CHALMERS
UNIVERSITY OF TECHNOLOGY

Evaluation of a bidirectional GAN on high dimensional data

With applications for financial data simulation

Master's thesis in Engineering Mathematics and Computational Science

MARCUS SAJLAND

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Evaluation of a bidirectional GAN on high dimensional data

With applications for financial data simulation

MARCUS SAJLAND

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
Gothenburg, Sweden 2022

Evaluation of a bidirectional GAN on high dimensional data
With applications for financial data simulation
MARCUS SAJLAND

Supervisor: Richard Henricsson, Svenska Handelsbanken AB
Examiner: Patrik Albin, Department of Mathematical Sciences, Chalmers University of Technology

Master's Thesis 2022
Department of Mathematical Sciences
Division of Applied Mathematics and Statistics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Evaluation of a bidirectional GAN on high dimensional data
With applications for financial data simulation
MARCUS SAJLAND
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

In statistics and machine learning it is well known that as the dimensionality of a space increases, an exponentially greater amount of data is necessary to accurately analyze it. This is a problem currently faced by Svenska Handelsbanken AB. As they aim to simulate future markets, they require methods of estimating densities of historical markets in order to generate new data points on which to produce the simulations. This thesis investigated the ability of a novel machine learning algorithm to generate data that manages to capture tail dependencies that common statistical models fail to do. The performance was first measured on a simulated data set where the means and variances were already known, followed by measuring the performance on real market data. The results on the market data made it clear that the algorithm was not capable of capturing tail dependencies as desired as it generally generated points of much smaller variance than the original data. However, the results on the simulated data implied that on a data set of roughly only ten times the size, which in machine learning is not extremely large, the algorithm would likely generate data according to the original distribution much more consistently.

Keywords: machine learning, deep learning, generative adversarial networks, curse of dimensionality, manifold learning, financial time series.

Acknowledgements

I would like to thank my supervisor, Richard Henricsson, at Svenska Handelsbanken AB for the arragement of this project and for his engagement throughout the course of it. I would also like to thank Per Sundin of Svenska Handelsbanken AB for his assistance and for introducing me to the work he does. Lastly, I would like to thank Qiao Liu from Stanford University, the creator of the Roundtrip algorithm, for his willingness to answer questions over email and provide me with tips for my specific use of the algorithm.

Marucs Sajland, Gothenburg, August 2022

Contents

List of Figures

List of Tables

1

Introduction

This section aims to provide the reader with an understanding of the purpose of this project, a brief background of the field of machine learning, and the aim and limitations of the project. Finally, some of the related research is explained and the ethical considerations are taken into account.

1.1 Background

An effective way to investigate how a model, program, or technology of some kind would perform under certain scenarios is to run simulations. Of course, this implies understanding what each of these scenarios actually are and the mechanics behind them. Sometimes they are well understood, say, when investigating how a simple pendulum swings, as gravity, friction, and air resistance are quiet easily incorporated into a model. In other cases, however, it is not so easy. Financial markets are one example which do not obey any natural laws and which are dependent on more factors than one could reasonably account for in a model. One of the most obvious ways to simulate financial markets is to look at how they have behaved in the past and to use this to build an understanding of the mechanisms driving the markets, and to then base simulations on this. Given that there exist decades worth of data, it is not too difficult to provide a relatively accurate estimate of where the markets will be tomorrow or next week: probably somewhere similar to today, when looking at the big picture. But to simulate the markets twenty years into the future is a much more difficult task, and one that is very much relevant, especially for Svenska Handelsbanken AB.

The bank offers customers the option to invest money with the bank's proprietary investment models, which seek relatively safe long term returns. In order to get an understanding of risk levels and possible outcomes at the time of investment, the bank requires simulations of the financial markets which are as accurate as possible. As these models are aimed at the long term, it is necessary to simulate the markets a couple decades into the future with the help of a couple decades worth of data, which, as mentioned already, is an enormously difficult task. Up until now, Handelsbanken has made use of statistical models to create an understanding of how and why markets behave the way they do, meaning that they have used traditional methods to estimate the densities of financial time series data.

1.2 Machine learning and generative adversarial networks

Artificial intelligence, or AI, as a field of research has been around for many decades, after John McCarthy proposed for it to be the topic of a summer research project at Dartmouth University in 1955 [1]. In the decades that followed, the field has had several ups and down. Periods of intense research have been followed by so called 'AI winters' on two occasions, when lack of confidence in the field dried up funding for research, but during the past two decades AI research has boomed, and especially the field of machine learning [2]. As opposed to hard-coding rules for the machine to follow, such as the 1989 Cyc project [3], machine learning utilizes raw data to learn for itself. Relatively simple algorithms, such as logistic regression, are included in the field of machine learning, but in recent years much effort has been put into the so-called area of deep learning. This category of machine learning often utilizes neural networks, which loosely resemble networks of real neurons, to learn at a much deeper level, which is aided by the increased availability of data [4].
In 2014 Ian Goodfellow et al. proposed Generative Adversarial Networks, or GANs, a form of deep learning that generates data with similar characteristics as the input data [5]. An example of its use would be using pictures of faces as input data which would result in the algorithm generating pictures of faces that look realistic but are not actually real. The way it works is that the network comprises two neural networks that compete against each other. Given some real input data, such as an image, one of the networks, called the generator, aims to create images similar to the input data from random noise. Meanwhile, the other network, called the discriminator, aims to tell the real and fake data apart. At first this is easy for the discriminator as the generated fake images are very unrealistic, but as the two networks compete against each other they both get better until eventually the generator creates images realistic enough to trick the discriminator. GANs have become very popular and people have attempted to expand upon them by giving them the power to actually estimate densities, which is not possible with the standard GAN [6], [7]. Building on this concept, Qiao Liu et al. proposed Roundtrip [8] in 2021 which combines two GANs, in the form of a bidirectional GAN, and has demonstrated even better performance. While these types of generative networks are often used to produce images of some sort, they can also be used for other data such as financial time series data. This essentially means that by feeding a network with this type of market data it can generate more market data.

1.3 The curse of dimensionality

A common problem in density estimation is what is referred to as the curse of dimensionality. This comes from the fact that as the dimensionality of a data set increases, an exponentially greater amount of data is needed to accurately analyze it. This happens as the available data becomes more and more sparse so enormous amounts of data are needed to estimate densities of high dimensional data. Roundtrip potentially provides a way to better tackle this problem which could allow for much

more accurate correlations to be found among financial time series [8].

1.4 Problem setting

The aim of this thesis is thus to investigate whether or not the Roundtrip algorithm can be used to generate realistic market data that captures correlations between markets that the statistical methods currently used by the bank fail to capture.

1.5 Limitations and assumptions

This project is solely be concerned with the generation and simulation of the financial time series data. It is not concerned with how this data is then used in the various models. It does also not delve deeply into how the bank currently produces simulated data, although the project's scope does include comparing the results of the methodologies with the hope that the data produced from this project is better. As this project heavily uses machine learning, it is important to set specific limitations for the neural networks as well. The Roundtrip algorithm was implemented as well as possible but to a reasonable extent, meaning that rigorous examination of specifics such as different cost functions and other components was not done. This thesis makes use of the available Roundtrip source code as it is efficiently implemented already, although it was altered slightly. When it comes to code pertaining to the financial industry it is common that there are stringent requirements on speed. This is, however, not the case with this project. The bank performs the simulations every couple of months so whether it takes a couple of minutes or a couple of hours to execute makes no difference.
The Roundtrip algorithm is capable of producing density estimates as demonstrated in the original paper, but this is done implicitly, meaning that it is not possible to extract a probability density function from which to generate data later on. Instead, data is generated by feeding noise to the fully trained generator, in a similar manner to how images would be created.
One major assumption and simplification that deserves mentioning is that Roundtrip was trained on time series data which is assumed to be stationary, meaning the data is assumed to not depend on time. While this is a large simplification, it is sufficient to gain an understanding of how effective Roundtrip can be in capturing correlations between the time series. If the algorithm is to be put to use outside of this project, though, it is worth keeping in mind the assumptions made here.

1.6 Related work

As already mentioned there have been several attempts to build upon GANs to allow for them to estimate densities [4], [7]. Furthermore, there have been many investigations into how to reduce the impact of the curse of dimensionality, although it is a problem that is proving difficult to solve in a feasible way [9].

The problem of market simulation can be generalized to generating synthetic time series data, which has applications from finance to health care to the automotive industry [10], [11]. A common theme in this field seems to be GANs, but it seems that a lot of research goes into applying GANs in a more intricate way, much like Roundtrip does. For example, one way to generate entire time series fragments, which can be seen as simulations, instead of individual data points is to use conditional GANs. These are trained in the same way as a normal GAN but are conditioned on the training data. This has been taken further as recurrent neural networks, or RNNs, are used for the generator and discriminator in the application of medicine [12]. In a similar way, Wasserstein GANs, or WGANs, have become popular for time series generation. [13] made use of this technique in an interesting way by converting time series data to grayscale images in order to generate new images that are then translated back to be presented in a time series form. The trends in the synthetic data sphere are relatively prominent and have room for much experimentation, and a good overview of is provided in [14]. As for the curse of dimensionality, there is also a lot of room for improvement. [15] provides an interesting example of a framework for dimension reduction for use in predictive maintenance for machines, and using manifold learning. In this case, discontinuity is an important feature of the data and an obstacle to overcome, which can be likened to discontinuities in certain financial data sets, such as options or swaps. It is also another example of how the problems in financial data generation and simulation are very similar to many other fields, and how the solution methods may very well be used across the different areas as well. In general it seems to be common to solve the problem of dimensionality in one step and then proceed with time series generation, which is different from the way in which it is done in Roundtrip. This break down may provide more options to solve the issue of the curse of dimensionality in the specific case of the 330 point 23 dimensional data set used in this thesis, and if not it may at least be worth investigating further. Regardless, the areas of time series generation and of dimensionality reduction are prominent in many fields and the methodologies mentioned here are but a few of what is available and what will be available in the near future.

1.7 Ethical considerations

The ethical aspects to take into consideration to are mostly related to the uncertainty of machine learning algorithms. Like most algorithms within the field, Roundtrip can be seen as a black box in some ways, where abstractions make it difficult to understand what is really happening. It is also important to note that results from studies such as this one do not always translate to other data sets, meaning that good performance on the data used in this project does not imply that similar results would be obtained with other market data.

2

Theory

This section aims to introduce all the theory that is necessary to properly understand this thesis. This section is broken down into two parts. Theory relevant to statistics will be covered first, and this will be followed by theory relevant to machine learning, as a great deal of the statistical theory is used in the machine learning part.

2.1 Statistics

2.1.1 Importance sampling

Importance sampling is a method that allows for properties of a certain distribution to be found while not actually being able to sample from this distribution. Samples are instead taken from another given distribution. This is extremely useful when looking at tail probabilities which would require a high number of samples if using the original distribution, but can instead be done with far fewer samples when picking a more suitable distribution to sample from.
Let $X : \Omega \to \mathbb{R}$ be a random variable in a probability space $(\Omega, \mathcal{F}, P)$, then a common way to estimate the expected value of X under probability distribution P would be by sampling from this distribution as

$$E_{P,n}[X] = \frac{1}{n}\sum_{i=1}^{n} x_i, \tag{2.1}$$

where the subscript P indicates that it is the expected value under probability measure P. In some cases, however, it may be difficult or impossible to collect these samples. It can therefore be useful to choose a second random variable, Y, such that $E_P[Y] = 1$ under probability measure P, and $Y \geq 0$. The probability P^Y can then be defined to satisfy

$$E_{P,n}[X] = E_{P^Y}[X/Y], \tag{2.2}$$

meaning that $E_{P,n}[X]$ can be estimated by sampling the random variable X/Y under probability measure P^Y. This is advantageous when it holds that

$$var_P[X] > var_{P^Y}[X/Y]. \tag{2.3}$$

2.1.2 Kullback-Leibler divergence and Jensen-Shannon divergence

The Kullback-Leibler divergence, or KL divergence, is a measure of how different two probability distributions are [16]. For two probability distributions P and Q in a probability space X the KL divergence can be formulated as

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) log \frac{P(x)}{Q(x)}, \tag{2.4}$$

and in essence tells the expected logarithmic difference between the two probabilities measured under probability P. Another fairly similar measure of difference is the Jensen-Shannon divergence, or JS divergence [17]. It can be formulated as

$$JSD(P||Q) = \frac{1}{2} D(P||M) + \frac{1}{2} D(Q||M), \tag{2.5}$$

with $M = \frac{1}{2}(P + Q)$.

2.1.3 Tail dependence

Tail dependence refers to the fact that random variables which exhibit little dependence can show correlation in their movements in extreme scenarios. This is a common phenomenon among stocks and financial markets and is difficult to accurately model.

2.2 Machine learning

2.2.1 Fully connected networks

The most simple form of neural network is a fully connected network, also referred to as a a multilayer perceptron. It is made up of layers of nodes, where each node represents some mathematical operation. There exists one input layer, one output layer, and at least one layer in between, called a hidden layer. Figure 2.1 makes clear the general structure of a multilayer perceptron.
Vectorized data is fed to the network, creating the input layer. As the data is propagated forward, each node takes on some so-called activation value, given through the formula

$$a_j^l = \sigma\Big(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \Big), \tag{2.6}$$

where the superscript represents the layer number and the subscript represents the node number. The term w_{jk} represents the *weight* of the connection between the k^{th} node in the $(l-1)^{th}$ layer and the j^{th} node in the l^{th} layer. The term b_j^l represents the *bias* of the j^{th} node in the l^{th} layer, which can be seen as a constant that makes training the network easier [18]. The function σ is called the activation function, that simply determines how the weighted sum is transformed. In essence, the activation of any given neuron is determined by a transformation of the sum of

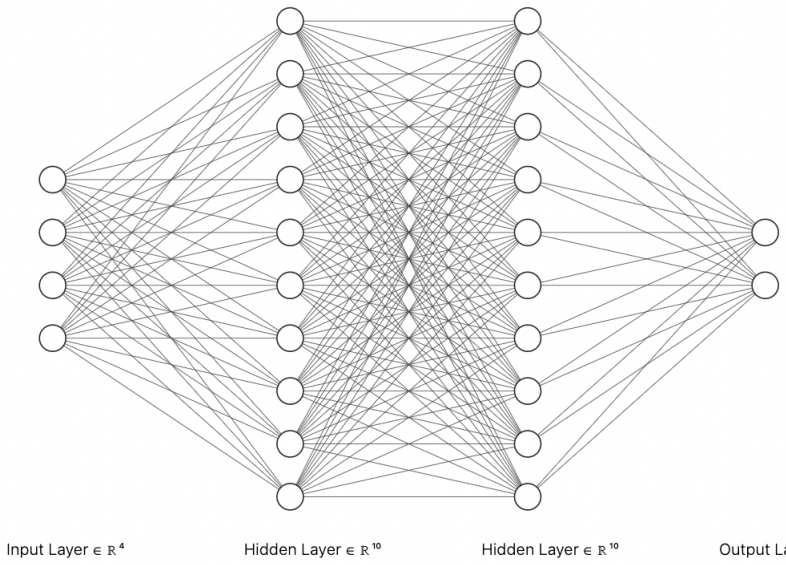

Figure 2.1: Diagram of a multilayer perceptron containing two hidden layers. The input layer has four nodes, the two hidden layers have ten nodes each, and the output layer has 2 nodes.

all the neurons, each multiplied by a weight, from the previous layer added together with a bias. This means that the input data is used for calculating the values in the first hidden layer, and the these values are then used in calculating the values in the second hidden layer, and so on. Eventually, the values for the output layer can be calculated.

To train a network it is necessary that the data used is labelled, making the algorithm what is referred to as *supervised*. When the input data has passed through the neural network and output values $\hat{y}$ have been generated, they can be compared to their label, or what they should really be, represented by y. By minimizing some loss function, measuring the difference between the obtained output and the desired output, the network can be improved. A common loss function is the mean squared error (MSE), given by

$$MSE = \frac{1}{n} \sum_{i=1^n} (y_i - \hat{y}_i)^2, \tag{2.7}$$

with n representing the number of output values and i indicating the specific value. Having this measure, it is possible to perform *backpropagation*, which is the process of one layer at a time moving through the network backwards and updating the weights. This is where methods such as stochastic gradient descent come in, which essentially amount to iteratively decreasing the value of the loss function. There are many ways in which this procedure can differ, but the premise is the same. One common aspect to alter is how the loss function is descended, and a relevant alternative is the Adam optimizer [19].

2.2.2 Manifold learning

It is common for data sets to be of a high dimensionality. This, on the one hand, allows for great detail in the data, but on the other hand, may cause difficulties in understanding patterns from the data. It is not always the case, however, that a higher dimensionality really does give more insight, as some dimensions may prove to be meaningless. For example, a two dimensional plane of data points can be rolled or twisted in any number of ways and embedded in three dimensions [20]. An observer of this three dimensional case may not see that in reality the data points come from a two dimensional plane, and this can lead to difficulties in understanding the data. Essentially, the data exists on a two dimensional manifold in three dimensional space. The process of reducing the dimensionality in this way is referred to as manifold learning, or as nonlinear dimensionality reduction. There are supervised ways of doing this, but in machine learning it is common to let the machine learning algorithm figure this out by itself.

2.2.3 Generative adversarial networks

The foundation of this project lies in generative adversarial networks, or GANs [5]. GANs are a type of machine learning framework that pit two neural networks, a generator and a discriminator, against each other. The way it works is that based off of example data the generator aims to produce results with the same distribution as the data, while the discriminator aims to tell the artificially produced data apart from the real data. As training progresses and the discriminator gets better at telling real data apart from artificially produced data, the generator gets better and better at producing realistic data.

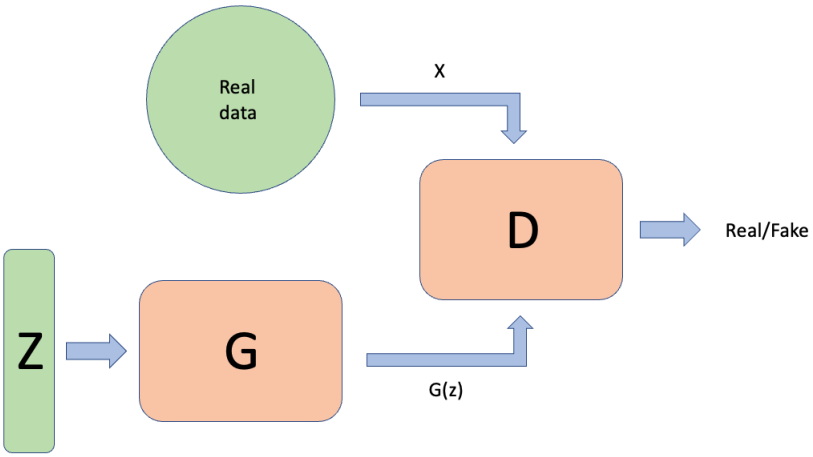

Figure 2.2: Diagram of a GAN. Random noise is sampled from latent spaze Z and fed to the generator G. The outcome, $G(z)$, is then evaluated by the discriminator D to determine whether it is real or fake.

The way this works mathematically is that the generator, G, and discriminator, D, play a minimax game where the generator aims to minimize the value function $V(G, D)$ while the discriminator aims to maximize it, with $V(G, D)$ given by

$$V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[ln(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[ln(1 - D(G(\mathbf{z})))]. \tag{2.8}$$

When the input into the discriminator is real data, $x \sim p_{data}$, the label should be 1, and when the input is fake data that has been created by the generator from noise, the label should be 0. This is very similar to the cross entropy loss function that is commonplace in machine learning. The function

$$min_G max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[ln(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[ln(1 - D(G(\mathbf{z})))] \tag{2.9}$$

thus implies that if the discriminator is good then $V(G,D)$ is large since $ln(D(x))$ will get a label close to 1 and $ln(1 - D(G(z)))$ will get a label close to 0. The opposite holds for the generator. Therefore, by training the discriminator to be as accurate as possible the system will by default create good fake data, which is the goal.
By differentiating the value function with respect to the discriminator and keeping G fixed it can be found that the optimal value of the discriminator is

$$D^*_G(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}. \tag{2.10}$$

Substituting this into the value function gives

$$V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[ln(\frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})})] + \mathbb{E}_{\mathbf{x}\sim p_g}[ln(1 - \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})})] \tag{2.11}$$

$$= \mathbb{E}_{\mathbf{x}\sim p_{data}}[ln(\frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})})] + \mathbb{E}_{\mathbf{x}\sim p_g}[ln(\frac{p_g(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})})]. \tag{2.12}$$

Now the minimax game can be reformulated as D maximizing $V(G,D)$. This result bears a lot of resemblance to a formulation of the Jensen-Shannon divergence,

$$JS(p1||p2) = \frac{1}{2}\mathbb{E}_{x\sim p_1}[ln(\frac{2p_1}{p_1 + p_2})] + \frac{1}{2}\mathbb{E}_{x\sim p_2}[ln(\frac{2p_2}{p_1 + p_2})], \tag{2.13}$$

where p_1 and p_2 are probability distributions. The value function $V(G,D)$ can be rearranged to fit this form but resulting in an additional $-2ln2$ term. This gives that

$$min_G V(G,D) = 2JS(p_{data}||p_g) - 2ln2, \tag{2.14}$$

and since it holds that the minimum value for any JS divergence is zero it is clear that the optimal value for the value function is $-2ln2$. This occurs when $p_{data} = p_g$.
The training is executed in a loop where each iteration consists of two parts. First the loss function for the discriminator is optimized using stochastic gradient ascent, and then once this is finished the same is done for the generator but with gradient descent. The difference here is of course because the discriminator wants to maximize the value function while the generator wants to minimize it. The training can be better understood with the help of the following pseudo code.
It is important to note that while the discriminator is updated k times in every iteration of the training loop, the generator is updated only once. This is due to

Algorithm 1 Training of a GAN

for no. training iterations **do**
  for k steps **do**
    - Choose m noise samples from $\mathbf{z}$
    - Choose m data samples from $\mathbf{x}$
    - Ascend stochastic gradient of D:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ln(D(\mathbf{x}^{(i)})) + ln(1 - D(G(\mathbf{z}^{(i)}))) \right]$$

  end for
  - Choose m noise samples from $\mathbf{z}$
  - Descend stochastic gradient of G:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[ln(1 - D(G(\mathbf{z}^{(i)}))) \right]$$

end for

the difference in gradients and it is a hyperparameter that needs to be adjusted on a case by case basis.

2.2.4 Roundtrip

2.2.4.1 Overview

The Roundtrip algorithm, proposed by Liu et al. [8] is at the core of this project. This is a deep generative model which generates both data and density estimates. This is an improvement over many generative networks as they often only generate data but cannot estimate the density itself.

Roundtrip is essentially set up as a bidirectional GAN, where data is passed through the two GANs creating a roundtrip. In the latent space a random variable $\mathbf{z} \in \mathbb{R}^m$ with a known density $p_{\mathbf{z}}(\mathbf{z})$, say Gaussian, is used as input for the first generator, G, and the output is then fed to a discriminator, D_x, along with a sample from the actual data, $\mathbf{x} \in \mathbb{R}^n$ with density $p_{\mathbf{x}}(\mathbf{x})$, for which the density estimation is desired. A second generator, H, then maps the actual data from the data space to the latent space, creating a backwards transformation. Once again, this generated latent variable along with the actual latent variable is fed to a discriminator, D_z. The dimension of the latent variable, m, is generally set to be less than that of the actual data, n. Thus the algorithm essentially uses manifold learning by letting the generators produce a manifold. Before the desired density estimation can be performed it is necessary to learn the generators G and H by training the network in a way similar to standard GANs.

2.2.4.2 Hyperparameters

As with GANs in general, Roundtrip utilizes classic multi-layer perceptrons for the generative and discriminative networks. The specifics of each of these four networks can easily be altered, but the original paper, [8], builds G with 10 layers with 512 nodes in each layer, and H with 10 layers with 256 nodes in each. The discriminator in the data space, D_x, is made up of four layers with 256 nodes in each, while the discriminator in the latent space, D_z, is made up of two layers with 128 nodes in each. The activation function used is always leaky-ReLu. Each of these choices can be seen as a hyperparameter of the overall system.

2.2.4.3 Learning G and H

To start off it is necessary to train the two generators. This is approached in the normal way, which is by minimizing some loss function. The total loss function used to train Roundtrip comprises several parts, the first of which pertain to the generators and discriminators specifically. These loss functions describing the adversarial training are denoted as

$$\begin{cases} L_{GAN}(G) & = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}(D_{\mathbf{x}}(G(\mathbf{z})-1)^2 \\ L_{GAN}(D_{\mathbf{x}}) & = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}(D_{\mathbf{x}}(\mathbf{x})-1)^2 + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}D_{\mathbf{x}}^2(G(\mathbf{z})) \\ L_{GAN}(H) & = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}(D_{\mathbf{z}}(H(\mathbf{x})-1)^2 \\ L_{GAN}(D_{\mathbf{z}}) & = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}(D_{\mathbf{z}}(\mathbf{z})-1)^2 + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})}D_{\mathbf{z}}^2(G(\mathbf{x})) \end{cases}. \tag{2.15}$$

It is worth pointing out that the loss functions used here differ from the loss functions normally used for GANs. Liu et al mention that the change of loss functions is due to a recommendation by LSGAN [21]. A roundtrip-loss is also defined in order to minimize the distance that a data point travels when transforming between the two spaces. The L_2 norm can be used to measure distance as the model assumes Gaussian errors, allowing for the roundtrip-loss to be defined as

$$L_{RT}(G,H) = \alpha||\mathbf{x}-G(H(\mathbf{x}))||_2^2 + \beta||\mathbf{z}-H(G(\mathbf{z}))||_2^2, \tag{2.16}$$

where α and β are constants that act as hyperparameters in the overall model. Having defined both the adversarial training loss functions and the Roundtrip loss function they can be combined to form the full training loss functions, which are defined as

$$\begin{cases} L(G,H) & = L_{GAN}(G) + L_{GAN}(H) + L_{RT}(G,H) \\ L(D_{\mathbf{x}},D_{\mathbf{z}}) & = L_{GAN}(D_{\mathbf{x}}) + L_{GAN}(D_{\mathbf{z}}) \end{cases}, \tag{2.17}$$

leading to the overall minimization problem

$$G^*,H^*,D_x^*,D_z^* = \begin{cases} argminL(G,H) \\ argminL(D_x,D_z) \end{cases}. \tag{2.18}$$

The parameters of the four networks are updated iteratively throughout the training process until the average log likelihood on an already chosen validation set shows no further improvement.

2.2.4.4 Density estimation

Given that the G and H are learned, the task of density estimation can be performed. The density of the latent variable, $p_{\mathbf{z}}(\mathbf{z})$, is from the beginning set to be Gaussian. The generator G is then assumed to produce a mapping $G(\mathbf{z}) = \tilde{\mathbf{x}}$ with an error following a Gaussian distribution, that is

$$\mathbf{x} = \tilde{\mathbf{x}} + \epsilon, \quad \epsilon_i \sim N(0, \sigma^2). \tag{2.19}$$

As mentioned already, given that $m < n$, this process can be viewed as allowing the generator G to learn a manifold and estimate the density of the data as a Gaussian mixture on the manifold. Choosing $p_{\mathbf{z}}(\mathbf{z})$ to be standard Gaussian, that is

$$p_{\mathbf{z}}(\mathbf{z}) = \left(\sqrt{2\pi}\right)^{-m} exp(-(1/2)||\mathbf{z}||_2^2, \tag{2.20}$$

allows for the conditional density of the data to be modelled as

$$p_{\mathbf{x}|\mathbf{z}}(\mathbf{x}|\mathbf{z}) = \left(\sqrt{2\pi}\right)^{-n} exp(-(1/2\sigma^2)||\mathbf{x} - G(\mathbf{z})||_2^2. \tag{2.21}$$

The density of the data can be estimated then by integrating this with respect to $\mathbf{z}$, giving a final density of

$$p_{\mathbf{x}}(\mathbf{x}) = \int p_{x|z}(\mathbf{x}|\mathbf{z}) p_{\mathbf{z}}(\mathbf{z}) d\mathbf{z} = \left(\frac{1}{\sqrt{2}\pi}\right)^{m+n} \sigma^{-n} \int exp(-(||\mathbf{z}||_2^2 + \sigma^{-2}||\mathbf{x} - G(\mathbf{z})||_2^2)/2). \tag{2.22}$$

Thus the desired density is obtained by evaluating the above integral. Liu et al. propose both importance sampling and Laplace approximation for doing so, but in the results of [8] it is clear that importance sampling consistently outperforms Laplace approximation in this task.

It is worth noting that Roundtrip only implicitly estimates the desired density, meaning that the estimated density can only be observed. It cannot be used as a PDF or CDF to generate data with this density. Data generation is, however, still easily performed as once the generator G is learned it can be used to generate as much data as desired.

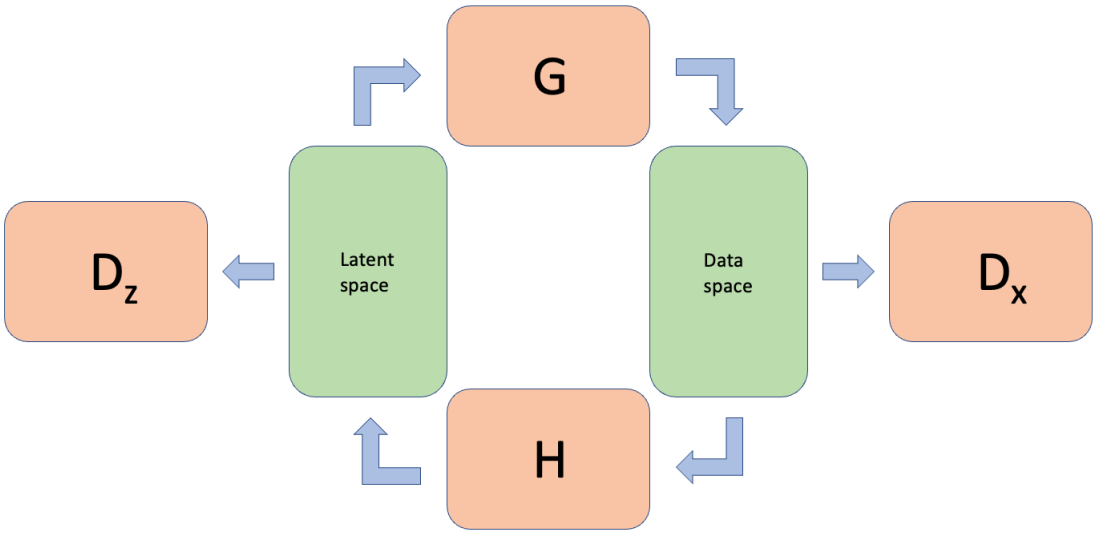

Figure 2.3: Diagram of the Roundtrip algorithm. G and H represent the forward and backward mapping generators, respectively, while D_z and D_x represent the two discriminators.

3
Methods

This section begins by describing the tools and programs used, the choice of data and how this data was collected. It carries on by giving a brief description of the modifications done to the Roundtrip algorithm, including hyperparameter tuning, and finishes off by explaining how evaluation of results was done.

3.1 Tools

The execution of this project was conducted with the help of the Python programming language, as well as the packages *NumPy* [22], *Pandas* [23], and *Matplotlib* [24]. The access to the Roundtrip source code was made available on Github by the original author, and this code was altered slightly to fit the data used in this thesis.

3.2 Data

The goal of this thesis was to investigate whether or not the Roundtrip algorithm is suitable for generating realistic financial data that captures correlations more accurately than the methods that the bank uses currently. Therefore, real market time series data was of course used, but before this it was instructive to use fake, or simulated, data with a known distribution. This allows for the algorithm to be tested in a variety of circumstances which might give insight into its general behavior, before being used on market data of which the distribution is not known. From here on the term "market data" will be used to describe the historical market data provided by Handelsbanken, while the term "simulated data" will be refer to the fake data that the algorithm will first be tested on. The term "generated data" will refer to the data that the Roundtrip algorithm produces. Context will tell whether the "generated data" refers to data generated after training on "market data" or on "simulated data". The one exception to this is in certain plots, where the term "original data" is used to denote the data on which the algorithm was trained.

3.2.1 Simulated data

The data that Roundtrip was first tested on is a simple set with a multivariate Gaussian distribution. This choice is motivated by the fact that in [8] Roundtrip performed relatively well on data with some form of Gaussian distribution and would therefore make it easy to understand which changes result in worsened performance. More specifically, the data that was used is four dimensional with the mean for each

dimension given by one of the two sets as shown in table 3.1 and covariance matrix shown in table 3.2.

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
set1	-0.20	-0.10	0.10	0.20
set2	0.80	0.90	1.10	1.20

Table 3.1: Means of simulated data

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Dimension 1	0.01	0.005	0.004	0.005
Dimension 2	0.005	0.02	0.003	0.005
Dimension 3	0.004	0.003	0.03	0.004
Dimension 4	0.005	0.005	0.004	0.04

Table 3.2: Covariance matrix for simulated data points

The means were chosen to somewhat loosely resemble returns in financial data, with the difference between them being that in set1 the values represent annual returns normalized to 0, while in set2 they represent annual returns normalized to 1. The outcome of this comparison allows for an understanding of how data should be normalized when fed to the Roundtrip algorithm, if there is any difference at all. Normalization of data is a common practice in machine learning as it can make allow for faster training, although it is still not fully understood why [25]. There are quite a few ways in machine learning to normalize data, but due to the fact that returns can be negative as well as positive, some methods were ruled out and the stated method is used. Further investigation into how normalizing should be performed could be of future interest. The covariance matrix was chosen rather arbitrarily, but with the single goal of ensuring that variances were not unrealistically large. Experimentation was first done with randomized covariance matrices but upon analyzing the randomly generated resulting data it was determined that the data points were in general not representative of realistic financial data, thus defeating the purpose. For each of the two sets of means, two sets of data were generated, the first with 300 data points, and the second with 3,000 data points. These choices are motivated by the fact that the available financial data contains a little over 300 data points for each time series, so understanding how Roundtrip performs on so little data is important. This could then be compared to a data set 10 times larger, with the goal of understanding how the behavior changes when more data is available. It seems evident that performance will simply improve, but understanding in which ways can be useful for other potential applications of the algorithm, especially considering that the data set used as the market data was unusually small.

3.2.2 Financial time series data

Roundtrip was also be tested on market time series data. This data set contains 23 separate time series with 330 data points in each. This can be interpreted as

330 23-dimensional points. The different time series include indices such as Sweden, the Nordics, Emerging Markets, VIX, and so on. As with much financial data, there exists strong correlations between many of the dimensions, especially when comparing, for example, Swedish and Nordic data, as Swedish data is part of Nordic data. The full list of separate time series included are shown in the list below.

- Barcta index
- Emgerging markets index
- Europe index
- GSCI commodity index
- Hedgefund index
- Noridc index
- North America index
- Norway index
- Pacific index
- Sweden index
- World index
- SE CPI
- SE REPO
- SE STIBOR 3M
- SE Swap 1 year
- SE Swap 2 year
- SE Swap 3 year
- SE Swap 5 year
- SE Swap 7 year
- SE Swap 10 year
- SE Swap 30 year
- USD SEK exchange rate
- VIX

The market index values in and of themselves are difficult to make use of, but the change in the values is the main point of interest. Therefore, the data set was restructured in such a way that it recorded only the relative differences between data points by dividing a value by the value before it. For example, an index value of 100 at one time step and a value of 105 at the next time step would yield a value of 1.05 after restructuring. Then, according to the what was found after experimentation with simulated data, this was either to be left as 1.05, having the data normalized to 1, or to be normalized to 0, meaning it would transform into 0.05.

3.3 Roundtrip Algorithm

3.3.1 Algorithm modifications

The Roundtrip algorithm was described in detail in section 2.2.4 but there are several aspects of the algorithm to take note of still. Primarily, throughout the experimentation with the algorithm, learning rates and constants pertaining to specific components of Roundtrip would remain constant, and were in fact unchanged from the original implementation in [8]. These include learning rates within the separate GANs as well as the optimizers for the neural networks making up the components of the GANs. One could very well use something other than the Adam Optimizer, or use it but with a different hyperparameter, but that is not included in this thesis. The reason for this is the Adam optimizer has in general demonstrated strong performance [19].

3.3.2 Hpyerparameter tuning

This thesis does, on the other hand, consider how the two hyperparameters, α and β, affect the algorithm. [8] did not experiment with altering these two hyperparameters

and instead left them both at values of 10. Two classic ways of choosing hyperparameters for hyperparameter tuning are gird search and random search. Grid search implies choosing hyperparameter values located at predetermined evenly distributed points, in this case on a two dimensional grid. Due to the time it takes to iterate through each grid point it is often more efficient to perform the hyperparameter tuning through a random search [26]. For this 9 pairs of hyperparameter values were chosen at random from integers in the interval of 1 to 20, with the tenth pair being (10,10), with the idea that the values from the original paper of (10,10) are located near the center.
The hyperparameter tuning was evaluated by looking at the average log likelihood for each case, and the hyperparameter pair giving the highest average log likelihood value was chosen to be used for generating data. Hyperparameter tuning was only done on the market data, as the simulated data was evaluated using the same values of α and β as in the original paper, which is 10.

3.4 Evaluation

3.4.1 Simulated data

Evaluating how Roundtrip performed on the simulated data was first done by visually interpreting the data generated by the algorithm. Since the simulated data set was four dimensional, the data generated by Roundtrip would also be four dimensional. One way of quickly gaining an understanding of the algorithm's behavior was to plot all 2-dimensional combinations of the generated data along with the corresponding plot for the simulated data. In total that resulted in 6 such combinations. This was done for all four simulated data sets, meaning for both means of $set1$ and $set2$ and for each of these with both 300 and 3,000 data points. The goal was that this would give an intuitive understanding of how the market data should later be normalized and an understanding of how much the performance varied with data set size.
Of particular interest were the tail dependencies as these often prove extremely difficult to estimate using traditional methods. This is due to the fact that far fewer data points belong to the tail of any given distribution, so in general even if the joint distribution of two data sets can be estimated relatively well, it does not at all mean that the tails can be estimated well. If Roundtrip was able to generate data with similar tail dependencies as the original data, then the algorithm in a sense proves itself capable of helping to reduce the 'curse of dimensionality'. From the generated data it was also easy to calculate the mean in each dimension which could be compared to the known mean, and the covariance of the generated data could be estimated using *NumPy* and then compared to the actual covariance matrix.
Additionally, the algorithm was tested with two, three, and four latent space variables as input. It is known that the simulated data was 4-dimensional making it realistic to assume that the scenario with four latent space variables would result in the best performance, but it would nonetheless be investigated in case the algorithm exhibits any peculiar behavior. Applying machine learning algorithms to small data sets is often extremely challenging and therefore it was of utmost importance to

gauge the performance under these various aspects that would normally not be of particular interest.

3.4.2 Market data

The performance on the market data was evaluated visually in the same way as the performance on the simulated data. As the generated data set was 23-dimensional and thus contained far too many 2-dimensional combinations to include them all, only some were to be evaluated and included for further discussion. Once again, the tail dependencies were of great interest. The number of latent space variables was set to 10 as this was similar to an example in the original source code.
Finally, to compare the results with the current methods used by Handelsbanken, Roundtrip was used to generate 10,000 20-year simulations for each index. Given that only 28 years worth of monthly data is available, simulating each index 20 years into the future is a difficult task but is ultimately the purpose of the project. The 10,000 simulations were shown in the form of a histogram, that is, evaluated visually and with common sense and compared to the current simulations performed by the bank. If Roundtrip fails to deliver reasonable simulations the 2-dimensional plots could potentially be used to give some insight into why. This way of simulating markets, buy taking the generated points straight from the output and compounding them, is a fairly naive but straightforward way of performing the simulation. It is very possible that the bank could make use of the generated data and process it in some other way to incorporate it into their own simulation methods.

4

Results

In this section the results from testing the Roundtrip algorithm on simulated data will be covered first, followed by the results from the market data. The point of the simulated data is to allow Roundtrip to be tested on a data set where the distribution is known. This can be seen as a safe environment to test Roundtrip in, or as a playground that can be used to gauge general properties of the algorithm before applying it on market data.

4.1 Simulated data

The first part of the results focuses on the data generated from training Roundtrip on simulated data. The results are presented in figures 4.1, 4.2, 4.3, and 4.4, where each figure shows the six possible 2-dimensional combinations. The simulated data on which the algorithm was first trained is included in the plots as well to make the comparison easy.

Figure 4.1 shows the outcome of training Roundtrip on 300 data points where the first dimension has a mean of -0.20, the second dimension has a mean of -0.10, the third has a mean of 0.10, and the fourth has a mean of 0.20. Figure 4.2 shows the outcome of training data with the same characteristics but with 3000 training points. 4.3 shows the case for 300 training points with means 0.80, 0.90, 1.10, and 1.20, while 4.4 shows the case for the same means but with 3000 training data points. For all four scenarios the covariance matrix presented in the section 3 is used.

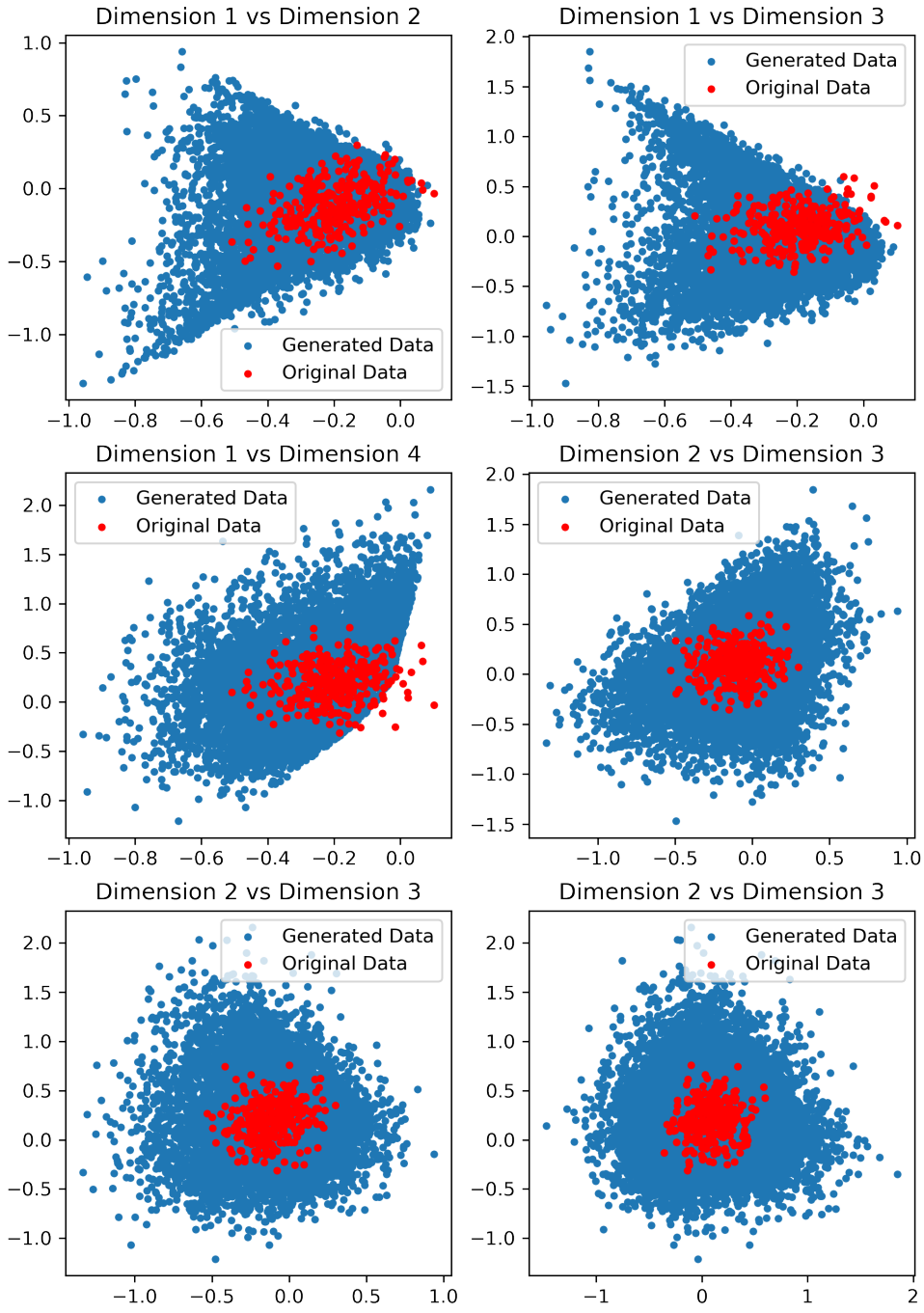

Figure 4.1: Generated data based on 300 training points with dimension means [-0.20, -0.10, 0.10, 0.20] and covariance matrix as described in the section 3.

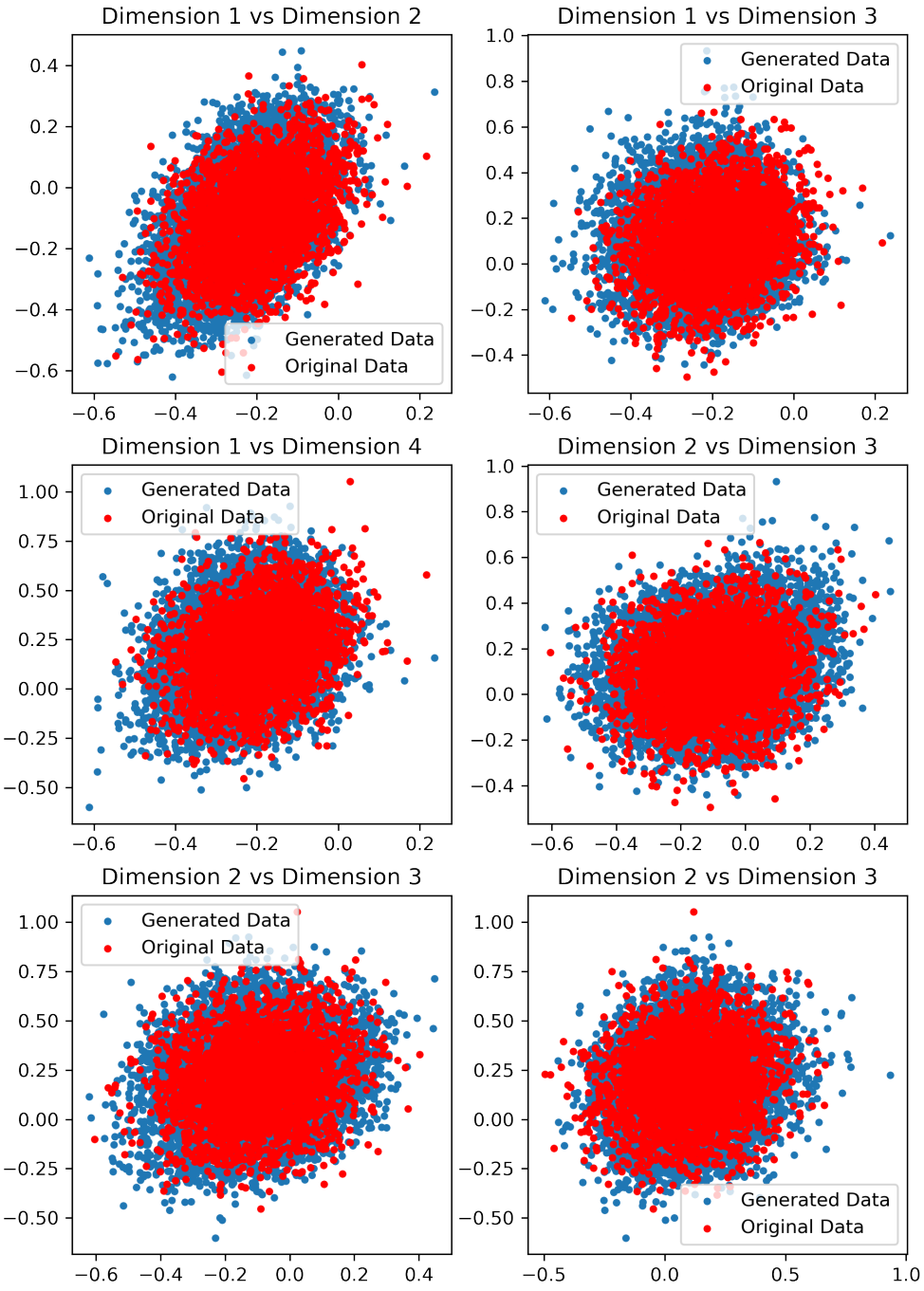

Figure 4.2: Generated data based on 3,000 training points with dimension means [-0.20, -0.10, 0.10, 0.20] and covariance matrix as described in the section 3.

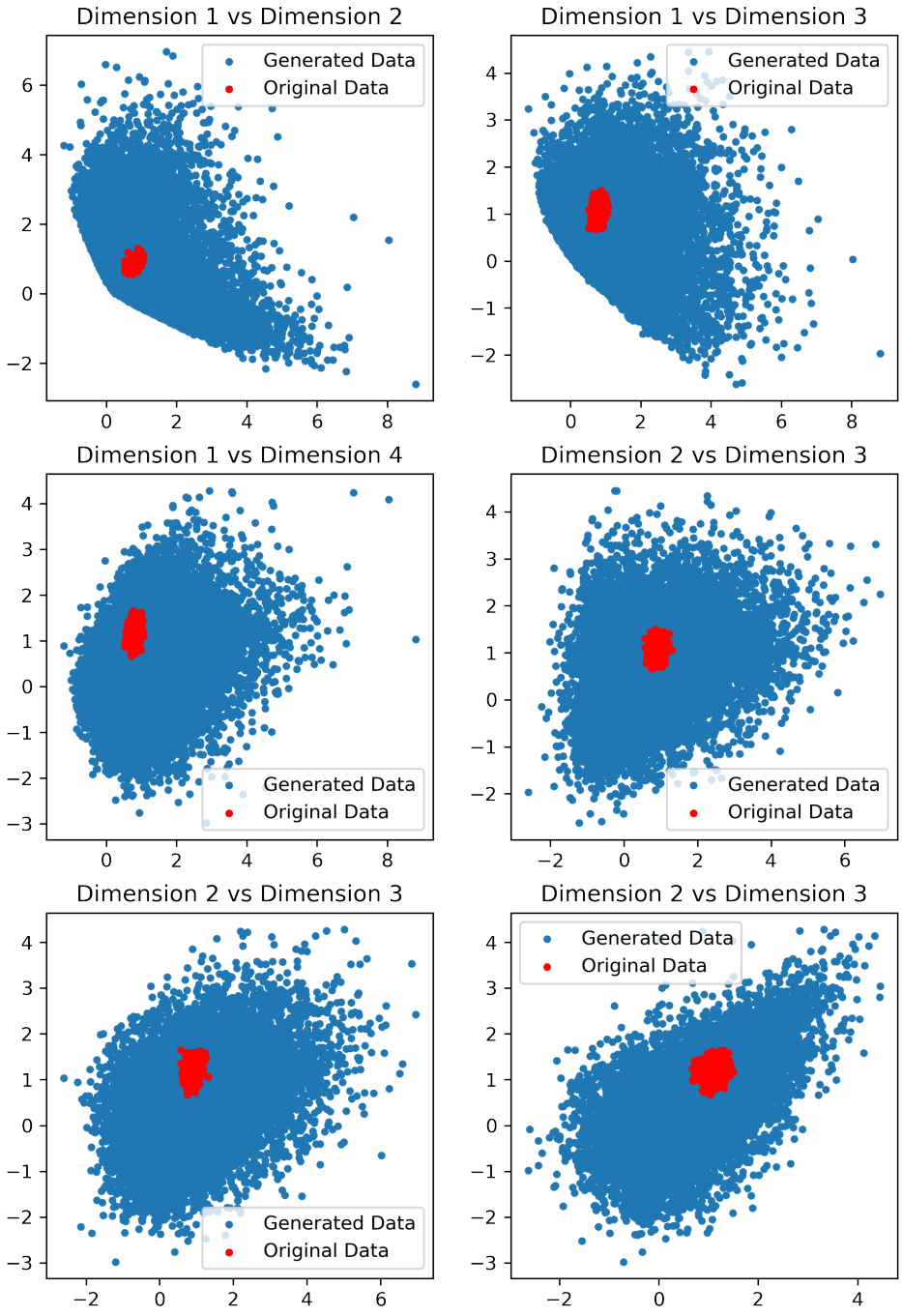

Figure 4.3: Generated data based on 300 training points with dimension means [0.80, 0.90, 1.10, 1.20] and covariance matrix as described in the section 3.

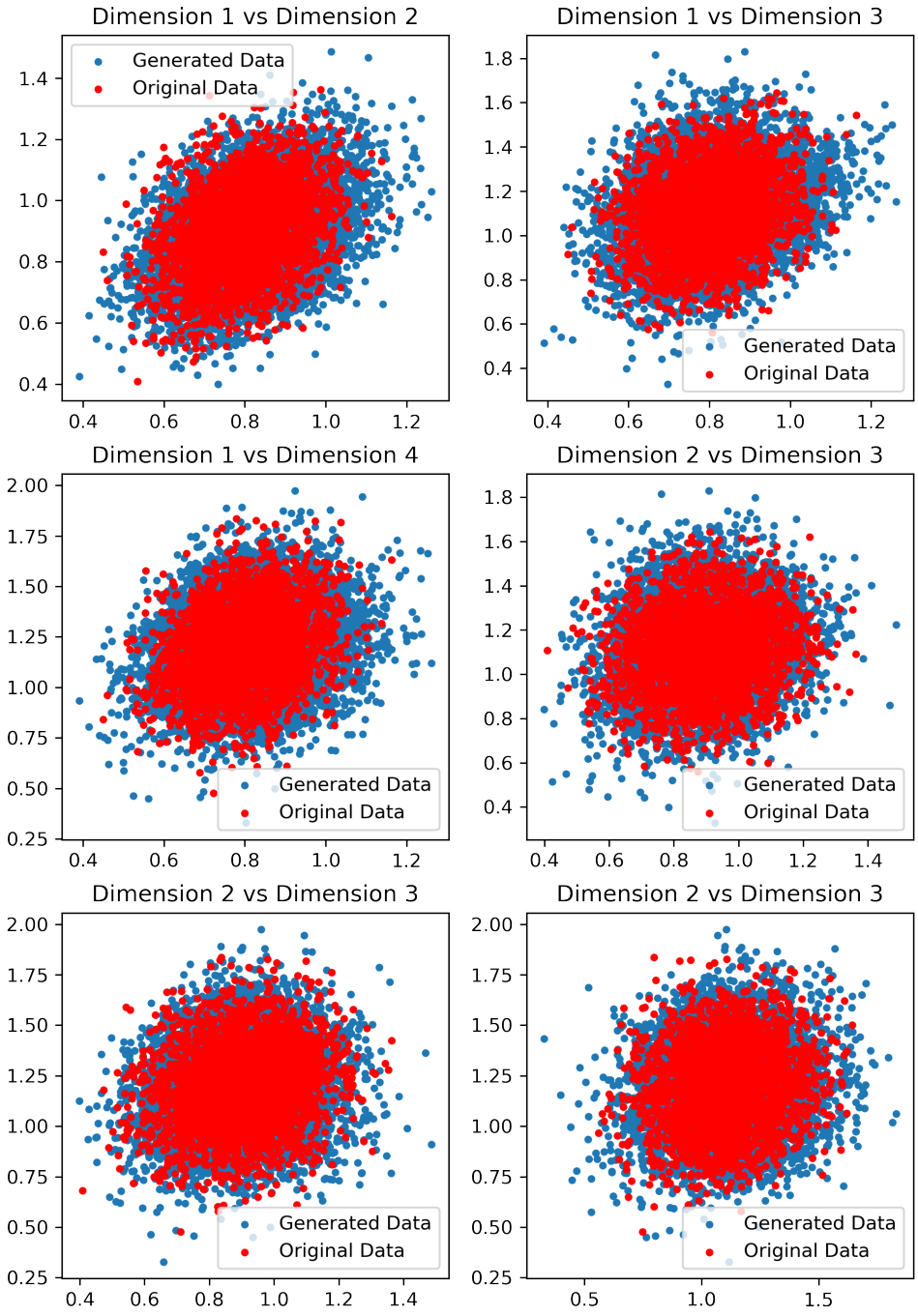

Figure 4.4: Generated data based on 3,000 training points with dimension means [0.80, 0.90, 1.10, 1.20] and covariance matrix as described in the section 3.

From the generated points it was also possible to use the *NumPy* Python package to find the mean along with estimates of covariance matrices. The means are shown in table 4.1 in the same way as was presented in the section 3. The estimated covariance matrices are presented in tables 4.2, 4.3, 4.4 and 4.5 where each table presents the estimated covariance matrix for one of the two sets of means and for training sets of either 300 or 3,000 data points.

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
set1 300	-0.27089	-0.07958	0.10014	0.20842
set1 3,000	-0.21579	-0.09983	0.10635	0.19871
set2 300	1.25380	0.88402	0.70818	0.45902
set2 3,000	0.81312	0.88398	1.10578	1.19748

Table 4.1: Means of generated data for each dimension

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Dimension 1	0.02541578	0.00963698	-0.01135291	0.02871501
Dimension 2	0.0096398	0.08465109	0.04558333	-0.01287723
Dimension 3	-0.01135291	0.04558333	0.18560274	-0.0096426
Dimension 4	0.02871501	-0.01287723	-0.0096426	0.18427518

Table 4.2: Estimated covariance matrix for generated points with means from set1 and training on 300 data points

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Dimension 1	1.49744783	-0.71773465	-0.2320037	0.39673559
Dimension 2	-0.71773465	1.83260892	0.37307454	0.51586191
Dimension 3	-0.2320037	0.37307454	1.06024278	0.65766823
Dimension 4	0.39763559	0.51586191	0.65766823	1.15120257

Table 4.3: Estimated covariance matrix for generated points with means from set2 and training on 300 data points

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Dimension 1	0.0091938	0.00675771	0.00078229	0.00444305
Dimension 2	0.00675771	0.02219179	0.00499066	0.00558318
Dimension 3	0.00078229	0.00499066	0.02781123	0.0040292
Dimension 4	0.00444305	0.00558318	0.0040292	0.04061673

Table 4.4: Estimated covariance matrix for generated points with means from set1 and training on 3,000 data points

	Dimension 1	Dimension 2	Dimension 3	Dimension 4
Dimension 1	0.01312451	0.00591715	0.00452181	0.00434858
Dimension 2	0.00591715	0.01841133	0.00363017	0.00351549
Dimension 3	0.00452181	0.00363017	0.03018594	0.00211967
Dimension 4	0.00434858	0.00351549	0.00211967	0.03744074

Table 4.5: Estimated covariance matrix for generated points with means from set2 and training on 3,000 data points

4.2 Real data

4.2.1 Hyperparameter tuning

The first part of applying the Roundtrip algorithm to market data is hyperparameter tuning. For the sake of efficiency the random search algorithm was chosen on a two dimensional grid with each dimension consisting of all integers from 1 to 20. 9 points were generated randomly using *NumPy*, with the tenth point being (10,10) as used in the original paper. The resulting points are presented in figure 4.5.

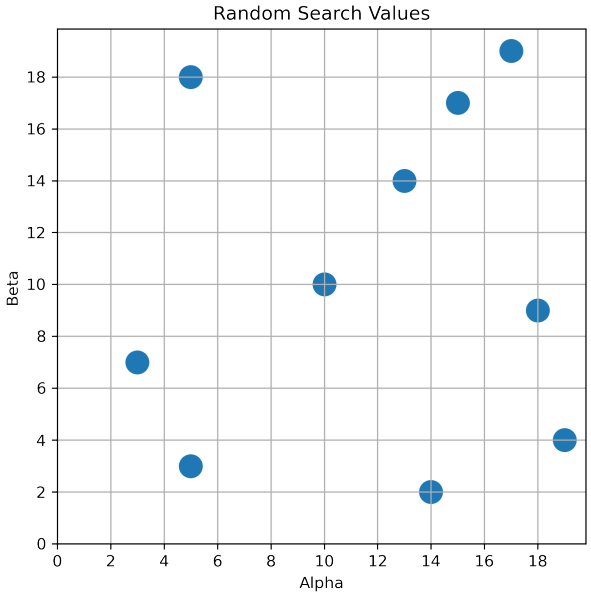

Figure 4.5: Grid showing which combinations of α and β were used in the random search for hyperparameter tuning.

Evaluation of the performance with each set of hyperparameters was based on the log-likelihood value generated by the algorithm. Roundtrip was tested several times for each hyperparameter combination, and in the end the different hyperparameter combinations differed in their respective log-likelihood value extremely little. Nonetheless, the hyperparameter combination that resulted in the best performance

was with $\alpha = 5$ and $\beta = 3$, and this combination was thus chosen to be used for the remainder of the result generation.

4.2.2 Market data

Observing the properties of all possible combinations of 23 dimensions is a bit overwhelming, so this section is limited to three 2-dimensional index comparisons: Sweden vs Norway, Sweden vs World, and Nordic vs SE Swaps 7 years. Figures 4.6, 4.7, and 4.8 provide 2-dimensional scatter plots showing the generated synthetic data along with the original data on which the algorithm was trained. The 2-dimensional combinations presented were chosen because they demonstrate the different ways in which the algorithm can behave. Along with each scatter plot are two marginal histograms which make clear the 1-dimensional distribution of both the generated data and the original data in each case.

4.2.3 Market simulation

In figure 4.9 the results of 10,000 20 year simulations are presented in the form of histograms for four separate markets. This allows for a clear visualization of the distribution of the simulation results. The results are presented in such a way that the histograms represent by what multiple the market changes by after 20 years. As can be seen in the upper left histogram, Roundtrip generated data results in 20 year simulations where the majority of the simulations indicate that the market will grow by a factor of around 5,000. In the same way, the simulations for the world index indicate that it will end up at a value of around 0.00015 of the current state.

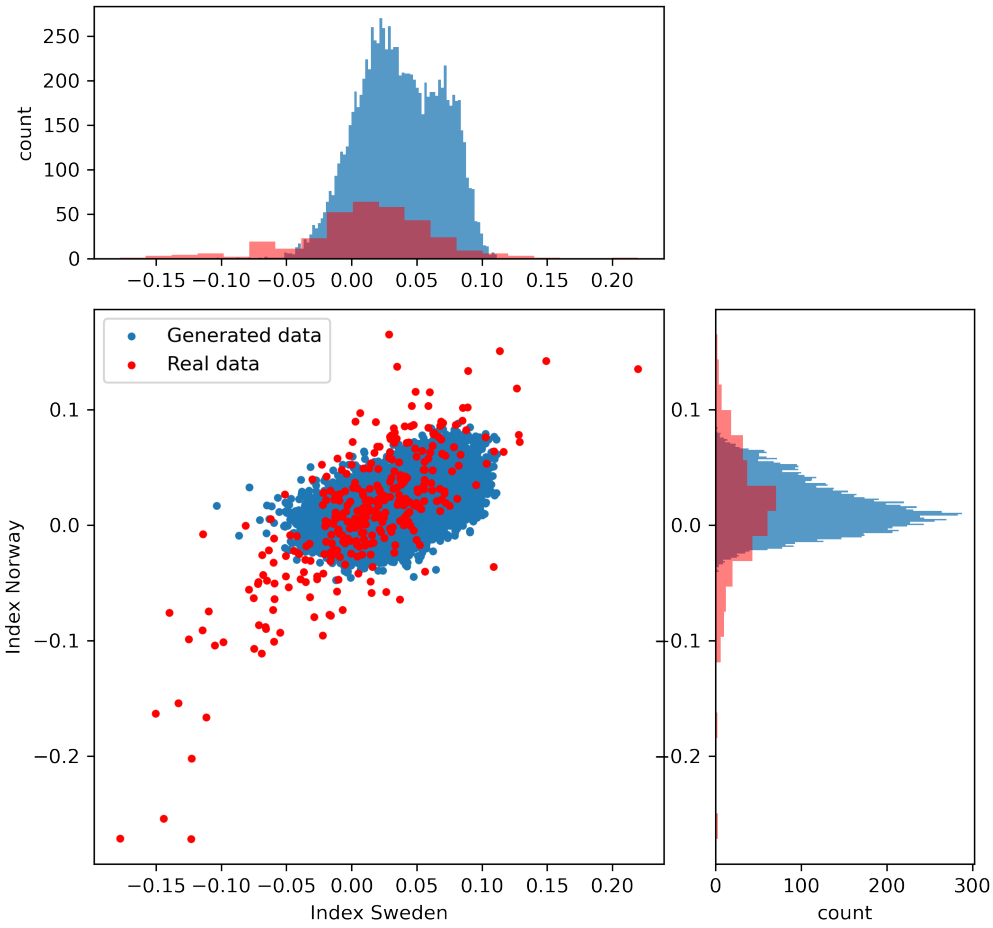

Figure 4.6: Scatter plot showing all generated data points along with the original data points for Index Sweden and Index Norway. The marginal histograms show the respective 1-dimensional distributions.

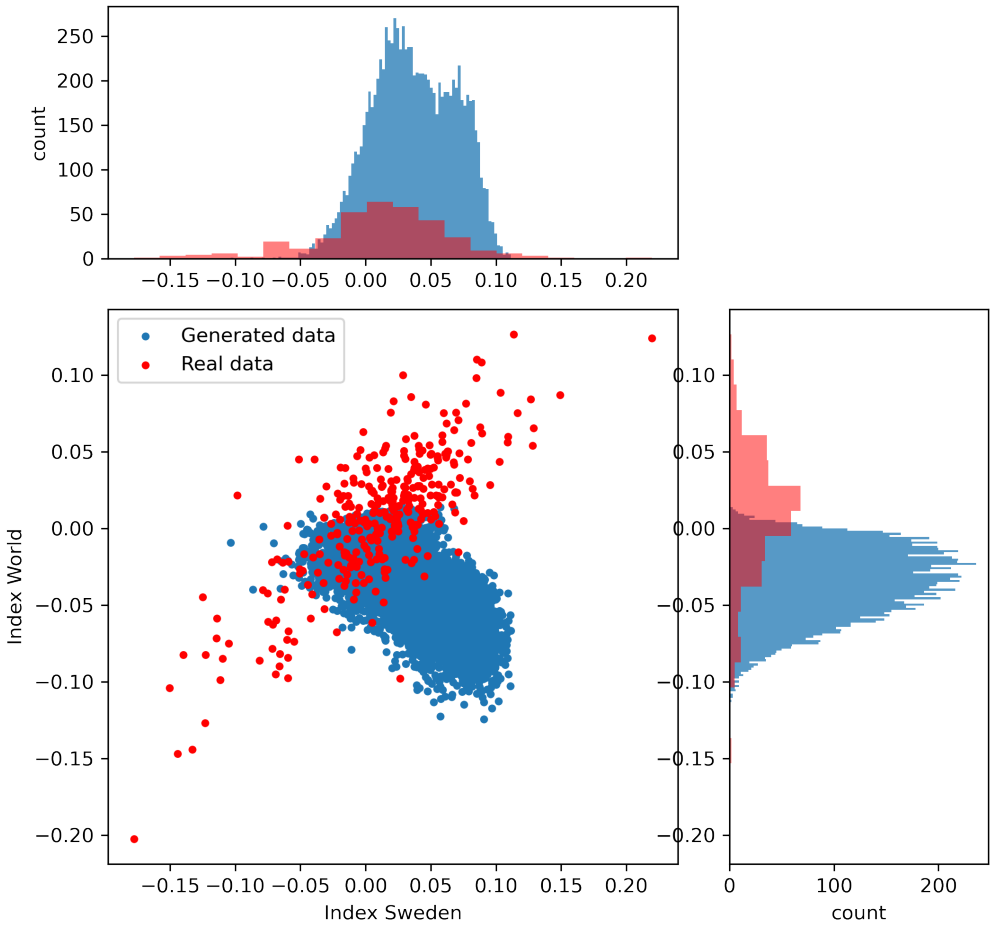

Figure 4.7: Scatter plot showing all generated data points along with the original data points for Index Sweden and Index World. The marginal histograms show the respective 1-dimensional distributions.

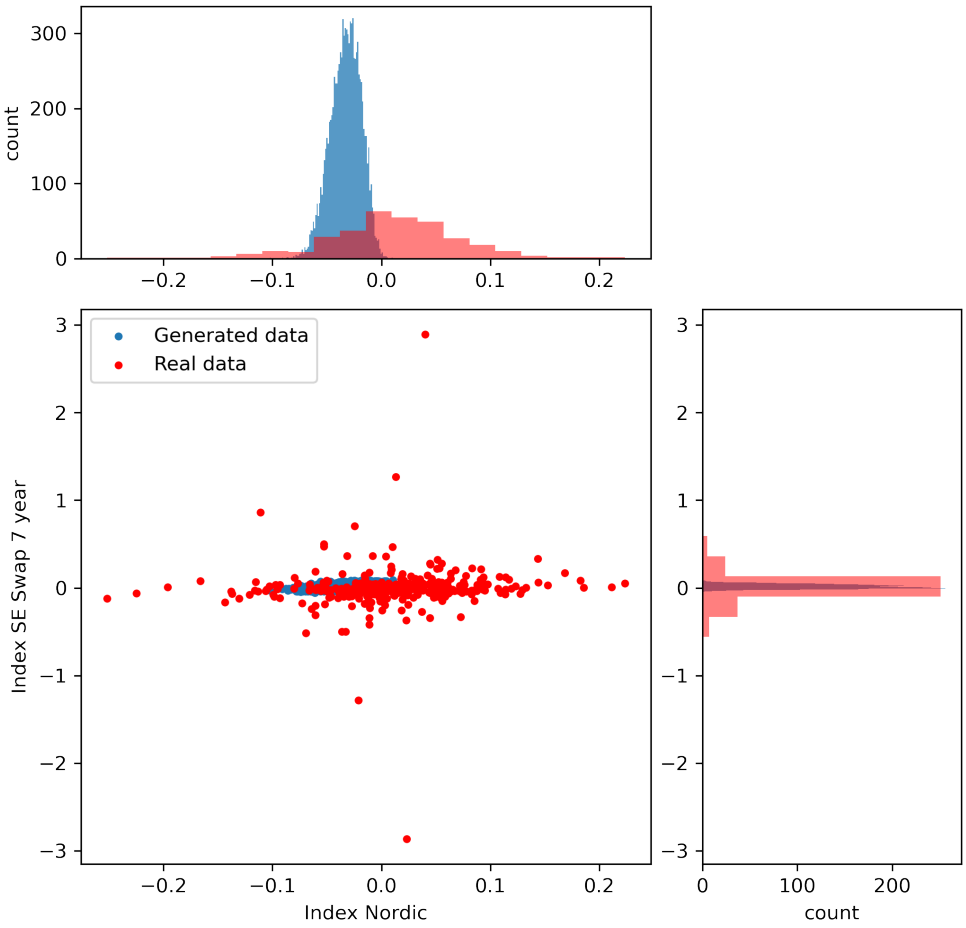

Figure 4.8: Scatter plot showing all generated data points along with the original data points for Index Nordic and Index SE Swap 7 year. The marginal histograms show the respective 1-dimensional distributions.

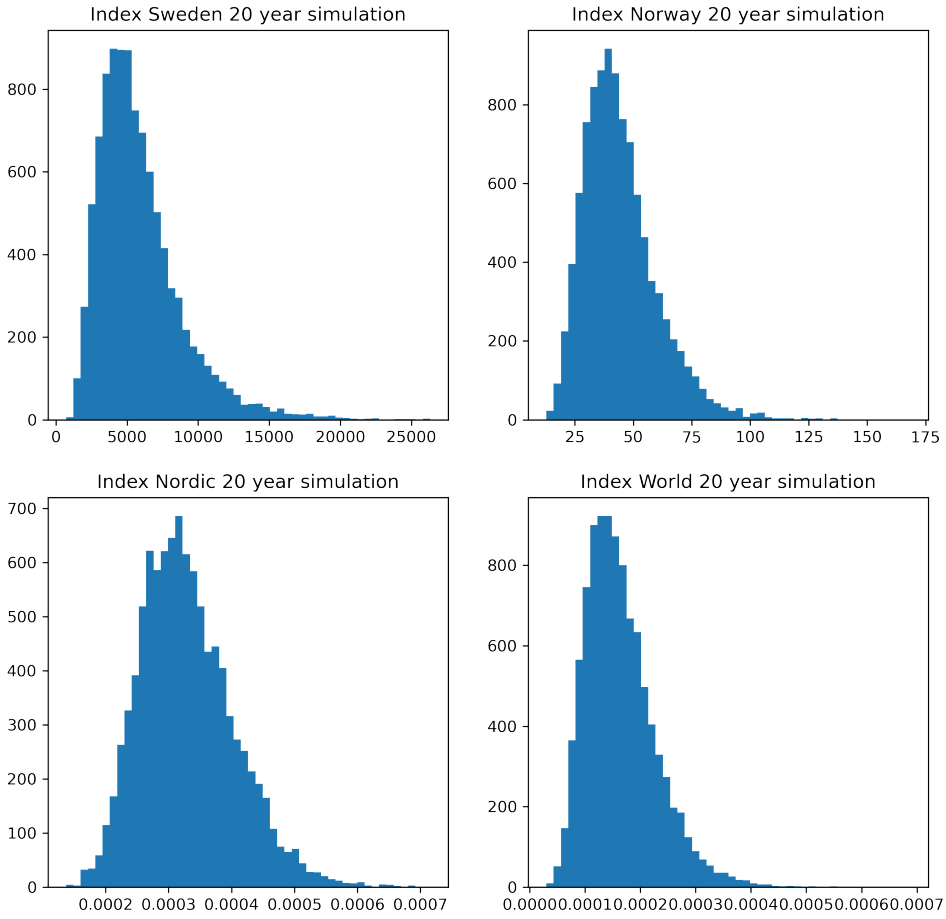

Figure 4.9: The figure shows 20 year simulations of four different markets. The histograms represent the distribution of the returns of 10,000 simulations in the format of multiples of growth. For example, 50 on an x-axis would mean that the 20 year simulation resulted in a growth of that market by a factor of 50.

5
Discussion

This section aims to provide the reader with a thorough understanding of the importance and significance of the results and to connect the results to the overall goal of the thesis. The section finishes off by mentioning some other potential uses for the Roundtrip algorithm that have not been investigated in this thesis.

5.1 Simulated data

5.1.1 Latent space investigation

The simulated data was first used to test in a very simple way how Roundtrip performs with different latent space dimensions. Given that the simulated data is in reality made up of four distinct dimensions, each with their own unique mean, it should in theory not be possible for Roundtrip to perform any sort of dimensionality reduction on the data. This would mean that using a two or three dimensional latent space should give worse results. The results from this experiment, however, seem to not be indicative of anything as in hindsight it would be more interesting to perform this with a greater number of dimensions in the data space than four. As the results of this give no greater insight into how Roundtrip performs in general, the results were not necessary of further analysis.

5.1.2 Evaluation

The point of the simulated data was to give a general understanding of how the Roundtrip algorithm might behave when applied on the fairly simple multivariate normally distributed data. The two major points of interest are how the data should be normalized and how big of an effect training data set size has on performance. How the data should be normalized is of interest because when Roundtrip is later on applied to market data there are not a lot of data points, so it is essential that its application is done correctly. Comparing figures 4.1 and 4.3, the two scenarios where only 300 training examples were used, it is right away obvious that Roundtrip performs better when returns are normalized to zero as opposed to one. The generated data in figure 4.1 is confined to a much narrower range than in figure 4.3.
As can easily be predicted, Roundtrip also performs significantly better when 3,000 data points were fed into it as opposed to 300. It is noteworthy, however, that with 3,000 training examples the difference between normalizing the data to zero or one is significantly reduced. It may seem trivial to compare performance between a data set and one that is ten times larger, but in reality 3,000 training samples is still very

small, especially when compared to [8] where the least number of training examples used is nearly 40,000. Given the improvement when ramping up the number of training examples to 3,000, it means that even if the algorithm were to perform poorly on the market data with 330 examples of monthly data, it may be very useful if weekly data can be collected, to take just one example.
One particularly striking feature of the results with the simulated data is how Roundtrip generated data is confined to certain regions. For example, in figure 4.1 in all plots containing 'Dimension 1', the generated data is contained in a cone-like region, even though the data on which the algorithm was trained exhibits no such attribute. It is extremely difficult to know with certainty what is occurring to cause this though and would require detailed monitoring of every step of the training progress.
Upon observing tables 4.1 - 4.5 the trends picked up upon by the plots are made even more clear. Means of generated data are much more accurate for when 3,000 data points were trained on for both set1 and set2. For the cases with 300 training points it is again obvious that set1 had much better performance, as set2 seems to have generated means nowhere close to the true values.

5.2 Market data

5.2.1 Hyperparameter tuning

Investigation of the market data began with performing hyperparameter tuning for the learning rates α and β. Evaluation was done through the average log-likelihood, but varying the two values resulted in only a negligible difference in performance. Nonetheless, the combination that gave the best log-likelihood value, with $\alpha = 5$ and $\beta = 3$, was chosen. A potential explanation for the similar performance for all hyperparameter combinations is that training with only 330 23-dimensional data points is already so difficult that performance cannot be achieved beyond a certain level. For larger data sets it could therefore be advantageous to perform hyperparameter tuning in a much more rigorous manner. It is also worth mentioning again that besides α and β no hyperparameters were altered from the original implementation in [8] as there are far too many factors that can be changed to be encompassed in this project, such as number of layers and nodes in the generators and discriminators, for example.

5.2.2 Evaluation

With 23 dimensions of data available it is unreasonable to investigate the relationships between all of them, so three were chosen for further analysis. Figures 4.6, 4.7 and 4.8 show how market data on which Roundtrip was trained compares to the generated data for the corresponding indices. The points depict returns from one time step to the next, normalized to zero, meaning that an index with value 100 at one time point and 110 the next would show a return of 0.1. Looking first at figure 4.6, comparing the Swedish and Norwegian indices, it is clear that the generated data is much more concentrated than the market data, and thus fails to capture the

points corresponding to more extreme returns. Focusing on the marginal histograms makes it clear that the generated data is centered relatively close to the center of the market data, but its variance is significantly lower.
Figure 4.7 instead compares the Swedish index to that of the world, so the x-axis shows the same distribution as in Figure 4.6 but the y-axis is different. Right away it is clear that the distribution of the generated data along the y-axis is significantly shifted away from the market data. The scatter plot also portrays a distribution of the generated data with an almost inverse correlation to the market data, failing to capture the more extreme points from the market data but also generating points in a region where the market data showed nothing.
Finally, Figure 4.8 presents a slightly better distribution than the two previous figures as it compares the Nordic index with the index for SE Swaps of 7 years. It is important here to not be fooled by the apparently narrow distribution along the y-axis of both the market data and the generated data, as the scale is actually much large than that of the x-axis. The histogram representing the swaps shows that the generated data is centered accurately when compared to the market data, but along the x-axis the histogram makes it evident that the generated data is centered a bit to the left of the market data. Both axes, however, demonstrate a similar characteristic as figures 4.6 and 4.7, namely that the generated data is confined to a much narrower range than the market data.

5.3 Simulated data vs market data

The behavior of Roundtrip when trained on the simulated data is remarkably different from the behavior when trained on market data. The most obvious aspect is that in all cases when the algorithm was trained on market data, the generated data was confined to a smaller region than the actual market data was, meaning that the market data showed far more extreme values. This is the opposite of the case when trained on simulated data, as the generated data shows far more extreme values. A possible reason for this is that the simulated data was created according to a random multivariate normal distribution, and the generator G in the Roundtrip algorithm assumes an error in the mapping that is normally distributed as well. How this particular mechanism affects the outcome is worth deeper investigation.

5.4 Market simulation results

The final and most essential test of Roundtrip was whether or not it can be used to simulate financial markets, by taking the generated returns and using them to simulate how a market would evolve over many years. The simulation results in this study are used according to the method laid out in the section 3, meaning that generated results represent growth or decay, and these are then compounded to create the final result. Figure 4.9 depicts simulated returns after 20 years for 10,000 simulations and how these returns are distributed. Four indices, Sweden, Norway, Nordic, and World are shown. All histograms except that for the Norwegian index show very unrealistic results, with returns of multiples of several thousands

or extreme decays. The returns for the Norwegian index can be considered very high but are nonetheless not nearly as unreasonable as they are centered at around a multiple of approximately 40. Considering that the Norwegian index has grown by a factor of about 15 over the past 28 years, the results would mean that over the coming 20 years the market will grow about 2.7 times more than it has grown over the past 28. In general, though, it is quite clear from figure 4.9 that with the amount of data used in this project and with the same set up of the algorithm, market simulation is not a feasible task. Roundtrip's failure to pick up on the more extreme data points in figures 4.6, 4.7 and 4.8 becomes all the more important when the points are compounded to build the simulated market. No advanced mathematical tools are necessary to confirm this, as common sense is enough to determine that one market growing by a factor of several thousand while another decays by a factor of several thousand is not a realistic or usable result. It could be possible, however, to take the generated data points and alter them in some way to such that certain characteristics are preserved but the simulations are more realistic. Due to the fact that it is proprietary data, the exact results from the simulations that Handelsbanken currently run are not included in this report. It is the case, though, that the banks current methods produce far more realistic predictions for the financial markets over the coming decades.

5.5 Tail dependencies and the curse of dimensionality

With 330 data points of 23 dimensions each it was not possible for Roundtrip to provide sufficient insight into the tail dependencies between the various market indices and thereby help resolve the Curse of Dimensionality. Distributions of the generated data were consistently too narrow and thus automatically failed to capture the dependencies of the more extreme points. It can also be assumed that this failure in part led to the extreme results in the simulations, as small inconsistencies in the tails compound over time. Worthy of further investigation is how altering the number of latent space variables would affect the outcome as this was not included in the evaluation on real market data. As described in [8], G effectively learns the manifold on which the data actually lies, and the Roundtrip algorithm works to map the forward and backward transformation as efficiently as possible. Tuning this could potentially be of interest each time a new data set is used, but since there exists no direct way into gauging the optimal number of latent space variables, it is very much a procedure of trial and error.

5.6 Roundtrip's potential use in financial applications

None of this is to say that Roundtrip is not worthwhile other applications in the financial industry. As demonstrated with figures 4.1 - 4.4, even increasing the size of the available training data by a factor of 10 makes a noticeable difference in

Roundtrip's performance. Trying to capture tail dependencies with only 330 data points of 23 dimensions is an extremely ambitious task, as in [8] the data sets on which the algorithm trains generally contain over 40,000 data points. Another application could be in outlier detection. While not investigated in this thesis, [8] goes through Roundtrip's performance in outlier detection when tested on three data sets and compares this to the performance with two common outlier detection methods, namely one-class SVM and Isolation Forest. Roundtrip performs well in these tasks and does at least as well as the methods it is compared to. This means that it could potentially be used in areas such as fraud detection and anomaly detection in trading data, to name just two examples. Lastly, trying to alter the generated results as mentioned above in such a way that certain characteristics are preserved but simulation results are more accurate could be an interesting topic. This would mean combining the Roundtrip theory with other statistical techniques.

6
Conclusion

This thesis investigated the potential use of the Roundtrip framework presented in [8] for the purpose of simulating financial markets more accurately than Handelsbanken's current proprietary models. Training the model on simulated data provided insight into the behavior of the algorithm, and following this, Roundtrip was trained on real market data and then used to generate new data. The generated data was not able to accurately capture tail dependencies in a way that would aid the bank, and this most likely led to the inability to simulate markets over a longer period. It is quite clear, however, from the results of training Roundtrip on simulated data that increasing the size of the data sets by only a factor of 10 would yield much better results, meaning that providing the algorithm with daily data would potentially result in useful market simulations. The Roundtrip algorithm can certainly not be ruled out for use in financial services as this thesis aimed for the very ambitious goal of helping to reduce the curse of dimensionality, which is by no means an easy task. Lastly, while not investigated in this thesis, Roundtrip has other potential uses as it allows for outlier detection as well, which has various applications such as anomaly detection in trading data.

Bibliography

[1] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI magazine*, vol. 27, no. 4, pp. 12–12, 2006.

[2] J. Hendler, "Avoiding another ai winter," *IEEE Intelligent Systems*, vol. 23, no. 02, pp. 2–4, 2008.

[3] D. B. Lenat, R. V. Guha, K. Pittman, D. Pratt, and M. Shepherd, "Cyc: toward programs with common sense," *Communications of the ACM*, vol. 33, no. 8, pp. 30–49, 1990.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[6] T. Liang, "How well can generative adversarial networks learn densities: A nonparametric view," *arXiv preprint arXiv:1712.08244*, 2017.

[7] S. Arora and Y. Zhang, "Do gans actually learn the distribution? an empirical study," *arXiv preprint arXiv:1706.08224*, 2017.

[8] Q. Liu, J. Xu, R. Jiang, and W. H. Wong, "Density estimation using deep generative neural networks," *Proceedings of the National Academy of Sciences*, vol. 118, no. 15, p. e2101344118, 2021.

[9] F. Bach, "Breaking the curse of dimensionality with convex neural networks," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 629–681, 2017.

[10] S. Dash, R. Dutta, I. Guyon, A. Pavao, A. Yale, and K. P. Bennett, "Synthetic event time series health data generation," *arXiv preprint arXiv:1911.06411*, 2019.

[11] D. Parthasarathy, K. Bäckstrom, J. Henriksson, and S. Einarsdóttir, "Controlled time series generation for automotive software-in-the-loop testing using gans," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2020, pp. 39–46.

[12] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," *arXiv preprint arXiv:1706.02633*, 2017.

[13] E. Brophy, Z. Wang, and T. E. Ward, "Quick and easy time series generation with established image-based gans," *arXiv preprint arXiv:1902.05624*, 2019.

[14] D. Zhang, M. Ma, and L. Xia, "A comprehensive review on gans for time-series signals," *Neural Computing and Applications*, pp. 1–21, 2022.

[15] O. O. Aremu, D. Hyland-Wood, and P. R. McAree, "A machine learning approach to circumventing the curse of dimensionality in discontinuous time series machine data," *Reliability Engineering & System Safety*, vol. 195, p. 106706, 2020.

[16] D. J. MacKay, D. J. Mac Kay *et al.*, *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[17] F. Nielsen, "On the jensen–shannon symmetrization of distances relying on abstract means," *Entropy*, vol. 21, no. 5, p. 485, 2019.

[18] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015, http://www.neuralnetworksanddeeplearning.com.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] Y. Ma and Y. Fu, *Manifold learning theory and applications*. CRC press Boca Raton, 2012, vol. 434.

[21] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2794–2802.

[22] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[23] W. McKinney, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, S. van der Walt and J. Millman, Eds., 2010, pp. 51 – 56.

[24] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[25] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" *Advances in neural information processing systems*, vol. 31, 2018.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY