

# Quant GANs: Deep Generation of Financial Time Series

Magnus Wiese<sup>\*1, 2</sup>      Robert Knobloch<sup>2</sup>      Ralf Korn<sup>1, 2</sup>  
Peter Kretschmer<sup>1, 2</sup>

<sup>1</sup>TU Kaiserslautern, Gottlieb-Daimler-Straße 48, 67663 Kaiserslautern, Germany

<sup>2</sup>Fraunhofer ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

July 17, 2019

## Abstract

Modeling financial time series by stochastic processes is a challenging task and a central area of research in financial mathematics. In this paper, we break through this barrier and present Quant GANs, a data-driven model which is inspired by the recent success of generative adversarial networks (GANs). Quant GANs consist of a generator and discriminator function which utilize temporal convolutional networks (TCNs) and thereby achieve to capture longer-ranging dependencies such as the presence of volatility clusters. Furthermore, the generator function is explicitly constructed such that the induced stochastic process allows a transition to its risk-neutral distribution. Our numerical results highlight that distributional properties for small and large lags are in an excellent agreement and dependence properties such as volatility clusters, leverage effects, and serial autocorrelations can be generated by the generator function of Quant GANs, demonstrably in high fidelity.

## 1 Introduction

Since the ground-breaking results of AlexNet [27] at the ImageNet competition deep neural networks excel in various areas [13, 18, 31, 36], even surpassing human-level performance [19, 35]. While applications in image analysis have already become standard practice, the usage of deep neural nets in finance is still in early stages. To cite a few examples, neural networks are used to hedge large portfolios of derivatives [6], solve optimal stopping problems [2] or detect anomalies in accounting data [34].

---

\*Corresponding author: [quant.gans@gmail.com](mailto:quant.gans@gmail.com)

In this paper, we shall look at the task of simulating stock price evolutions by introducing Quant GANs. Quant GANs are based on the application of generative adversarial networks (GANs) and located between pure data-based approaches, such as historical simulation, and model-driven methods such as Monte Carlo simulation assuming an underlying stock price model like the Black Scholes model, the Heston stochastic volatility model or Lévy process based modeling.

Applying the GAN concept to time series modeling comes with several potential pitfalls. Most importantly, the question regarding a reasonable network architecture has to be answered. There exist promising approaches in the literature aiming at applying the GAN idea to time series [12], recently also with a focus on financial time series, see [26, 37, 42, 43]. However, our numerical analysis suggests that the architectures do not work as well as the proposed Quant GAN model constructed in the present paper. We believe that our paper contributes to increasing the popularity of generative models as a potential substitute for Monte Carlo simulations in various contexts, specifically related to applications in a financial setting.

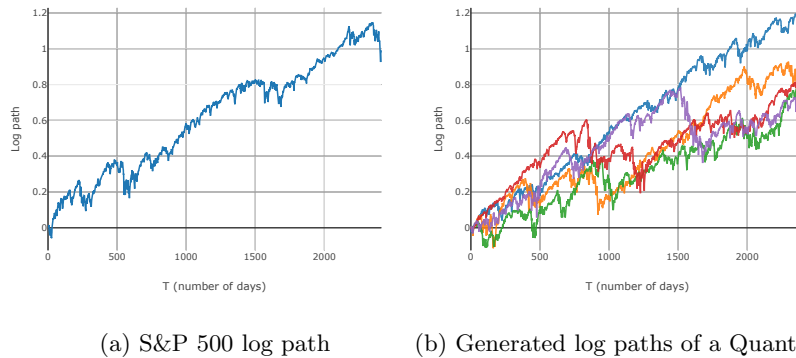


Figure 1: Comparison of a S&P 500 log path (a) with generated log paths of a calibrated Quant GAN (b).

Using two different neural networks as opponents is the fundamental principle of GANs. While one neural network, the so-called generator, is responsible for the generation of stock price paths, the second one, the discriminator, has to judge if the generated paths are artificial or seem to be generated from the same underlying distribution as the data (i.e. the past prices). Training both networks simultaneously is the main challenge of this concept and recent advances in GAN optimization allow the generator to nicely approximate the underlying data distribution [5, 29].

In contrast to the classic approach [3, 4, 20] where the financial time series is modeled by assuming that the log spot price inherits the dynamics of a designated

stochastic process, we directly try to learn the characteristics from the data. This is a fundamental difference as only very weak modeling assumptions are required.

In the spirit of GARCH processes, the generator framework used in Quant GANs is called *Stochastic Volatility Neural Network (SVNN)* and consists of a volatility, drift and an innovation network. The volatility and drift network is represented by a temporal convolutional network (TCN) which is particularly suited for modeling long-range dependencies and allows for parallelization. Moreover, SVNNs are constructed such that the generated paths can be evaluated under their risk-neutral distribution and, as a special case, constrained to exhibit conditionally normal log returns.

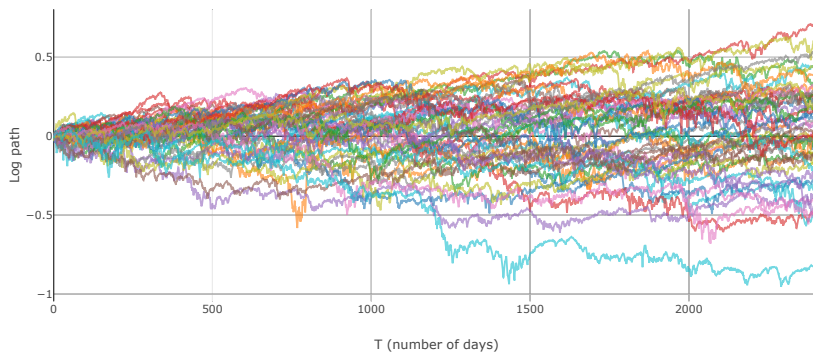


Figure 2: Fifty daily log paths without drift generated by a Quant GAN calibrated on the daily log returns of the S&P 500 index from 3. June 2009 to 2. January 2019. Distributional and dependence properties can be closely approximated.

One of our main contributions is the rigorous mathematical definition of TCNs for the first time in the literature. As this definition requires the use of heavy notation, we will illustrate it by simple examples and graphical representations. This should help to make the definition accessible for mathematicians that are not familiar with the specialized language of neural nets.

Another task is the pre- and postprocessing of raw financial data in order to train the discriminator. For this, we will describe a detailed pipeline making training possible. An innovative use of the Lambert W transform will play an essential role for generating heavy tailed stock price returns by using methods for normalized Gaussian data.

Finally, we demonstrate the usefulness of Quant GANs by applying it to real S&P-500 data.

## 2 Structure

In Section 3 the main contributions of this paper will be summarized. In the following Section 4 an overview of the stylized facts of asset returns is given. Afterward, the required neural network topologies used to construct SVNNs, such as the multilayer perceptron (MLP) and the TCN, are introduced in Section 5. In Section 6, we define GANs rigorously for random variables and then extend them to the setting of stochastic processes by using TCNs as the underlying generator and discriminator.

With sufficient background knowledge on neural nets and stylized facts, we will turn to the problem of generating log returns by using SVNNs in Section 7 and prove theoretical results for the proposed model. After having introduced SVNNs we define our preprocessing pipeline in Section 8 in order to train the proposed model with a log return series. Last, Section 9 provides numerical results that demonstrate that Quant GANs achieve to approximate distributional as well as dependence properties of the S&P 500 closely.

## 3 Main Contributions

In this paper we provide formal definitions of a novel neural network architecture carefully tailored for the task of learning the underlying dynamics of stationary stochastic processes. Furthermore, we will introduce the *Quant GAN*, a TCN based log return model trained using the GAN formalism, for which an explicit transition to its risk-neutral distribution will be derived as well as theoretical results regarding the generation of unbounded moments which relates to current work on modeling tails with generative networks [40]. Moreover, we will motivate how this model naturally extends well-known stochastic volatility models. Finally, in a numerical study we will demonstrate that the generation of daily log return series by Quant GANs outperform classical approaches like *GARCH*-models.

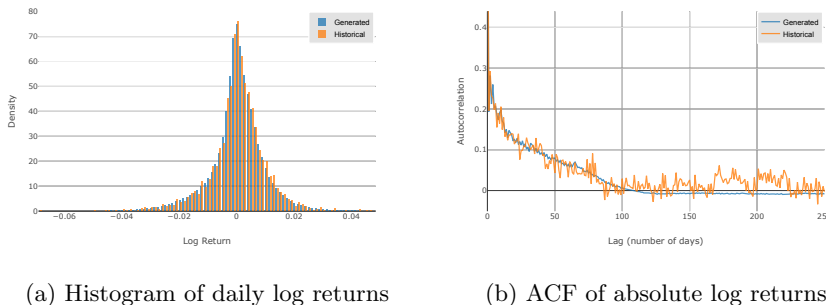


Figure 3: Figures (a) and (b) display a comparison of the generated Quant GAN (blue) and empirical S&P 500 (orange) samples.

## 4 Background on Financial Time Series

For investment decisions or option pricing purposes the absolute stock price is of minor importance. It is the relative return – either in form of  $R_t = (S_t - S_{t-1})/S_{t-1}$  or as log return  $R_t = \log(S_t/S_{t-1})$  – that is used to evaluate the performance of a stock over a certain period (here for a day, a month or a year, depending on the used time scale). Therefore, the generation of asset returns is the main objective of this paper.

The characteristic properties of asset returns are well-studied. The most important such properties are called their *stylized facts*. These properties will also be used later to judge the quality of the generated asset returns. A list of the most important stylized facts include (see e.g. [7, 10]):

- asset returns admit heavier tails than the normal distribution,
- the distribution of asset returns seems to be more peaked than the normal one,
- asset returns admit phases of high activity and low activity in terms of price changes, an effect which is called *volatility clustering*,
- the volatility of asset returns is negatively correlated with the return process, an effect named *leverage effect*,
- empirical asset returns are considered to be uncorrelated but not independent.

All those stylized facts are not shared by the classic Black-Scholes model, which is based on the assumption of normally distributed log returns.

## 5 Neural Network Topologies

In this section we introduce neural network topologies that are essential to construct stochastic volatility neural networks. We begin with the most basic model used in deep learning, namely the *multilayer perceptron (MLP)*. Afterward, we provide a formal definition of *temporal convolutional networks (TCNs)* which are also known as *WaveNets* [31]. TCNs are convolutional architectures that achieve state-of-the-art results in the sequence domain for processing time series with long ranging dependencies [1]. Roughly speaking, TCNs are an approach that aims to replace Recurrent Neural Networks (RNNs) by using Convolutional Neural Networks (CNNs) and prove to be more powerful for our purpose.

### 5.1 Multilayer Perceptrons

The MLP lies at the core of deep learning models and is constructed by composing affine transformations with so-called *activation functions*. Figure 4 depicts the construction of an MLP with 2 hidden layers where the inputs are 3-dimensional

and the output is 1-dimensional. We begin with the activation function as the crucial ingredient and then define the MLP formally.

**Definition 5.1** (Activation Function). A function  $\phi : \mathbb{R} \rightarrow \mathbb{R}$  that is Lipschitz continuous, monotonic and satisfies  $\phi(0) = 0$  is called *activation function*.

*Remark 5.2.* Definition 5.1 comprises a large class of functions used in the deep learning literature. Examples of activation functions in the sense of Definition 5.1 include rectifier linear units (*ReLU*s) [30], parametric ReLUs (*PReLU*s) [19], MaxOut [17] and a vast amount of other functions in the literature [9, 25, 28, 41]. Note that the property  $\phi(0) = 0$  is desirable during stochastic gradient descent (cf. [24]).

**Definition 5.3** (Multilayer Perceptron). Let  $L, N_0, \dots, N_{L+1} \in \mathbb{N}$ ,  $\phi$  an activation function,  $\Theta$  an Euclidean vector space and for any  $l \in \{1, \dots, L+1\}$  let  $a_l : \mathbb{R}^{N_{l-1}} \rightarrow \mathbb{R}^{N_l}$  be an affine mapping. A function  $f : \mathbb{R}^{N_0} \times \Theta \rightarrow \mathbb{R}^{N_{L+1}}$ , defined by

$$f(x, \theta) = a_{L+1} \circ f_L \circ \dots \circ f_1(x),$$

where  $\circ$  denotes the composition operator and

$$f_l = \phi \circ a_l \quad \text{for all } l \in \{1, \dots, L\}$$

and  $\phi$  being applied component-wise, is called a *multilayer perceptron with  $L$  hidden layers*. In this setting  $N_0$  represents the *input dimension*,  $N_{L+1}$  the *output dimension*,  $N_1, \dots, N_L$  the *hidden dimensions* and  $a_{L+1}$  the *output layer*. Furthermore, for any  $l \in \{1, \dots, L+1\}$  the function  $a_l$  takes the form  $a_l : x \mapsto W^{(l)}x + b^{(l)}$  for some *weight matrix*  $W^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$  and *bias*  $b^{(l)} \in \mathbb{R}^{N_l}$ . With this representation, the MLP's parameters are defined by

$$\theta := \left( W^{(1)}, \dots, W^{(L+1)}, b^{(1)}, \dots, b^{(L+1)} \right) \in \Theta.$$

We denote the class of MLPs with  $L$  hidden layers mapping from  $\mathbb{R}^{d_0}$  to  $\mathbb{R}^{d_1}$  by  $\text{MLP}_{d_0, d_1, L}$ .

*Remark 5.4.* In future derivations we will call a function  $f : \mathbb{R}^{d_0} \times \Theta \rightarrow \mathbb{R}^{d_1}$  with parameter space  $\Theta$  a *network* if it is Lipschitz continuous.

A well-known theorem that justifies the high applicability of multilayer perceptrons is the so-called *universal approximation theorem* of one-layer perceptrons. For completeness, we state Theorem 1 and Theorem 2 from [22] in the form as is given in [6, Theorem 4.2].

**Theorem 5.5** (Universal Approximation Theorem). *Assume the activation function  $\phi$  to be bounded and non-constant. The following statements hold:*

- For any finite measure  $\mu$  on  $(\mathbb{R}^{d_0}, \mathcal{B}(\mathbb{R}^{d_0}))$  and  $1 \leq p < \infty$ , the set  $\text{MLP}_{d_0, 1, 1}$  is dense in  $L^p(\mathbb{R}^{d_0}, \mu)$ .
- If  $\phi$  is additionally continuous, the set  $\text{MLP}_{d_0, 1, 1}$  is dense in  $C(\mathbb{R}^{d_0})$  with respect to the topology of uniform convergence on compact sets.

Note that this easily carries over to the case of MLPs with output dimension  $d_1 > 1$  and more than one hidden layer.

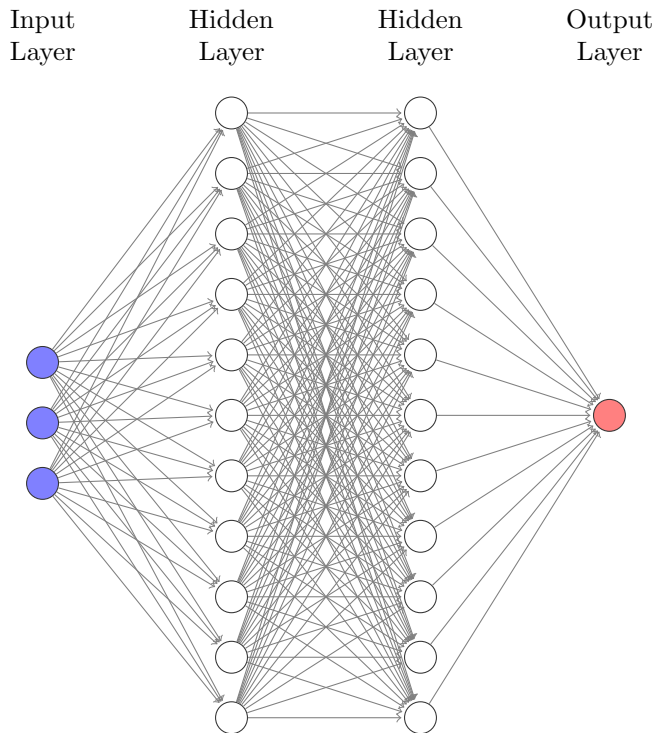


Figure 4: 2-layer MLP with input dimension  $N_0 = 3$ , output dimension  $N_3 = 1$  and hidden dimensions  $N_1 = N_2 = 11$ .

## 5.2 Temporal Convolutional Networks

It is a stylized fact that the autocorrelation function of absolute returns decays slowly as a function of the time lag, which indicates the presence of long range dependencies. For this reason, the log return process is often decomposed into a stochastic volatility and an innovations process. In order to model stochastic volatility, we propose the use of TCNs.

TCNs are convolutional architectures which have recently shown to be competitive on many sequence-related modeling tasks [1]. In particular, empirical results suggest that TCNs are able to capture long range dependencies in sequences more effectively than well-known recurrent architectures [15, Chapter 10] such as the LSTM [21] or the GRU [8]. One of the main advantages of TCNs compared to RNNs is the absence of exponentially vanishing and exploding gradients through time [32], which is one of the main issues why RNNs are difficult to optimize. Although LSTMs address this issue by using gated activations, empirical studies show that TCNs are comparatively more capable [1]. Last, in the context of stochastic process generation TCNs are beneficial as they can be used to approximate stationary processes very efficiently.

The construction of TCNs is simple. The crucial ingredient are so-called *dilated causal convolutions*. Causal convolutions are convolutions, where the output only depends on past sequence elements. Dilated convolutions, also known as *atrous* convolutions, are convolutions “with holes”. Figure 6 illustrates a TCN with 4 hidden layers, kernel size  $K = 2$  and dilation factor  $D = 1$  and Figure 7 depicts a TCN with  $K = D = 2$ , where the dilation increases in every layer by a factor of two. Comparing both networks it becomes clear that the use of increasing dilations in each layer makes it possible to model longer sequences and hence longer ranging dependencies.

In what follows, we define the TCN as well as related concepts formally. For the rest of this section, let  $N_I, N_O, K, D, T \in \mathbb{N}$ . Let us begin by defining the convolution of a time series and a weight matrix with dilation  $D$ .

**Definition 5.6** ( $*_D$  Operator). Let  $X \in \mathbb{R}^{T \times N_I}$  be an  $N_I$ -variate sequence of length  $T$  and  $W \in \mathbb{R}^{K \times N_I \times N_O}$  a tensor. Then for  $t \in \{D(K-1) + 1, \dots, T\}$  and  $m \in \{1, \dots, N_O\}$  the operator  $*_D$ , defined by

$$(W *_D X)_{t,m} := \sum_{i=1}^K \sum_{j=1}^{N_I} W_{i,j,m} \cdot X_{t-D(K-i),j} ,$$

is called *dilated causal convolutional operator* with *dilation*  $D$  and kernel size  $K$ .

A visualization of the operator for different dilations and kernel sizes is given in Figure 5.

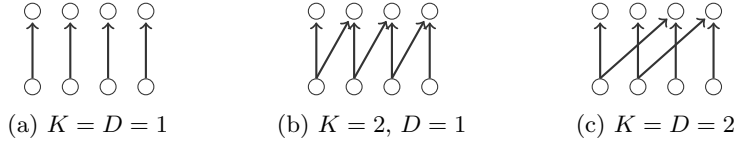


Figure 5: Dilated causal convolutional operator for different dilations  $D$  and kernel sizes  $K$

By using the notation from above we can immediately define a causal convolutional layer with dilation  $D$ .

**Definition 5.7** (Causal Convolutional Layer). Let  $W$  be as in Definition 5.6 and  $b \in \mathbb{R}^{N_O}$ . A function

$$w : \mathbb{R}^{T \times N_I} \rightarrow \mathbb{R}^{(T-D(K-1)) \times N_O}$$

defined for  $t \in \{D(K-1) + 1, \dots, T\}$  and  $m \in \{1, \dots, N_O\}$  by

$$w(X)_{t,m} := (W *_D X)_{t,m} + b_m$$

is called *causal convolutional layer with dilation*  $D$ .



*Remark 5.8.* The quadruple  $(N_I, N_O, K, D)$  will be called the *arguments* of a causal convolution  $w$  and represent the *input dimension*, *output dimension*, *kernel size*, and *dilation*, respectively.

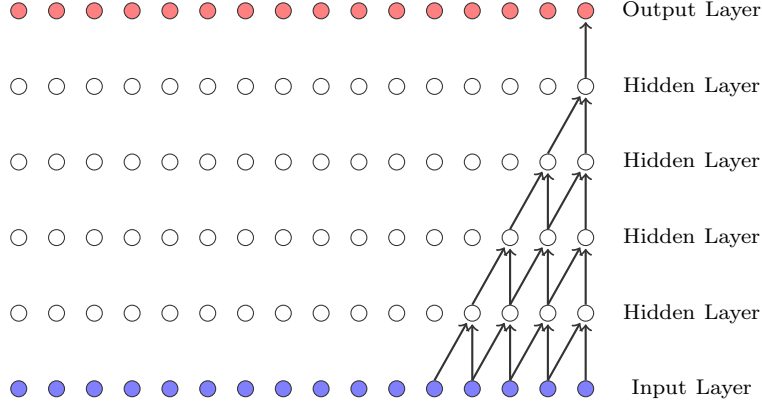


Figure 6: Vanilla TCN with 4 hidden layers, kernel size  $K = 2$  and dilation factor  $D = 1$  (cf. [31]).

**Example 5.9** ( $1 \times 1$  convolutional layer). Let  $X \in \mathbb{R}^{T \times N_I}$  be an  $N_I$ -variate sequence and  $w : \mathbb{R}^{T \times N_I} \rightarrow \mathbb{R}^{T \times N_O}$  a causal convolutional layer with arguments  $(N_I, N_O, 1, 1)$ . We call such a layer a  $1 \times 1$  *convolutional layer*.<sup>1</sup>

In the previous section we constructed MLPs by composing affine transformations with activation functions. The *vanilla TCN* construction follows a similar approach: causal convolutional layers are composed with activation functions. In order to allow for more expressive transformations we will generalize the vanilla TCN construction by introducing *block modules*.

**Definition 5.10** (Block Module). Let  $S \in \mathbb{N}$ . A function  $\psi : \mathbb{R}^{T \times N_I} \rightarrow \mathbb{R}^{(T-S) \times N_O}$  that is Lipschitz continuous is called *block module* with arguments  $(N_I, N_O, S)$ .

With the above definitions the TCN and vanilla TCN can be defined formally. The TCN follows a general block module construction, whereas the vanilla TCN is constructed similarly to the MLP by composing dilated causal convolutions with activation functions. For completeness, both definitions are given below.

**Definition 5.11** (Temporal Convolutional Network). Let  $T_0, L, N_0, \dots, N_{L+1} \in \mathbb{N}$ . Moreover, for  $l \in \{1, \dots, L\}$  let  $S_l \in \mathbb{N}$  such that  $\sum_{l=1}^L S_l \leq T_0 - 1$ . Hence, for  $T_l := T_{l-1} - S_l$  it holds

$$T_L = T_0 - \sum_{l=1}^L S_l \geq 1. \quad (1)$$

<sup>1</sup>Note that using a  $1 \times 1$  convolution is equivalent to applying an affine transformation along the time (first) dimension of  $X$ .

Furthermore, let  $\psi_l : \mathbb{R}^{T_{l-1} \times N_{l-1}} \rightarrow \mathbb{R}^{T_l \times N_l}$  for  $l \in \{1, \dots, L\}$  represent block modules and  $w : \mathbb{R}^{T_L \times N_L} \rightarrow \mathbb{R}^{T_L \times N_{L+1}}$  a  $1 \times 1$  convolutional layer. A function  $f : \mathbb{R}^{T_0 \times N_0} \times \Theta \rightarrow \mathbb{R}^{T_L \times N_{L+1}}$ , defined by

$$f(X, \theta) := w \circ \psi_L \circ \dots \circ \psi_1(X) ,$$

is called *temporal convolutional network* with  $L$  hidden layers. The class of TCNs with  $L$  hidden layers mapping from  $\mathbb{R}^{d_0}$  to  $\mathbb{R}^{d_1}$  will be denoted by  $\text{TCN}_{d_0, d_1, L}$  ( $d_0 = N_0, d_1 = N_{L+1}$ ).

**Definition 5.12** (Vanilla TCN). Let  $f \in \text{TCN}_{N_0, N_{L+1}, L}$  such that for all  $l \in \{1, \dots, L\}$  each block module  $\psi_l$  is defined as a composition of a causal convolutional layer  $w_l$  with arguments  $(N_{l-1}, N_l, K_l, D_l)$  and an activation function  $\phi$ , i.e.  $\psi_l = \phi \circ w_l$ . Then we call  $f : \mathbb{R}^{T_0 \times N_0} \times \Theta \rightarrow \mathbb{R}^{T_L \times N_{L+1}}$  a *vanilla TCN*. Moreover, if  $D_l = D^{l-1}$  for all  $l \in \{1, \dots, L\}$ , we call  $f$  a *vanilla TCN with dilation factor  $D$* . Whenever  $K_l = K$  for all  $l \in \{1, \dots, L\}$ , we say that  $f$  has *kernel size  $K$* .

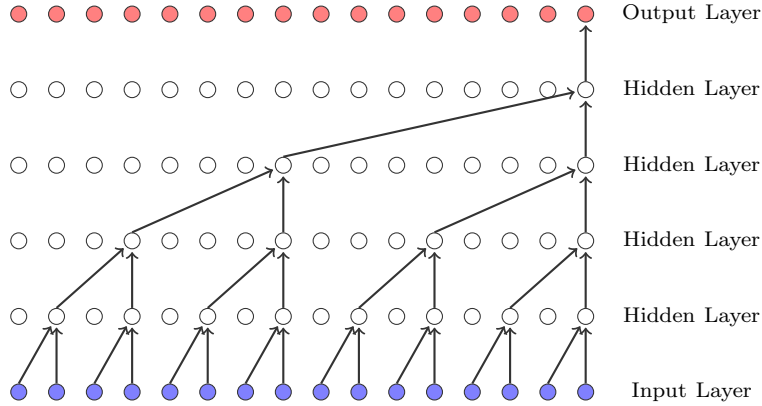


Figure 7: Vanilla TCN with 4 hidden layers, kernel size  $K = 2$  and dilation factor  $D = 2$  (cf. [31]).

TCN's ability to model long range dependencies becomes ultimately apparent when comparing the two vanilla TCNs displayed in Figure 6 and Figure 7. In Figure 6, the network is a function of 5 sequence elements, whereas the network in Figure 7 has 16 sequence elements as input. We call the number of sequence elements that the TCN can capture the *receptive field size* and give a formal definition below:

**Definition 5.13** (Receptive Field Size). Let  $f \in \text{TCN}_{d_0, d_1, L}$  and let  $S_1, \dots, S_L$  be as in Definition 5.11. The constant

$$R := 1 + \sum_{l=1}^L S_l$$

is called *receptive field size (RFS)*.

*Remark 5.14.* For vanilla TCNs with kernel size  $K$  and dilation factor  $D > 1$ , the RFS  $R$  can be computed using the formula for the sum of a geometric sequence with finite length:

$$R = 1 + (K - 1) \cdot \left( \frac{D^L - 1}{D - 1} \right) .$$

Therefore, the RFS  $R$  is the minimum initial time dimension  $T_0$  of an input  $X \in \mathbb{R}^{T_0 \times N_0}$  such that the sequence  $X$  can be inferred (compare Equation 1).

*Remark 5.15.* Note that an MLP can be seen as a vanilla TCN in which each causal convolution is a  $1 \times 1$  convolution. Thus, MLPs are a subclass of TCNs with an RFS equal to 1.

The idea of residual connections can also be utilized.

**Definition 5.16** (TCN with Skip Connections). Assume the notation from Definition 5.11 and for  $N_{skip} \in \mathbb{N}$  let

$$\gamma_l : \mathbb{R}^{T_{l-1} \times N_{l-1}} \rightarrow \mathbb{R}^{T_l \times N_l} \times \mathbb{R}^{T_L \times N_{skip}} \quad \text{for } l \in \{1, \dots, L\}$$

denote block modules. Moreover, let  $\gamma$  be a block module with arguments  $(N_{skip}, N_{L+1}, 0)$ . If the output  $Y \in \mathbb{R}^{T_L \times N_{L+1}}$  of a TCN  $f : \mathbb{R}^{T_0 \times N_0} \times \Theta \rightarrow \mathbb{R}^{T_L \times N_{L+1}}$  is defined recursively by

$$\begin{aligned} (X^{(l)}, H^{(l)}) &= \gamma_l (X^{(l-1)}) \quad \text{for } l \in \{1, \dots, L\} \\ Y &= \gamma \left( \sum_{l=1}^L H^{(l)} \right) , \end{aligned}$$

where  $X^{(0)} \in \mathbb{R}^{T_0 \times N_0}$ , then  $f$  is called a *temporal convolutional network with skip connections*.

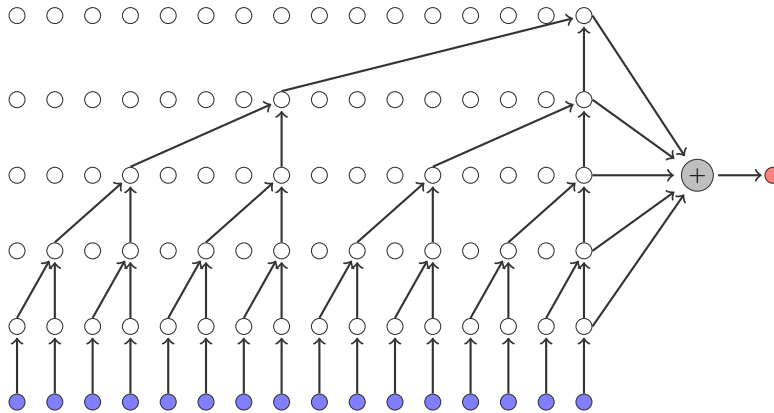


Figure 8: Vanilla TCN with skip connections and a  $1 \times 1$  convolutional layer.

One of the downsides of TCNs is that the length of time series to be processed is restricted to the TCN’s receptive field size. Hence, in order to model long range dependencies we require a RFS  $R \gg 1$  leading to computational bottlenecks. Furthermore, it becomes questionable if such large networks can be trained to model something meaningful and if sufficient data is available. Although interesting extensions to TCNs with skip modules exist to model long range dependencies, we leave it as future work to develop these methods.

The neural network topologies introduced in this section are the core components used to achieve the results presented in Section 9. In order to train these networks to generate time series we now formulate GANs in the setting of random variables and stochastic processes.

## 6 Generative Adversarial Networks

Generative adversarial networks (*GANs*) [16] are a relatively new class of algorithms to learn the distribution of a realization of a random variable, i.e. a dataset or the distribution of a random variable itself. GANs were originally formulated from a game-theoretic perspective. Prior to introducing the adversarial modeling framework, let us introduce some notation and concepts for clarity.

**Notation 6.1.** Throughout this chapter let  $N_Z, N_X \in \mathbb{N}$  such that  $N_Z \leq N_X$  and  $\mathcal{Z} := \mathbb{R}^{N_Z}$  and  $\mathcal{X} := \mathbb{R}^{N_X}$  represent two Euclidean vector spaces. If not stated otherwise, the used norm will be clear by context. Moreover, throughout this section let  $(\Omega, \mathcal{F}, \mathbb{P})$  be a probability space and  $(\mathcal{Z}, \mathfrak{Z})$  and  $(\mathcal{X}, \mathfrak{X})$  be two measurable spaces. Furthermore, assume that  $X$  and  $Z$  are  $\mathcal{X}$  and  $\mathcal{Z}$ -valued random variables respectively. The distribution of a random variable  $Y$  will be denoted by  $\mathbb{P}_Y$ .

### 6.1 Formulation for Random Variables

In the context of GANs,  $(\mathcal{Z}, \mathfrak{Z})$  and  $(\mathcal{X}, \mathfrak{X})$  are called the *latent* and *data measure space*, respectively. The random variable  $Z$  represents the *noise prior* and  $X$  the *targeted (or data) random variable*. The goal of GANs is to train a network  $g : \mathcal{Z} \times \Theta^{(g)} \rightarrow \mathcal{X}$  such that the induced random variable  $g_\theta(Z) := g_\theta \circ Z$  for some parameter  $\theta \in \Theta^{(g)}$  and the targeted random variable  $X$  are equal in distribution, i.e.  $g_\theta(Z) \stackrel{d}{=} X$ . In order to achieve this, GOODFELLOW et al. proposed the adversarial modeling framework for deep neural networks and introduced the *generator* and *discriminator* as follows:

**Definition 6.2** (Generator). Let  $g : \mathcal{Z} \times \Theta^{(g)} \rightarrow \mathcal{X}$  be a network with parameter space  $\Theta^{(g)}$ . The random variable  $\tilde{X}$ , defined by

$$\begin{aligned} \tilde{X} : \Omega \times \Theta^{(g)} &\rightarrow \mathcal{X} \\ (\omega, \theta) &\mapsto g_\theta(Z(\omega)) , \end{aligned}$$

is called the *generated random variable*. Furthermore, the network  $g$  is called *generator* and  $\tilde{X}_\theta$  the *generated random variable with parameter  $\theta$* .<sup>2</sup>

**Definition 6.3** (Discriminator). Let  $\tilde{d} : \mathcal{X} \times \Theta^{(d)} \rightarrow \mathbb{R}$  be a network with parameters  $\eta \in \Theta^{(d)}$  and  $\sigma : \mathbb{R} \rightarrow [0, 1] : x \mapsto \frac{1}{1+e^{-x}}$  be the sigmoid function. A function  $d : \mathcal{X} \times \Theta^{(d)} \rightarrow [0, 1]$  defined as  $d : (x, \eta) \mapsto \sigma \circ \tilde{d}_\eta(x)$  is called a *discriminator*.

**Notation 6.4.** Throughout this section we assume the notation used in Definition 6.2 and Definition 6.3. A batch (set) of  $M \in \mathbb{N}$  realizations of a random variable  $Y$  will be denoted by

$$\{y_i\}_{i=1}^M := \{Y(\omega_i)\}_{i=1}^M$$

for  $\omega_1, \dots, \omega_M \in \Omega$ .

In the adversarial modeling framework two agents, the generator and the discriminator (also referred to as the adversary), are contesting with each other in a game-theoretic zero-sum game. Roughly speaking, the generator aims at generating samples  $\{\tilde{x}_{\theta,i}\}_{i=1}^M$  such that the discriminator can not distinguish whether the realizations were sampled from the target or generator distribution. In other words, the discriminator  $d_\eta : \mathcal{X} \rightarrow [0, 1]$  acts as a classifier, trying to approximate the probability that a sample  $x \in \mathcal{X}$  comes from the generator or the dataset.

The optimization of GANs is formulated in two steps. First, the discriminator's parameters  $\eta \in \Theta^{(d)}$  are optimized to maximize the probability of correctly classifying real and generated samples. Hence, maximizing the function

$$\begin{aligned} \mathcal{L}(\theta, \eta) &:= \mathbb{E}[\log(d_\eta(X))] + \mathbb{E}[\log(1 - d_\eta(g_\theta(Z)))] \\ &= \mathbb{E}[\log(d_\eta(X))] + \mathbb{E}\left[\log(1 - d_\eta(\tilde{X}_\theta))\right]. \end{aligned}$$

In the second step, the generator's parameters  $\theta \in \Theta^{(g)}$  are trained to minimize the probability of generated samples being identified as such and not from the data distribution. In summary, we receive the min-max game

$$\min_{\theta \in \Theta^{(g)}} \max_{\eta \in \Theta^{(d)}} \mathcal{L}(\theta, \eta),$$

which we will refer to as the *GAN objective*.

**Training** The generator's and discriminator's parameters  $(\theta, \eta)$  are trained by alternating the computation of their stochastic gradients  $\nabla_\eta \mathcal{L}(\theta, \eta)$  and  $\nabla_\theta \mathcal{L}(\theta, \eta)$  and updating their respective parameters. To get a close approximation of the optimal discriminator  $d^*$  [16, Proposition 1] it is common to compute the discriminators stochastic gradient multiple times and ascent the parameters  $\eta$ . Algorithm 1 describes the procedure in detail.

---

<sup>2</sup>The subscript  $\theta$  of  $\tilde{X}_\theta$  represents the dependency with respect to the neural operator's parameters  $\theta$ .

---

**Algorithm 1** Stochastic gradient optimization of a generative adversarial network (cf. [16]).

---

**INPUT:** generator  $g$ , discriminator  $d$ , mini-batch size  $M \in \mathbb{N}$ , generator learning rate  $\alpha_g$ , discriminator learning rate  $\alpha_d$ , number of discriminator optimization steps  $k$

**OUTPUT:** parameters  $(\theta, \eta)$

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

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

    Sample  $M$  samples from  $\tilde{X}_\theta$ :  $\{\tilde{x}_{\theta,i}\}_{i=1}^M$ .

    Sample  $M$  samples from  $X$ :  $\{x_i\}_{i=1}^M$ .

    Compute and store the gradient

$$\Delta_\eta \leftarrow \nabla_\eta \frac{1}{M} \sum_{i=1}^M \log(d(x_i)) + \log(1 - d(\tilde{x}_{\theta,i})) .$$

    Ascent the discriminator's parameters:  $\eta \leftarrow \eta + \alpha_d \cdot \Delta_\eta$ .

**end for**

  Sample  $M$  samples from  $\tilde{X}_\theta$ :  $\{\tilde{x}_{\theta,i}\}_{i=1}^M$ .

  Compute and store the gradient

$$\Delta_\theta \leftarrow \nabla_\theta \frac{1}{m} \sum_{i=1}^m \log(d(\tilde{x}_{\theta,i})) .$$

  Descent the generator's parameters:  $\theta \leftarrow \theta - \alpha_g \cdot \Delta_\theta$ .

**end while**

---

## 6.2 Formulation for Stochastic Processes

We now consider the formulation of GANs in the context of stochastic process generation by using TCNs as it turns out that the properties of TCNs are quite intriguing for this. The following notation is used for brevity.

**Notation 6.5.** Consider a stochastic process  $(X_t)_{t \in \mathbb{Z}}$  parametrized by some  $\theta \in \Theta$ . For  $s, t \in \mathbb{Z}$ ,  $s \leq t$ , we write

$$X_{s:t,\theta} := (X_{s,\theta}, \dots, X_{t,\theta})$$

and for an  $\omega$ -realization

$$X_{s:t,\theta}(\omega) := (X_{s,\theta}(\omega), \dots, X_{t,\theta}(\omega)) \in \mathcal{X}^{t-s+1}.$$

We can now introduce the concept of neural (stochastic) processes.

**Definition 6.6** (Neural Process). Let  $(Z_t)_{t \in \mathbb{Z}}$  be an i.i.d. noise process with values in  $\mathcal{Z}$  and  $g : \mathcal{Z}^{T^{(g)}} \times \Theta^{(g)} \rightarrow \mathcal{X}$  a TCN with RFS  $T^{(g)}$  and parameters  $\theta \in \Theta^{(g)}$ . A stochastic process  $\tilde{X}$ , defined by

$$\begin{aligned} \tilde{X} : \Omega \times \mathbb{Z} \times \Theta^{(g)} &\rightarrow \mathcal{X} \\ (\omega, t, \theta) &\mapsto g_\theta(Z_{t-(T^{(g)}-1):t}(\omega)) \end{aligned}$$

such that  $\tilde{X}_{t,\theta} : \Omega \rightarrow \mathcal{X}$  is a  $\mathcal{F} - \mathfrak{X}$  measurable mapping for all  $t \in \mathbb{Z}$  and  $\theta \in \Theta^{(g)}$ , is called *neural process* and will be denoted by  $\tilde{X}_\theta := (\tilde{X}_{t,\theta})_{t \in \mathbb{Z}}$ .

In the context of GANs the i.i.d. noise process  $Z = (Z_t)_{t \in \mathbb{Z}}$  from Definition 6.6 represents the noise prior. Throughout this paper we assume for simplicity that for all  $t \in \mathbb{Z}$  the random variable  $Z_t$  follows a multivariate standard normal distribution, i.e.  $Z_t \sim \mathcal{N}(0, I)$ . In particular, the neural process  $\tilde{X}_\theta = (\tilde{X}_{t,\theta})_{t \in \mathbb{Z}}$  is obtained by inferring  $Z$  through the TCN generator  $g$ .

Since  $Z$  is i.i.d. noise, we conclude that the neural process  $\tilde{X}_\theta$  is stationary. Therefore, the targeted process can only be approximated if it obeys the following assumption:

**Assumption 6.7.** The targeted process  $X = (X_t)_{t \in \mathbb{N}}$  is stationary.

However, this is not a restriction when trying to approximate the log return process as it is generally considered to be stationary.

In our GAN framework for stochastic processes the discriminator is similarly represented by a TCN  $d : \mathcal{X}^{T^{(d)}} \times \Theta^{(d)} \rightarrow [0, 1]$  with RFS  $T^{(d)}$ . With these modifications to the original GAN setting for random variables, the GAN objective for stochastic processes can be formulated as

$$\min_{\theta \in \Theta^{(g)}} \max_{\eta \in \Theta^{(d)}} \mathcal{L}(\theta, \eta),$$

where

$$\mathcal{L}(\theta, \eta) := \mathbb{E}[\log(d_\eta(X_{1:T^{(d)}}))] + \mathbb{E}[\log(1 - d_\eta(\tilde{X}_{1:T^{(d)},\theta}))]$$

and  $X_{1:T^{(d)}}$  and  $\tilde{X}_{1:T^{(d)},\theta}$  denote the real and generated process respectively. Hence, analogue to the GAN setting for random variables the discriminator is trained to distinguish real from generated sequences, whereas the generator aims at simulating sequences which the discriminator can not distinguish from the real ones.

In order to train the generator and discriminator we proceed in a similar fashion as in the case of random variables. We sample from the generated neural process and from the target distribution  $M$ -sized batches  $\{\tilde{x}_{1:T^{(d)},\theta}^{(i)}\}_{i=1}^M$  and  $\{x_{1:T^{(d)}}^{(i)}\}_{i=1}^M$ . Each element of the batch is then inferred into the discriminator to generate a  $[0, 1]$ -valued output (the classifications), which are then averaged batch-wise to give a Monte Carlo estimate of the discriminator loss.

## 7 The Model

After having introduced the main neural network topologies in Section 5 and defined GANs in the context of stochastic process generation via TCNs in Section 6, we will now turn to the problem of generating financial time series and define the generator architecture of Quant GANs. As we have mentioned in the introduction, orthogonal to the conventional approaches of assuming pre-defined dynamics for the price processes, we learn the generating mechanism of the series by using a highly parametrized neural network. Moreover, contrary to typical recursive models, there is no burn-in period necessary.

Therefore, our proposed model makes no assumptions on the dynamics themselves (such as the GARCH or the Heston model), except the one that an optimal parameter vector  $\theta \in \Theta$  exists such that the generated neural process represents the log return process:

**Assumption 7.1.** The spot log returns  $R := (R_t)_{t \in \mathbb{N}}$  can be represented by a *log return neural process*  $R_\theta := (R_{t,\theta})_{t \in \mathbb{N}}$  for some  $\theta \in \Theta$ .

We start by defining the log return neural process (log return NP) in the first part of this section. The remainder of this section is devoted to answering the following questions:

- Section 7.2: How heavy are the tails generated by a log returns NP?
- Section 7.3: Can the risk-neutral distribution of log return NPs be derived?
- Section 7.4: Can log return NPs be seen as a natural extension of already existing time-series models?

### 7.1 Log Return Neural Processes

Log return NPs are motivated by the volatility-innovation decomposition of various stochastic volatility models used in practice [4, 10, 20, 38]. The construction consists of three main components:

- the latent process  $Z := (Z_t)_{t \in \mathbb{Z}}$ ,  $\mathcal{Z}$ -valued i.i.d. Gaussian noise,
- the *volatility and drift TCN*  $g^{(\sigma,\mu)} : \mathcal{Z}^{T^{(g)}} \times \Theta^{(\sigma,\mu)} \rightarrow \mathcal{X}^2$  with receptive field size  $T^{(g)}$ ,
- and the *innovation network*  $g^{(\epsilon)} : \mathcal{Z} \times \Theta^{(\epsilon)} \rightarrow \mathcal{X}$ .

The volatility and drift TCN receives  $T^{(g)}$  latent variables  $Z_{(t-1)-(T^{(g)}-1):(t-1)}$  from the past whereas the innovation network receives the latent variable  $Z_t$ . The outputs are then combined element-wise. The network which induces a log return NP will be called *stochastic volatility neural network (SVNN)* and is defined below. Figure 9 illustrates the architecture.



**Definition 7.2** (Log Return Neural Process). Let  $Z = (Z_t)_{t \in \mathbb{Z}}$  be  $\mathcal{Z}$ -valued i.i.d. Gaussian noise,  $g^{(\sigma, \mu)} : \mathcal{Z}^{T^{(g)}} \times \Theta^{(\sigma, \mu)} \rightarrow \mathcal{X}^2$  a TCN with RFS  $T^{(g)}$  and parameters  $\nu \in \Theta^{(\sigma, \mu)}$ . Moreover, let  $g^{(\epsilon)} : \mathcal{Z} \times \Theta^{(\epsilon)} \rightarrow \mathcal{X}$  be a network with parameters  $\gamma \in \Theta^{(\epsilon)}$ . A stochastic process  $R$ , defined by

$$R : \Omega \times \mathbb{Z} \times \Theta \rightarrow \mathcal{X} \\ (\omega, t, \theta) \mapsto [\sigma_{t, \nu} \odot \epsilon_{t, \gamma} + \mu_{t, \nu}] (\omega),$$

where  $\odot$  denotes the Hadamard product and

$$\begin{pmatrix} \sigma_{t, \nu} \\ \mu_{t, \nu} \end{pmatrix} := \begin{pmatrix} \left| g_{\nu}^{(\sigma, \mu)} (Z_{(t-1)-(T^{(g)}-1):(t-1)})_1 \right| \\ g_{\nu}^{(\sigma, \mu)} (Z_{(t-1)-(T^{(g)}-1):(t-1)})_2 \end{pmatrix} \\ \epsilon_{t, \gamma} := g_{\gamma}^{(\epsilon)} (Z_t),$$

is called *log return neural process (log return NP)*. Furthermore, the NPs  $\sigma_{\nu} := (\sigma_{t, \nu})_{t \in \mathbb{Z}}$  and  $\mu_{\nu} := (\mu_{t, \nu})_{t \in \mathbb{Z}}$  obtained by inducing the latent process through  $g^{(\sigma, \mu)}$  are called *volatility* and *drift NP*, respectively.  $\epsilon_{\gamma} := (\epsilon_{t, \gamma})_{t \in \mathbb{Z}}$  is called *innovation NP*. The generator architecture defining the log return NP is called *stochastic volatility neural network (SVNN)*.

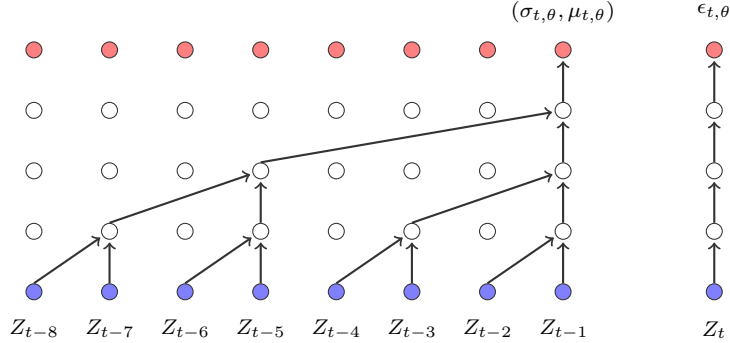


Figure 9: Structure of the SVNN architecture. The volatility and drift process are generated by inferring the latent process  $Z_{t-8:t-1}$  through the TCN, whereas the innovation is generated by inferring  $Z_t$ .

*Remark 7.3.* For simplicity, we will in the following not distinguish the different NP parameters and just write  $\theta$ .

Denote with  $(\mathcal{F}_t^Z)_{t \in \mathbb{Z}}$  the natural filtration of the latent process  $Z$ . By construction,  $\sigma_{t, \theta}$  and  $\mu_{t, \theta}$  are  $\mathcal{F}_{t-1}^Z$ -measurable and  $\epsilon_{t, \theta}$  is  $\mathcal{F}_t^Z$ -measurable. In particular, the log return NP  $R_{t, \theta}$  is  $\mathcal{F}_t^Z$ -measurable. Moreover, the random variables  $(\sigma_{t, \theta}, \mu_{t, \theta})$  and  $\epsilon_{t, \theta}$  are independent for all  $t \in \mathbb{Z}$ , since  $Z$  is i.i.d. noise. As it turns out, the proposed construction is convenient when deriving the transition to the risk-neutral distribution in Section 7.3.

## 7.2 $L^p$ -Space Characterization of $R_\theta$

It is a stylized fact that asset log returns  $R$  have heavy-tails. We inspect next whether this property can be captured by the log return NP. We first prove a result concerning generative networks in general and then conclude a Corollary for the log return NP.

**Proposition 7.4** ( $L^p$ -Characterization of Neural Nets). Let  $p \in \mathbb{N}$ ,  $Z \in L^p(\mathcal{Z})$  and  $g : \mathcal{Z} \times \Theta \rightarrow \mathcal{X}$  a network with parameters  $\theta \in \Theta$ . Then,  $g_\theta(Z) \in L^p(\mathcal{X})$ .

*Proof.* Observe that for any Lipschitz continuous function  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$  there exists a suitable constant  $L > 0$  such that

$$\|f(x) - f(0)\| \leq L \|x\| \Rightarrow \|f(x)\| \leq L \|x\| + \|f(0)\| \quad (2)$$

as  $\|x\| - \|y\| \leq \|x - y\|$  for  $x, y \in \mathbb{R}^n$ . Now, using the Lipschitz property of neural networks (cf. Remark 5.4), we can apply Equation 2 and as  $Z \in L^p(\mathcal{Z})$  we obtain

$$\begin{aligned} \mathbb{E} [\|g_\theta(Z)\|^p] &\leq \mathbb{E} [(L \|Z\| + \|g_\theta(\mathbf{0})\|)^p] \\ &= \sum_{k=0}^p \binom{p}{k} L^k \mathbb{E} [\|Z\|^k] \|g_\theta(\mathbf{0})\|^{p-k} \\ &< \infty, \end{aligned}$$

where  $L$  is the networks Lipschitz constant and  $\mathbf{0} \in \mathcal{Z}$  the zero vector. This proves the statement.  $\square$

Since we assume that our latent process  $Z$  is Gaussian i.i.d. noise and thus square-integrable, we conclude from Proposition 7.4 that mean and variance of the volatility, drift and innovation NPs are finite. Additionally, these properties carry over to the log return NP as the following Corollary proves.

**Corollary 7.5.** Let  $R_\theta$  be a log return NP parametrized by some  $\theta \in \Theta$ . Then, for all  $t \in \mathbb{Z}$  and  $p \in \mathbb{N}$  the random variable  $R_{t,\theta}$  is an element of the space  $L^p(\mathcal{X})$ .

*Proof.* The latent process  $Z$  is Gaussian i.i.d. noise. Hence, Proposition 7.4 yields  $\sigma_{t,\theta}, \epsilon_{t,\theta}, \mu_{t,\theta} \in L^p(\mathcal{X})$ . As it holds

$$\|R_{t,\theta}\|^p = \|\sigma_{t,\theta} \odot \epsilon_{t,\theta} + \mu_{t,\theta}\|^p \leq (\|\sigma_{t,\theta} \odot \epsilon_{t,\theta}\| + \|\mu_{t,\theta}\|)^p,$$

we obtain using the binomial identity

$$\begin{aligned} \|R_{t,\theta}\|_p^p &= \mathbb{E}[\|R_{t,\theta}\|^p] \\ &\leq \sum_{k=0}^p \binom{p}{k} \mathbb{E}[\|\sigma_{t,\theta} \odot \epsilon_{t,\theta}\|^k \|\mu_{t,\theta}\|^{p-k}] \\ &\leq \sum_{k=0}^p \binom{p}{k} \left( \mathbb{E} [\|\sigma_{t,\theta} \odot \epsilon_{t,\theta}\|^{2k}] \mathbb{E} [\|\mu_{t,\theta}\|^{2(p-k)}] \right)^{\frac{1}{2}} \end{aligned}$$

where the last inequality derives from Cauchy-Schwarz. Using the independence and the  $L^p$ -property of the volatility and innovation NP, we obtain for arbitrary  $q \in \mathbb{N}$

$$\mathbb{E} [\|\sigma_{t,\theta} \odot \epsilon_{t,\theta}\|^q] = \mathbb{E} \left[ \sum_{i=1}^{N_X} |\sigma_{t,\theta,i} \cdot \epsilon_{t,\theta,i}|^q \right] = \sum_{i=1}^{N_X} \mathbb{E} [|\sigma_{t,\theta,i}|^q] \mathbb{E} [|\epsilon_{t,\theta,i}|^q] < \infty$$

□

Due to the existence of all moments, we present in Section 8 a heuristic that works well to achieve empirically heavy tails. Although it remains an open question how to generate certain tail asymptotics with deep neural nets, we found that this heuristic works well when applied to TCNs.

Lastly, we motivate the choice of the latent process. We give a proof for the MLP which can be naturally extended to the setting of TCNs and especially SVNNs. The statement demonstrates that choosing the latent process as i.i.d. Gaussian noise benefits stability during optimization, i.e. back-propagation and parameter updates. Furthermore, the generated distribution is not necessarily bounded, as would be the case for uniform i.i.d. noise.

**Corollary 7.6.** *Under the assumptions of Proposition 7.4 the random variable of back-propagated gradients  $\nabla_{\theta} g_{\theta}(Z)$  is an element of the space  $L^p(\Theta)$ .*

*Proof.* Without loss of generality assume that  $\mathcal{X} = \mathbb{R}$ . Using the notation from Definition 5.3 and that  $g_{\theta}$  has  $L$  hidden layers, the gradient of  $g_{\theta}(z)$  with respect to the hidden weight matrix  $W^{(k)}$ ,  $k \leq L + 1$ , is defined for  $z \in \mathcal{Z}$  by

$$\nabla_{W^{(k)}} g_{\theta}(z) = \left( \prod_{l=k}^L D^{(l)}(z) W^{(l+1)T} \right) \otimes g_{1:k-1,\theta}(z)$$

where  $\otimes$  denotes the outer product,  $g_{1:k-1,\theta} := g_{k-1,\theta} \circ \dots \circ g_{1,\theta}$  and  $D^{(l)}(z) = \text{diag}(\phi'(W^{(l)} g_{1:l-1,\theta}(z)))$  (compare [15, Chapter 6.5]). Since the MLP is defined as a composition of Lipschitz functions Proposition 7.4 yields  $g_{1:k,\theta}(Z) \in L^p(\mathbb{R}^{N_k})$  for all  $k \leq L$ . Similarly, the boundedness of  $\phi'$  implies that for an induced matrix norm the random variable  $\|D^{(l)}(Z)\|$  is  $\mathbb{P}$ -almost surely bounded by some constant  $A > 0$  for all  $l \leq L$ . By applying both properties we obtain

$$\mathbb{E} [\|\nabla_{W^{(k)}} g_{\theta}(Z)\|^p] \leq B_k \mathbb{E} [\|g_{1:k-1,\theta}(Z)\|^p] < \infty$$

with

$$B_k := A^{p(L-k+1)} \left( \prod_{l=k}^L \|W^{(l+1)T}\| \right)^p.$$

With a similar argument one can show that the random gradients with respect to the biases  $b^{(k)}$ ,  $k = 1, \dots, L + 1$  are also an element of  $L^p$ , thus concluding the proof. □

### 7.3 Risk-Neutral representation of $R_\theta$

At this point we cannot evaluate options under a log return NP as we do not know a transition to its risk-neutral distribution. That aspect will be addressed in this section. To this end, consider a one-dimensional log return NP

$$R_{t,\theta} = \sigma_{t,\theta} \cdot \epsilon_{t,\theta} + \mu_{t,\theta} .$$

The spot prices are then defined recursively by

$$S_{t,\theta} = S_{t-1,\theta} \cdot \exp(R_{t,\theta}) \quad \text{for all } t \in \mathbb{N} ,$$

where  $S_{0,\theta} = S_0$  denotes the current price of the underlying. Moreover, assume a constant interest rate  $r$  and define the discounted stock price process  $(\tilde{S}_{t,\theta})_{t \in \mathbb{N}}$  by

$$\tilde{S}_{t,\theta} := \frac{S_{t,\theta}}{\exp(rt)} .$$

In particular, the discounted price process fulfils the recursion

$$\tilde{S}_{t,\theta} = \tilde{S}_{t-1,\theta} \cdot \exp(R_{t,\theta} - r) .$$

In its risk-neutral representation, the discounted stock price process has to be a martingale. Therefore, we can use that  $\tilde{S}_{t-1,\theta}$  is  $\mathcal{F}_{t-1}^Z$ -measurable and get

$$\begin{aligned} \mathbb{E}[\tilde{S}_{t,\theta} | \mathcal{F}_{t-1}^Z] &= \mathbb{E}[\tilde{S}_{t-1,\theta} \cdot \exp(R_{t,\theta} - r) | \mathcal{F}_{t-1}^Z] \\ &= \tilde{S}_{t-1,\theta} \cdot \exp(-r) \cdot \mathbb{E}[\exp(\sigma_{t,\theta} \cdot \epsilon_{t,\theta} + \mu_{t,\theta}) | \mathcal{F}_{t-1}^Z] . \end{aligned}$$

Hence, to obtain a martingale we have to correct for the corresponding term. Therefore, let us consider the conditional expectation in more detail. As the volatility and drift NPs are  $\mathcal{F}_{t-1}^Z$ -measurable and  $\epsilon_{t,\theta}$  is independent of  $\mathcal{F}_{t-1}^Z$ , we can write

$$\mathbb{E}[\exp(\sigma_{t,\theta} \cdot \epsilon_{t,\theta} + \mu_{t,\theta}) | \mathcal{F}_{t-1}^Z] = \mathbb{E}[\exp(\sigma \cdot \epsilon_{t,\theta} + \mu)]_{\substack{\sigma = \sigma_{t,\theta} \\ \mu = \mu_{t,\theta}}} =: h(\sigma_{t,\theta}, \mu_{t,\theta}) .$$

Depending on the innovation NP  $\epsilon_{t,\theta}$ , the function  $h$  might be given explicitly or has to be estimated using a Monte Carlo estimator.

As a result, we can define the *risk-neutral log return Neural Process*  $R_{t,\theta}^M$  as

$$R_{t,\theta}^M := R_{t,\theta} - \log(h(\sigma_{t,\theta}, \mu_{t,\theta})) + r$$

which is a corrected log return NP. The corresponding *discounted risk-neutral spot price process* is then given by the recursion

$$\tilde{S}_{t,\theta}^M = \tilde{S}_{t-1,\theta}^M \cdot \exp(R_{t,\theta}^M - r) = \tilde{S}_{t-1,\theta}^M \cdot \exp(R_{t,\theta} - \log(h(\sigma_{t,\theta}, \mu_{t,\theta})))$$

and defines a martingale.

In particular, this recursion can be solved to obtain an explicit formula for the (discounted) risk-neutral spot price process

$$\begin{aligned}\tilde{S}_{t,\theta}^M &= S_0 \cdot \exp\left(\sum_{s=1}^t [R_{s,\theta} - \log(h(\sigma_{s,\theta}, \mu_{s,\theta}))]\right) \\ S_{t,\theta}^M &= S_0 \cdot \exp\left(\sum_{s=1}^t [R_{s,\theta} - \log(h(\sigma_{s,\theta}, \mu_{s,\theta}))] + rt\right)\end{aligned}$$

It remains the problem of inferring the parameters of the underlying model. In the case of financial time series, the discriminator is used to distinguish between generated and real (observable) financial time series. What is different here is that risk-neutral asset paths are not observable. Therefore, we can not train the generator-discriminator pair in the same way as for financial time series. An approach would be a classical least square calibration by option prices, where we would use Monte Carlo of the generated risk-neutral paths as an estimate of the models option price. We leave this as future work.

## 7.4 Constrained Log Return Neural Process

An interesting application is to constrain either the volatility or the innovations NP to satisfy certain conditions. We exemplify this for the one-dimensional case, i.e  $\dim(\mathcal{X}) = 1$ , where the innovations NP is constrained to represent a standard normal distributed random variable

$$\epsilon_{t,\theta} \sim \mathcal{N}(0, 1) \quad \text{for all } t \in \mathbb{Z}.$$

In this case, the risk-neutral dynamics can be simplified, since the conditional expectation  $h(\sigma_{t,\theta}, \mu_{t,\theta})$  can be calculated explicitly:

$$h(\sigma_{t,\theta}, \mu_{t,\theta}) = \mathbb{E}[\underbrace{\exp(\sigma \cdot \epsilon_{t,\theta} + \mu)}_{\sim \mathcal{N}(\mu, \sigma^2)}]_{\substack{\sigma=\sigma_{t,\theta} \\ \mu=\mu_{t,\theta}}} = \exp\left(\mu_{t,\theta} + \frac{\sigma_{t,\theta}^2}{2}\right).$$

Hence, the risk-neutral log return NP is given by

$$R_{t,\theta}^M = \sigma_{t,\theta} \cdot \epsilon_{t,\theta} - \frac{\sigma_{t,\theta}^2}{2} + r$$

and the discounted risk-neutral price process solves the recursion

$$\tilde{S}_{t,\theta}^M = \tilde{S}_{t-1,\theta}^M \cdot \exp\left(\sigma_{t,\theta} \cdot \epsilon_{t,\theta} - \frac{\sigma_{t,\theta}^2}{2}\right).$$

In particular, solving the recursion gives the explicit representations

$$\tilde{S}_{t,\theta}^M = S_0 \cdot \exp\left(\sum_{s=1}^t \left(\sigma_{s,\theta} \cdot \epsilon_{s,\theta} - \frac{\sigma_{s,\theta}^2}{2}\right)\right)$$

$$S_{t,\theta}^M = S_0 \cdot \exp \left( \sum_{s=1}^t \left[ \sigma_{s,\theta} \cdot \epsilon_{s,\theta} - \frac{\sigma_{s,\theta}^2}{2} \right] + rt \right) .$$

*Remark 7.7* (Comparison to Black-Scholes model). In the one-dimensional Black-Scholes model, the risk-neutral distribution of the price process is given by

$$S_t^{Q,BS} = S_0 \cdot \exp \left( \left( r - \frac{1}{2} \sigma^2 \right) t + \sigma W_t^Q \right) = S_0 \cdot \exp \left( \sigma W_t^Q - \frac{1}{2} \sigma^2 t + rt \right) .$$

The similarities to the price process given by the risk-neutral log return NP are clearly visible. Most importantly, in contrast to Black-Scholes, the model presented here does not assume a constant volatility and instead models it using the volatility generator.

In the same way, the volatility NP can be constrained to represent a known stochastic process such as the CIR process or the variance process of the GARCH( $p, q$ ) model. Both settings allow us to generate insights of the latent dynamics of the stochastic process at hand and thereby enable to validate modeling assumptions.

## 8 Pre- and Postprocessing

Prior to passing a realization of a financial time series  $s_{0:T} \in \mathcal{X}^{T+1}$  to the discriminator, the series has to be preprocessed. The applied pipeline is displayed in Figure 10. We will briefly explain each of the steps taken. Note that all of the used transformations, excluding the rolling window, are invertible and thus, allow a series sampled from a log return NP to be post-processed by inverting the steps 1-4 to obtain the desired form.

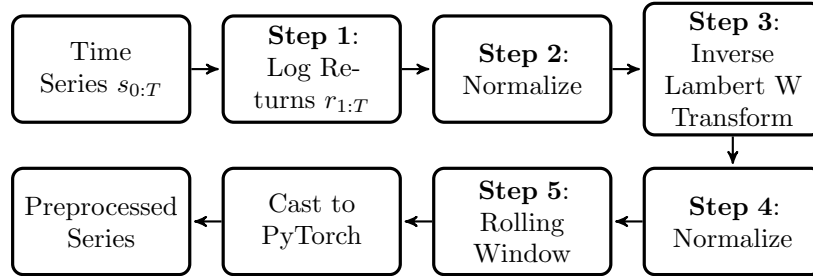


Figure 10: Condensed representation of the preprocessing pipeline.

### Step 1: Log returns $r_{1:T}$

Calculate the log return series

$$r_t = \log \left( \frac{s_t}{s_{t-1}} \right) \quad \text{for all } t \in \{1, \dots, T\}$$

### Step 2 & 4: Normalize

For numerical reasons, we normalize the data in order to obtain a series with zero mean and unit variance, which is thoroughly derived in [28].

### Step 3: Inverse Lambert W transform

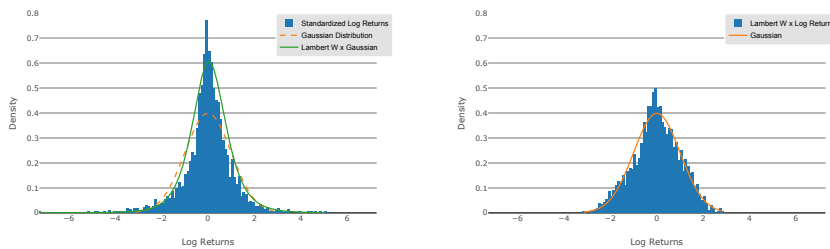
Since financial time series are generally considered to exhibit heavy-tails and an unbounded  $p$ -th moment for some  $p \in (2, 5]$  (cf. [10]), we utilize the *Lambert W probability transform*, as mentioned in [14], for being able to generate heavier tails. The Lambert W probability transform of an  $\mathbb{R}$ -valued random variable is defined as follows.

**Definition 8.1** (Lambert  $W \times F_X$ ). Let  $\delta \in \mathbb{R}$  and  $X$  be an  $\mathbb{R}$ -valued random variable with mean  $\mu$ , standard deviation  $\sigma$  and cumulative distribution function  $F_X$ . The location-scale Lambert  $W \times F_X$  transformed random variable  $Y$  is defined by

$$Y = U \exp\left(\frac{\delta}{2}U^2\right) \sigma + \mu, \quad (3)$$

where  $U := \frac{X - \mu}{\sigma}$  is the normalizing transform.

For  $\delta \in [0, \infty)$  the transformation used in (3) is of special interest as it is guaranteed to be bijective and differentiable. Hence, the transformations specific parameters  $\gamma = (\mu, \sigma, \delta)$  can be estimated via maximum likelihood. Moreover, for  $\delta > 0$  the Lambert  $W \times F_X$  transformed random variable has heavier tails than  $X$ .



(a) Standardized Log Returns

(b) Lambert W x Log Returns

Figure 11: (A) The original S&P 500 log returns and the fitted probability density function of a Lambert  $W \times$  Gaussian random variable. (B) The inverse Lambert  $W$  transformed log returns and the probability density function of a Gaussian random variable.

In particular, when applying the Lambert  $W$  transform to a normally distributed random variable, a Lambert  $W \times$  Gaussian random variable with positive excess kurtosis is obtained. By assuming that the log returns  $r_{1:T}$  are

approximately Lambert  $W \times$  Gaussian distributed, the quasi maximum likelihood principle can be employed to estimate the parameters  $(\mu, \sigma)$  of the normal distribution and the transformation specific parameter  $\delta$  (cf. [14, Section 4.1]). Hence, using the estimated parameters the inverse Lambert  $W$  transform can be applied to the log returns to yield a sequence of approximately normal distributed random variables with mean  $\mu$  and standard deviation  $\sigma$ .

The suggested transformation applied to the log returns of the S&P 500 is displayed in Figure 11. It shows the standardized original distribution of the S&P 500 log returns and the Lambert  $W$  inverse transformed log return distribution. Observe that the transformed standardized log return distribution in Figure 11b approximately follows the standard normal distribution and thereby circumvents the issue of not being able to generate the heavy tail of the original distribution.

### Step 5: Rolling window

When considering a discriminator with receptive field size  $T^{(d)}$ , we apply a rolling window of corresponding length and stride 1 to the preprocessed log return sequence  $r_t^{(\rho)}$ . Hence, for  $t \in \{1, \dots, T - T^{(d)}\}$  we define the sub-sequences

$$r_{1:T^{(d)}}^{(t)} := r_{t:(T^{(d)}+t-1)}^{(\rho)} \in \mathcal{X}^{T^{(d)}} . \quad (4)$$

Note that sliding a rolling window introduces a bias, since log returns at the beginning and end of the time series are under-sampled when training the Quant GAN. This bias can be corrected by using a (non-uniform) weighted sampling scheme when sampling batches from the training set.

## 9 Numerical Results

In this section we test the generative capabilities of Quant QANs by modeling the log returns of the S&P 500 index. For comparison we will apply the well-known GARCH(1,1) model to the same data. Our numerical results highlight that Quant GANs can learn a neural process that matches the empirical distribution and dependence properties far better than the presented GARCH model.

### 9.1 Setting and Implementation

The implementation was written with the programming language python. Neural network architectures were implemented by employing the python package pytorch [33], an automatic differentiation library primarily used for neural network computations and optimization. The training time of neural networks was decreased by using the CUDA-backend of pytorch in combination with a CUDA-enabled graphics processing unit (GPU). We used the GPU RTX 2070 from NVIDIA in order to train larger models. During preprocessing and evaluation we used the packages numpy and scipy [23].

Quant GANs were trained by using the *GAN stability algorithm* proposed by MESCHEDER, GEIGER, and NOWOZIN [29]. Furthermore, we used the TCN



setup proposed in [1] for both the generator and the discriminator and made use of skip connections as in [31] (see also Definition 5.16).

## 9.2 Models

For modeling the log returns of the S&P 500, we will have a look at three different model architectures: a pure TCN model, a constrained log return Neural Process and for comparison a simple GARCH model.

### Pure TCN

To check the capabilities of a pure TCN model, we model the log returns directly using a TCN with receptive field size  $T^{(g)}$  as generator, i.e. the return process is given by

$$R_{t,\theta} = g_\theta(Z_{t-(T^{(g)}-1):t})$$

for the noise prior  $Z_t \stackrel{iid}{\sim} \mathcal{N}(0, 1)$ .

### Constrained log return Neural Process

Assume a constrained log return NP (see Definition 7.2 and Section 7.4)

$$R_{t,\theta} = \sigma_{t,\theta} \cdot \epsilon_{t,\theta} + \mu_{t,\theta}$$

with volatility NP  $\sigma_{t,\theta}$ , drift NP  $\mu_{t,\theta}$  and an innovations NP  $\epsilon_{t,\theta}$  constrained to being i.i.d.  $\mathcal{N}(0, 1)$ -distributed.

### GARCH(1,1) with constant drift

Assume a GARCH(1,1) model with constant drift, where

$$\begin{aligned} R_{t,\theta} &= \xi_t + \mu \\ \xi_t &= \sigma_t \epsilon_t \\ \sigma_t^2 &= \omega + \alpha \xi_{t-1}^2 + \beta \sigma_{t-1}^2 \\ \epsilon_t &\stackrel{iid}{\sim} \mathcal{N}(0, 1) \end{aligned}$$

for  $\mu \in \mathbb{R}$ ,  $\omega > 0$ ,  $\alpha, \beta \in [0, 1]$  such that  $\alpha + \beta < 1$  and the parameter vector  $\theta = (\omega, \alpha, \beta, \mu)$ . For more details on GARCH-processes see [4].

## 9.3 Metrics and Scores

To compare the three different models with the S&P 500, we propose the use of the following metrics and scores.

## Distributional Metrics

**Earth Mover Distance** Let  $\mathbb{P}^h$  denote the historical and  $\mathbb{P}^g$  the generated distribution of the (possibly lagged) log returns. The *Earth Mover Distance* (or *Wasserstein-1 distance*) is defined as

$$\text{EMD}(\mathbb{P}^h, \mathbb{P}^g) = \inf_{\pi \in \Pi(\mathbb{P}^h, \mathbb{P}^g)} \mathbb{E}_{(X,Y) \sim \pi} [\|X - Y\|]$$

where  $\Pi(\mathbb{P}^h, \mathbb{P}^g)$  denotes the set of all joint probability distributions with marginals  $\mathbb{P}^h$  and  $\mathbb{P}^g$ . Loosely speaking the earth mover distance describes how much probability *mass* has to be moved to transform  $\mathbb{P}^h$  into  $\mathbb{P}^g$ . For more details, see [39].

**DY Metric** Additionally we compute the *DY metric* proposed by DRĂGULESCU and YAKOVENKO in [11]. The DY metric is for  $t \in \mathbb{N}$  defined by

$$\text{DY}(t) = \sum_x |\log P_t^h(A_{t,x}) - \log P_t^g(A_{t,x})| ,$$

where  $P_t^h$  and  $P_t^g$  denote the empirical density function of the historical and generated  $t$ -differenced log path. Further,  $(A_{t,x})_x$  denotes a partitioning of the real number line such that for fixed  $t$  and all  $x$  we (approximately) have

$$\log P_t^*(A_{t,x}) = 5/T$$

for  $T$  the number of historical log returns. During evaluation we consider the time lags  $t \in \{1, 5, 20, 100\}$  which represent a comparison of the daily, weekly, monthly and 100-day log returns.

## Dependence Scores

**ACF Score** The next score is proposed to compare the dependence properties of the historical and generated time series. Let  $r_{1:T}$  denote the historical log return series and  $\{r_{1:T,\theta}^{(1)}, \dots, r_{1:T,\theta}^{(M)}\}$  a set of generated series. The autocorrelation is defined as a function of time lag  $\tau$  and the series  $r_{1:T}$  and measures the correlation of the lagged time series and the series itself

$$\mathcal{C}(\tau; r) = \text{Corr}(r_{t+\tau}, r_t).$$

Denoting by  $C : \mathbb{R}^T \rightarrow [-1, 1]^S : r_{1:T} \mapsto (\mathcal{C}(1; r), \dots, \mathcal{C}(S; r))$  the autocorrelation function up to lag  $S$ , the ACF( $f$ ) score is computed for a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  as

$$\text{ACF}(f) := \left\| C(f(r_{1:T})) - \frac{1}{M} \sum_{i=1}^M C\left(f\left(r_{1:T,\theta}^{(i)}\right)\right) \right\|_2$$

where the function  $f$  is applied element-wise to the series. Motivated by the non-linear dependence structure of financial time series, we compute the ACF score for the functions  $f(x) = x$ ,  $f(x) = x^2$  and  $f(x) = |x|$  and constants  $S = 250$ ,  $M = 500$ .

**Leverage Effect Score** Similar to the ACF score the leverage effect score provides a comparison of historical and generated time dependence. The leverage effect will be measured using the correlation of the lagged, squared log returns and the log returns themselves, i.e. we consider

$$\mathcal{L}(\tau; r) = \text{Corr}(r_{t+\tau}^2, r_t)$$

for lag  $\tau$ . Denoting by  $L : \mathbb{R}^T \rightarrow [-1, 1]^S : r_{1:T} \mapsto (\mathcal{L}(1; r), \dots, \mathcal{L}(S; r))$  the leverage effect function up to lag  $S \leq T - 1$ , the leverage effect score is defined as

$$\left\| L(r_{1:T}) - \frac{1}{M} \sum_{i=1}^M L(r_{1:T,\theta}^{(i)}) \right\|_2$$

In our numerical results we compute the leverage effect score for  $S = 250$  and  $M = 500$ , i.e. the same as for the ACF score.

## 9.4 Generating the S&P 500 Index

We consider daily spot-prices of the S&P-500 from Mai 2009 until December 2018. For each of the three described models, we present in the appendix

- histograms of the real and generated log returns on a daily, weekly, monthly and 100-day basis,
- mean-autocorrelation functions of the serial, squared and absolute log returns,
- the correlations between the squared, lagged and the non-squared log returns as a proxy for the leverage effect,
- 5 plus additionally 50 exemplary generated log paths.

Further, Table 2 shows the values of the evaluated metrics for each of the models.

time series	time span	# of observations
S&P 500	Mai 2009 - December 2018	2413

Table 1: Considered financial time series.

### 9.4.1 Pure TCN

The displayed graphics show that the TCN model is capable of precisely modeling distributional and dependence properties present in the real S&P.

As can be seen in Figure A.3, the generated log returns closely match the histogram of the real returns on each of the presented time scales. Even for 100-day lagged returns, the fit is quite good.

The same holds true for the ACFs and the leverage effect plot, which deal with the dependence structure inherent in the data (see Figure A.4). The TCN accurately models the sharp drop in the ACF of the serial returns as well as the slowly decaying ACF of the squared and absolute log returns. Moreover, the

leverage effect is captured by a negative correlation between the squared and non-squared log returns for small time lags.

Recall that the displayed ACFs corresponding to the generated returns are mean ACFs and thereby much smoother than the ACF of the real returns.

Furthermore, the exemplary log paths shown in Figure A.1 and Figure A.2 exhibit reasonable patterns and demonstrate the structural diversity possible in the TCN model.

For all except two metrics evaluated in Table 2, the TCN performs best. In particular, the TCN clearly outperforms the GARCH(1,1) model in each metric, often by a factor 2-10.

#### 9.4.2 Constrained SVNN

The results of the C-SVNN are very similar to the TCN. This is expectable as the C-SVNN is constructed using a TCN and uses only a slight modification to decompose the log return process into its volatility, drift and innovation.

As is displayed in the graphics, the C-SVNN is able to capture the same properties as the TCN (see Figure A.7 and Figure A.8). Merely the ACF of the squared and absolute log returns is better modeled by the TCN.

The evaluated metrics of the C-SVNN in Table 2 are comparable to the results of the TCN, but for most of the metrics slightly worse. Furthermore, the C-SVNN also outperforms GARCH(1,1) model significantly.

Recall again that compared to the pure TCN, the SVNN has the structural advantages that the volatility can be directly modeled and a transition to its martingale distribution is known.

#### 9.4.3 GARCH(1,1) with constant drift

The GARCH model is clearly outperformed by the previously considered GAN-approaches two presented models in modeling distributional as well as dependence properties.

As indicated by the stylized facts of asset returns (see Section 4), the assumed normal distribution of the GARCH model places too less probability mass at the peak and the tails of the log return distribution as displayed by the histograms in Figure A.11.

In contrast, the autocorrelation function is captured quite well. This should not come as a surprise, as the GARCH structure was designed to capture this dependence. The main characteristics of the ACF plots in Figure A.12 are modeled, but the GARCH approach fails in exactness compared to the TCN and C-SVNN model. Note further that the leverage effect is not captured at all.

Table 2 supports this graphical assessment as the GARCH model performs worst in each of the evaluated metrics. For the ACF scores, the GARCH model is quite comparable to the other models as was already pointed out looking at the graphics. In contrast, in terms of the EMD and DY metric which focus on the return distribution, the GARCH model is clearly outperformed by the other two models.

	TCN	C-SVNN with drift	GARCH(1,1)
EMD(1)	<b>0.0039</b>	0.0040	0.0199
EMD(5)	<b>0.0039</b>	0.0040	0.0145
EMD(20)	<b>0.0040</b>	0.0069	0.0276
EMD(100)	<b>0.0154</b>	0.0464	0.0935
DY(1)	<b>19.1199</b>	19.8523	32.7090
DY(5)	<b>21.1167</b>	21.2445	27.4760
DY(20)	26.3294	<b>25.0464</b>	39.3796
DY(100)	28.1315	<b>25.8081</b>	46.4779
ACF(id)	<b>0.0212</b>	0.0220	0.0223
ACF( · )	<b>0.0248</b>	0.0287	0.0291
ACF((·) <sup>2</sup> )	<b>0.0214</b>	0.0245	0.0253
Leverage Effect	<b>0.3291</b>	0.3351	0.4636

Table 2: Evaluated metrics for the three models applied. For each row, the best value is printed bold.

## 10 Conclusion and Future Work

In this paper we showed that recently developed neural network architectures can be used in an adversarial modeling framework to approximate stochastic processes. Although these methods have been notoriously hard to train, advances in GANs showed that they can deliver competitive results and - as GAN training algorithms progress - promise even better performance in future.

For Quant GANs to flourish in the future there are two fundamental challenges that need to be addressed. The first is an exact modeling and extrapolation of the generated tail by incorporating prior knowledge such as the estimated tail-index. Second, a single metric needs to be developed which unifies distributional metrics with dependence scores we used in this paper and allows to benchmark different generator architectures. Once these points are sufficiently studied Quant GANs offer a data-driven method that surpasses the performance of other conventional models from mathematical finance.

## References

- [1] S. BAI, J. Z. KOLTER, and V. KOLTUN, *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*, CoRR abs/1803.01271 (2018), arXiv: 1803.01271.
- [2] S. BECKER, P. CHERIDITO, and A. JENTZEN, *Deep optimal stopping*, 2018, eprint: arXiv: 1804.05394.
- [3] F. BLACK and M. SCHOLES, *The Pricing of Options and Corporate Liabilities*, Journal of Political Economy 81.3 (1973), pp. 637–54.
- [4] T. BOLLERSLEV, *Generalized autoregressive conditional heteroskedasticity*, Journal of Econometrics 31.3 (1986), pp. 307–327.
- [5] A. BROCK, J. DONAHUE, and K. SIMONYAN, *Large Scale GAN Training for High Fidelity Natural Image Synthesis*, CoRR abs/1809.11096 (2018), arXiv: 1809.11096.
- [6] H. BÜHLER et al., *Deep hedging*, Quantitative Finance (Feb. 2019), pp. 1–21.
- [7] A. CHAKRABORTI et al., *Econophysics review: I. Empirical facts*, Quantitative Finance 11.7 (2011), pp. 991–1012, eprint: <https://doi.org/10.1080/14697688.2010.539248>.
- [8] J. CHUNG et al., *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, CoRR abs/1412.3555 (2014), arXiv: 1412.3555.
- [9] D. CLEVERT, T. UNTERTHINER, and S. HOCHREITER, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, in: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.
- [10] R. CONT, *Empirical properties of asset returns: stylized facts and statistical issues*, Quantitative Finance 1.2 (2001), pp. 223–236.
- [11] A. A. DRĂGULESCU and V. M. YAKOVENKO, *Probability distribution of returns in the Heston model with stochastic volatility*, Quantitative Finance 2.6 (2002), pp. 443–453, eprint: <https://doi.org/10.1080/14697688.2002.0000011>.
- [12] C. ESTEBAN, S. L. HYLAND, and G. RÄTSCH, *Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs*, ArXiv abs/1706.02633 (2018).
- [13] L. A. GATYS, A. S. ECKER, and M. BETHGE, *A Neural Algorithm of Artistic Style*, CoRR abs/1508.06576 (2015), arXiv: 1508.06576.
- [14] G. M. GOERG, *The Lambert Way to Gaussianize heavy-tailed data with the inverse of Tukey’s h transformation as a special case*, The Scientific World Journal 2015 (Oct. 2010).
- [15] I. GOODFELLOW, Y. BENGIO, and A. COURVILLE, *Deep Learning*, <http://www.deeplearningbook.org>, MIT Press, 2016.
- [16] I. GOODFELLOW et al., *Generative Adversarial Nets*, in: *Advances in Neural Information Processing Systems 27*, ed. by Z. GHAHRAMANI et al., Curran Associates, Inc., 2014, pp. 2672–2680.
- [17] I. GOODFELLOW et al., *Maxout Networks*, in: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML’13*, JMLR.org, Atlanta, GA, USA, 2013, pp. III-1319–III-1327.
- [18] A. GRAVES and J. SCHMIDHUBER, *Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks*, in: *Advances in Neural Information Processing Systems 21*, ed. by D. KOLLER et al., Curran Associates, Inc., 2009, pp. 545–552.
- [19] K. HE et al., *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, in: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV ’15, IEEE Computer Society, Washington, DC, USA, 2015, pp. 1026–1034, ISBN: 978-1-4673-8391-2.

- [20] S. L. HESTON, *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, Review of Financial Studies 6 (1993), pp. 327–343.
- [21] S. HOCHREITER and J. SCHMIDHUBER, *Long Short-term Memory*, Neural computation 9 (Dec. 1997), pp. 1735–80.
- [22] K. HORNIK, *Approximation capabilities of multilayer feedforward networks*, Neural Networks 4.2 (1991), pp. 251–257, ISSN: 0893-6080.
- [23] E. JONES, T. OLIPHANT, P. PETERSON, et al., *SciPy: Open source scientific tools for Python*, 2001–.
- [24] A. KARPATY, *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*, URL: <http://cs231n.stanford.edu/syllabus.html>.
- [25] G. KLAMBAUER et al., *Self-Normalizing Neural Networks*, CoRR abs/1706.02515 (2017), arXiv: 1706.02515.
- [26] A. S. KOSHIYAMA, N. FIROOZY, and P. C. TRELEAVEN, *Generative Adversarial Networks for Financial Trading Strategies Fine-Tuning and Combination*, CoRR abs/1901.01751 (2019), arXiv: 1901.01751.
- [27] A. KRIZHEVSKY, I. SUTSKEVER, and G. E. HINTON, *ImageNet classification with deep convolutional neural networks*, Commun. ACM 60.6 (2017), pp. 84–90.
- [28] Y. LECUN et al., *Efficient BackProp*, in: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, Springer-Verlag, London, UK, UK, 1998, pp. 9–50, ISBN: 3-540-65311-2.
- [29] L. MESCHEDER, A. GEIGER, and S. NOWOZIN, *Which Training Methods for GANs do actually Converge?*, in: *International Conference on Machine learning (ICML)*, 2018.
- [30] V. NAIR and G. E. HINTON, *Rectified Linear Units Improve Restricted Boltzmann Machines*, in: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, Omnipress, Haifa, Israel, 2010, pp. 807–814, ISBN: 978-1-60558-907-7.
- [31] A. van den OORD et al., *WaveNet: A Generative Model for Raw Audio*, in: *Arxiv*, 2016.
- [32] R. PASCANU, T. MIKOLOV, and Y. BENGIO, *On the difficulty of training recurrent neural networks*, in: *Proceedings of the 30th International Conference on Machine Learning*, ed. by S. DASGUPTA and D. MCALLESTER, vol. 28, Proceedings of Machine Learning Research 3, PMLR, Atlanta, Georgia, USA, 17–19 Jun 2013, pp. 1310–1318.
- [33] A. PASZKE et al., *Automatic differentiation in PyTorch*, in: *NIPS-W*, 2017.
- [34] M. SCHREYER et al., *Detection of Anomalies in Large Scale Accounting Data using Deep Autoencoder Networks*, CoRR abs/1709.05254 (2017), arXiv: 1709.05254.
- [35] D. SILVER et al., *Mastering the game of Go with deep neural networks and tree search*, Nature 529 (2016), pp. 484–503.
- [36] S. SUWAJANAKORN, S. M. SEITZ, and I. KEMELMACHER-SHLIZERMAN, *Synthesizing Obama: Learning Lip Sync from Audio*, ACM Trans. Graph. 36.4 (July 2017), 95:1–95:13, ISSN: 0730-0301.
- [37] S. TAKAHASHI, Y. CHEN, and K. TANAKA-ISHII, *Modeling financial time-series with generative adversarial networks*, Physica A: Statistical Mechanics and its Applications 527 (Aug. 2019), p. 121261.
- [38] P. TANKOV, *Financial modelling with jump processes*, vol. 2, CRC press, 2003.
- [39] C. VILLANI, *Optimal transport – Old and new*, in: vol. 338, Jan. 2008, pp. xxii+973.
- [40] M. WIESE, R. KNOBLOCH, and R. KORN, *Copula & Marginal Flows: Disentangling the Marginal from its Joint* (2019), arXiv: 1907.03361.
- [41] B. XU et al., *Empirical Evaluation of Rectified Activations in Convolutional Network*, CoRR abs/1505.00853 (2015), arXiv: 1505.00853.

- [42] K. ZHANG et al., *Stock Market Prediction Based on Generative Adversarial Network*, *Procedia Computer Science* 147 (2019), pp. 400–406.
- [43] X. ZHOU et al., *Stock Market Prediction on High-Frequency Data Using Generative Adversarial Nets*, *Mathematical Problems in Engineering* 2018 (2018), pp. 1–11.



## A Numerical Results

### A.1 Pure TCN

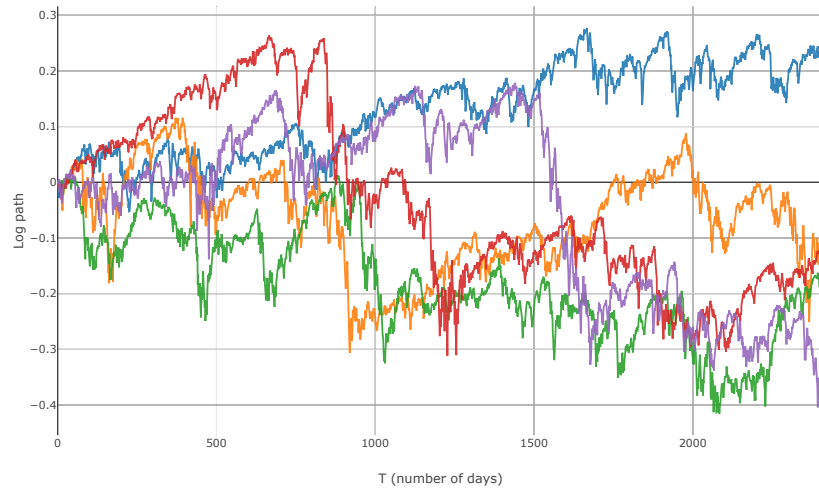


Figure A.1: 5 generated log paths

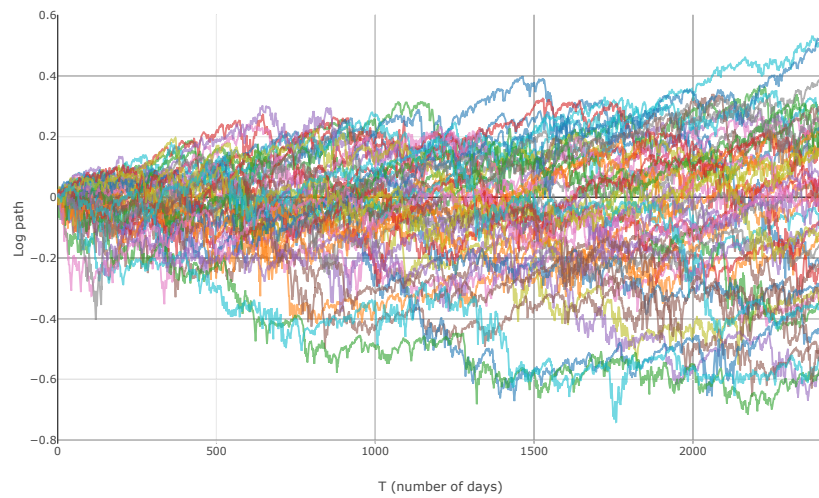


Figure A.2: 50 generated log paths

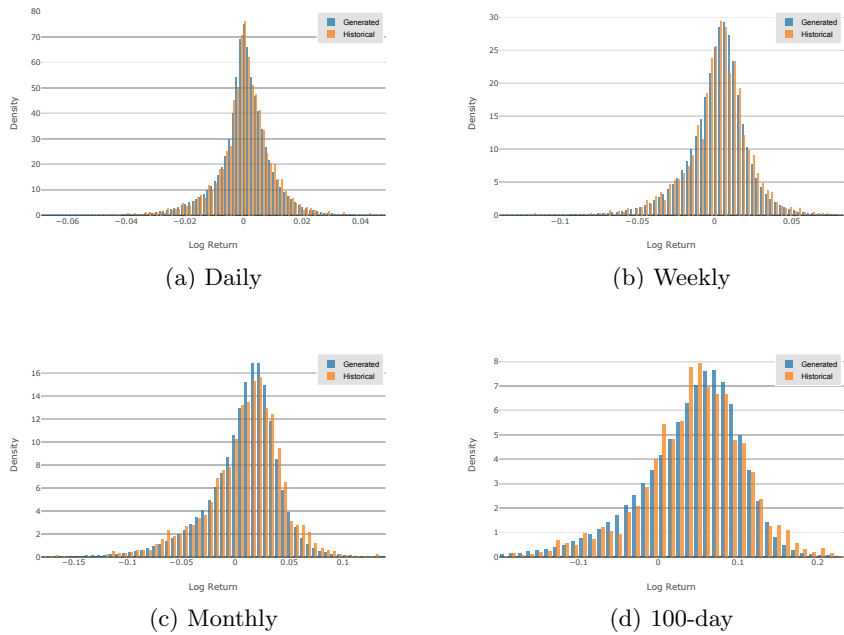


Figure A.3: Comparison of generated and historical densities of the S&P500.

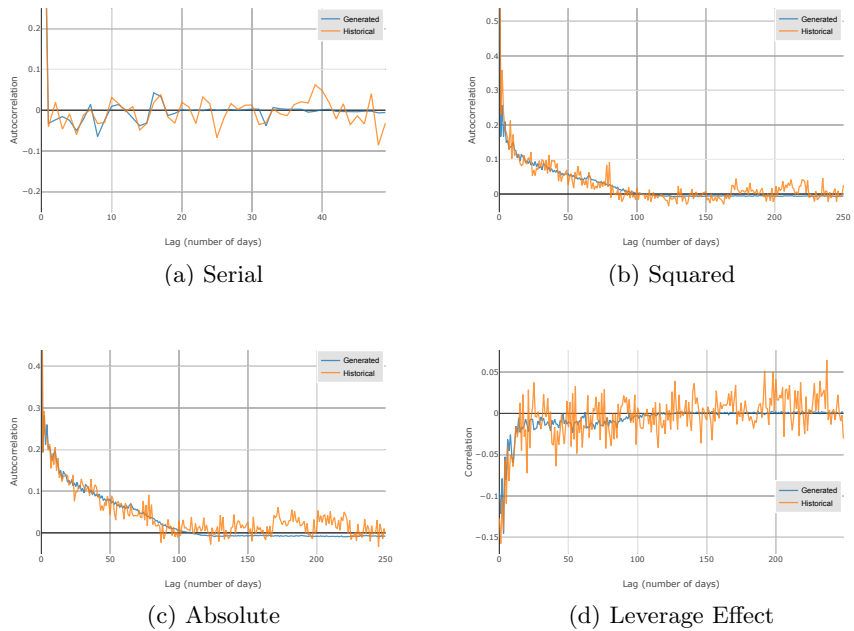


Figure A.4: Mean autocorrelation function of the absolute, squared and identical log returns and leverage effect.

## A.2 Constrained SVNN

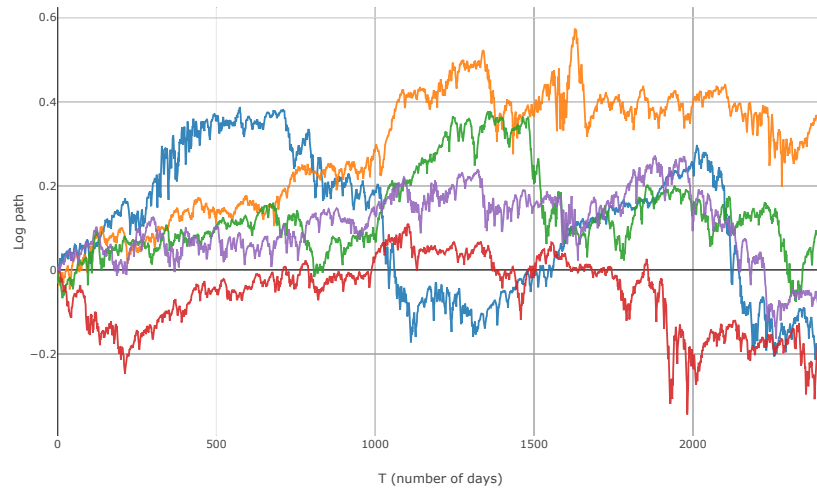


Figure A.5: 5 generated log paths

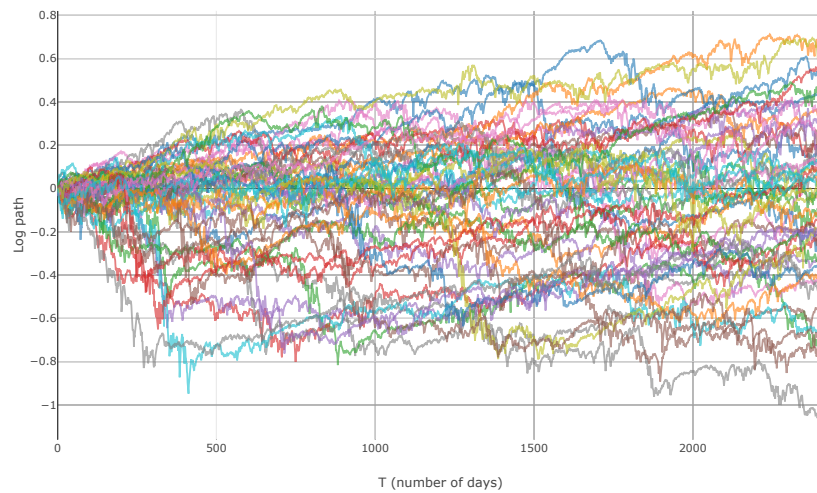


Figure A.6: 50 generated log paths

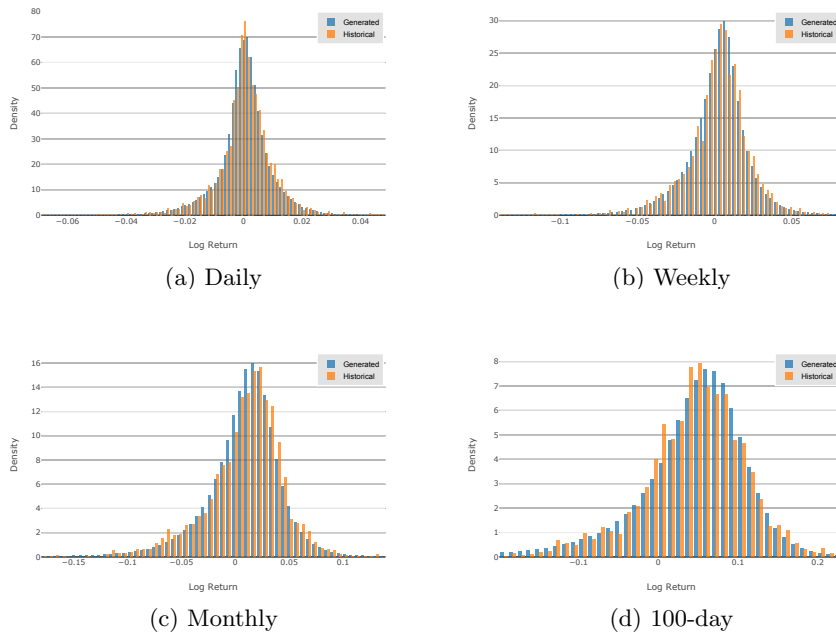


Figure A.7: Comparison of generated and historical densities of the S&P500.

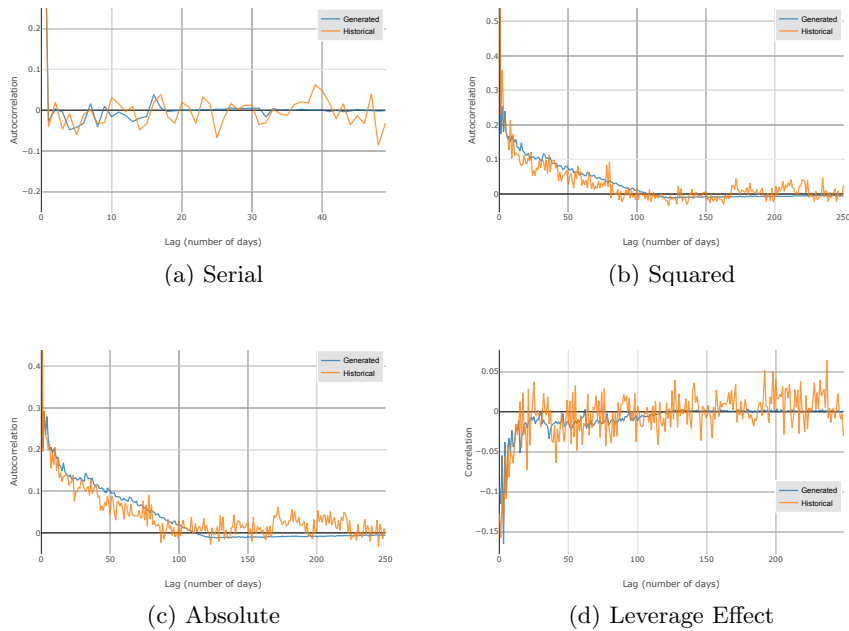


Figure A.8: Mean autocorrelation function of the absolute, squared and identical log returns and leverage effect.

### A.3 GARCH(1,1) with constant drift

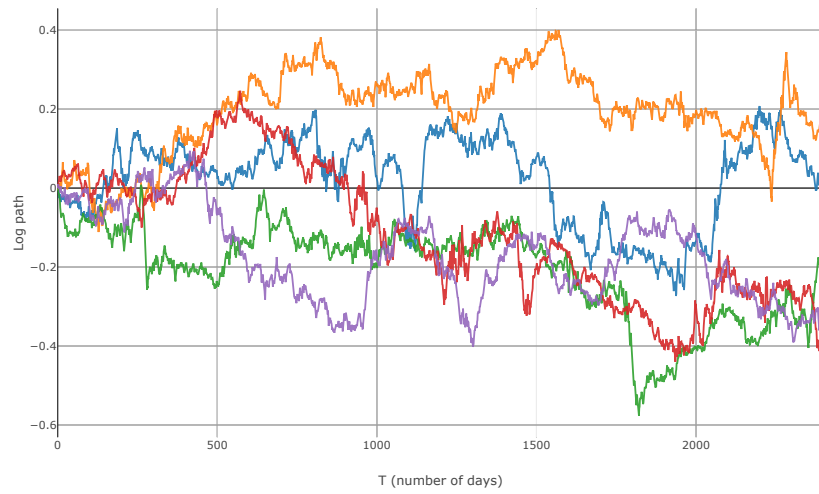


Figure A.9: 5 generated log paths

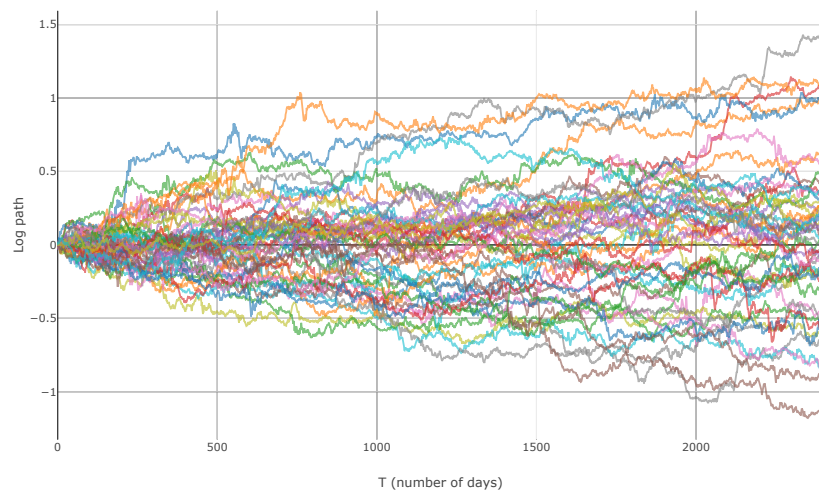


Figure A.10: 50 generated log paths



Figure A.11: Comparison of generated and historical densities of the S&P500.

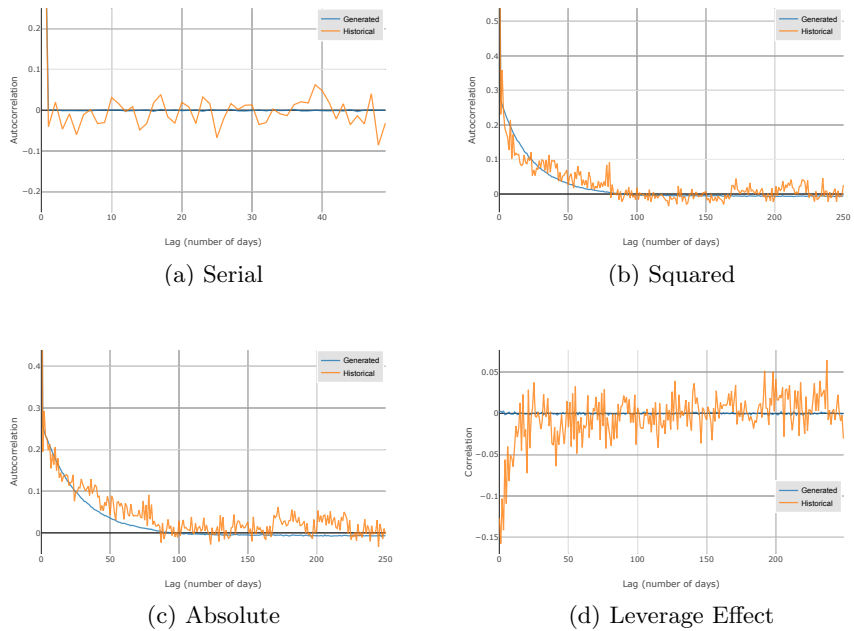


Figure A.12: Mean autocorrelation function of the absolute, squared and identical log returns and leverage effect.