394–394.

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*.

Marti, G. (2019) CorrGAN: Sampling realistic financial correlation matrices using generative adversarial networks. *arXiv preprint*.

McNeil, A. J. – Frey, R. (2000) Estimation of tail-related risk measures for heteroscedastic financial time series: an extreme value approach. *Journal of Empirical Finance*, vol. 7 (3), 271–300, special issue on Risk Management.

de Meer Pardo, F. (2019) *Enriching Financial Datasets with Generative Adversarial Networks*. Master's thesis, Delft University of Technology.

Minsky, M. – Papert, S. (1969) *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA.

Mirza, M. – Osindero, S. (2014) Conditional generative adversarial nets. *arXiv preprint*.

Mustafa, M., Bard, D., Bhimji, W., Lukić, Z., Al-Rfou, R. – Kratochvil, J. M. (2019) CosmoGAN: creating high-fidelity weak lensing convergence maps using generative adversarial networks. *Computational Astrophysics and Cosmology*, vol. 6.

Nadarajah, S. – Chan, S. (2016) *Estimation Methods for Value at Risk*, chap. 12, 283–356. John Wiley  Sons, Ltd.

Polyak, B. (1964) Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, vol. 4, 1–17.

Pritsker, M. (2001) The hidden dangers of historical simulation. *FEDS Discussion Paper No. 2001-27.*

Pérignon, C. – Smith, D. R. (2010) The level and quality of value-at-risk disclosure by commercial banks. *Journal of Banking  Finance*, vol. 34 (2), 362 – 377.

Radford, A., Metz, L. – Chintala, S. (2015) Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint.*

Rosenblatt, F. (1958) The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, vol. 65 (6), 386–408.

Rumelhart, D., Hinton, G. – Williams, R. (1986) Learning representations by back-propagating errors. *Nature*, vol. 323, 533–536.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. – Salakhutdinov, R. (2014) Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, vol. 15, 1929–1958.

Swirszcz, G., Czarnecki, W. M. – Pascanu, R. (2016) Local minima in training of neural networks. *arXiv preprint.*

Takahashi, S., Chen, Y. – Tanaka-Ishii, K. (2019) Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, vol. 527, 121261.

Uricar, M., Krizek, P., Hurych, D., Sobh, I., Yogamani, S. – Denny, P. (2019) Yes we GAN: applying adversial techniques for autonomous driving. *arXiv preprint.*

Werbos, P. (1974) *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences.* Ph.D. thesis, Harvard University.

Wiese, M., Knobloch, R., Korn, R. – Kretschmer, P. (2019) Quant GANs: deep generation of financial time series. *arXiv preprint.*

Zhang, K., Zhong, G., Dong, J., Wang, S. – Wang, Y. (2019) Stock market prediction based on generative adversarial network. *Procedia Computer Science*, vol. 147, 400–406, 2018 International Conference on Identification, Information and Knowledge in the Internet of Things.

Zhou, X., Pan, Z., Hu, G., Tang, S. – Zhao, C. (2018) Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*.