Quant GANs: Deep Generation of Financial Time Series

Magnus Wiese[1, 2,*] Robert Knobloch[2] Ralf Korn[1, 2]

Peter Kretschmer[1, 2]

[1]TU Kaiserslautern, Gottlieb-Daimler-Straße 48, 67663 Kaiserslautern, Germany
[2]Fraunhofer ITWM, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany

Abstract

Modeling financial time series by stochastic processes is a challenging task and a central area of research in financial mathematics. As an alternative, we introduce 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 long-range dependencies such as the presence of volatility clusters. 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 (Krizhevsky et al., 2012) at the ImageNet competition, neural networks (NNs) excel in various areas ranging from generating realistic audio waves (van den Oord et al., 2016) to surpassing human-level performance on ImageNet (He et al., 2015) or beating the world champion in the game Go (Silver et al., 2016). While NNs have already become standard tools in image analysis, the application in finance is still in early stages. Citing a few, Buehler et al. (2019a,b) use NNs to hedge large portfolios of derivatives, Becker et al. (2018) solve optimal stopping problems and Schreyer et al. (2017, 2019a,b) detect anomalies in accounting data and show how auditing firms can be attacked by DeepFakes.

In this article, we consider the problem of approximating a realistic asset price simulator by using NNs and adversarial training techniques. Such a path simulator is useful as it can be used to extend and enrich limited real-world datasets, which in turn can be used to fine-tune or robustify financial trading strategies.

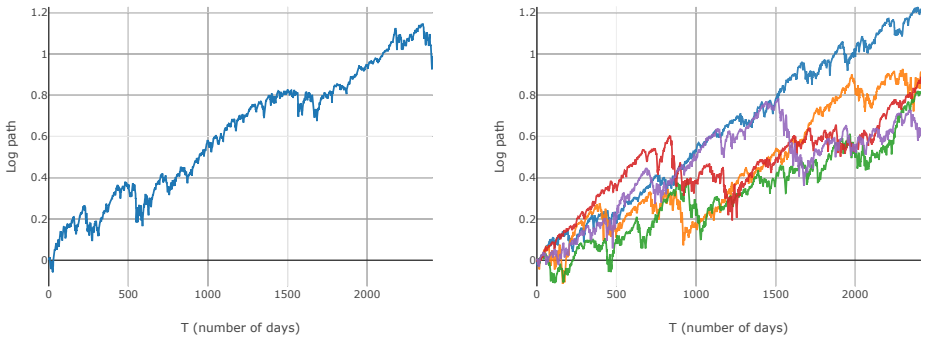
To approximate a data-driven path simulator we propose *Quant GANs*. Quant GANs are based on the application of generative adversarial networks (GANs) (Goodfellow et al., 2014) and are located between pure data-based approaches such as historical simulation and model-driven methods such as Monte Carlo simulation assuming an underlying stock price model like the Black Scholes model (Black and Scholes, 1973), the Heston stochastic volatility model (Heston, 1993) or Lévy process based modeling (Tankov, 2003).

Using two different NNs as opponents is the fundamental principle of GANs. While one NN, the so-called generator, is responsible for the generation of stock price paths, the second one, the

*Corresponding author: quant.gans@gmail.com

discriminator, has to judge whether the generated paths are synthetic or from the same underlying distribution as the data (i.e. the past prices).

Various pitfalls exist when training GANs ranging from limited convergence when optimizing both networks simultaneously (cf. Arjovsky and Bottou (2017); Mescheder et al. (2018)) to extrapolation problems when using recurrent generation schemes. To address the latter issue we propose the use of *temporal convolutional networks* (*TCNs*), also known as *WaveNets* (van den Oord et al., 2016), as the generator architecture. A TCN generator comes with multiple benefits: it is particularly suited for modeling long-range dependencies, allows for parallization and guarantees stationarity.



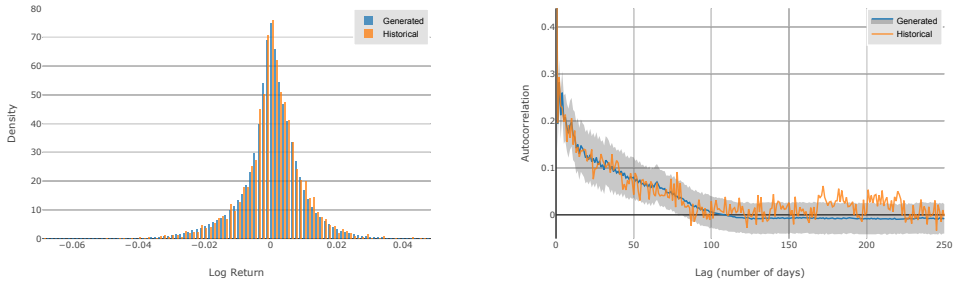
(a) S&P 500 log path

(b) Generated log paths of a Quant GAN

Figure 1: Comparison of the S&P 500 index in log-space (a) with generated log paths of a calibrated Quant GAN (b).

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 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.

After introducing NNs (and in particular TCNs) in Section 3 and GANs in Section 4, we define the proposed generator architecture of Quant GANs, namely *Stochastic Volatility Neural Networks (SVNNs)* in Section 5. In spirit of stochastic volatility models, the SVNN architecture consists of a volatility and drift TCN and an innovation NN. 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.



(a) Histogram of daily log returns

(b) ACF of absolute log returns

Figure 2: Figures (a) and (b) display a comparison of the generated Quant GAN (blue) and empirical S&P 500 (orange) samples.

For SVNNs we prove theoretical L^p-space-related claims, which demonstrate that all moments of the process exist. Since financial time series are understood to exhibit heavy-tails, we show that the Lambert W transformation plays an essential role for generating heavy-tailed stock price returns by using methods for normalized Gaussian data.

In Section 7, we present a numerical study applying Quant GANs to the S&P 500 index from May 2009 - December 2018 (see Figure 1). Our results demonstrate that SVNNs and TCNs outperform a typical GARCH-model with respect to distributional and dependence properties. Figure 2 illustrates a comparison of some sample properties of synthetic paths with the S&P 500 index.

2 Generative modeling of financial time series

In this section, we briefly collect some facts and modeling issues of financial time series, which motivate our choice of NN types. Furthermore, we review existing literature related to (data-driven) modeling of financial time series.

First, note that the performance of a stock over a certain period (as e.g. a day, a month or a year) is given by its relative return, either $R_t = (S_t - S_{t-1})/S_{t-1}$ or its log return $R_t = \log(S_t) - \log(S_{t-1})$. Therefore, the generation of asset returns is the main objective of this paper. The characteristic properties of asset returns are well-studied and commonly known as *stylized facts*. A list of the most important stylized facts includes (see e.g. Chakraborti et al. (2011); Cont (2001)):

- 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.

A large literature of financial time series models exist ranging from a variety of discrete-time GARCH-inspired models (Bollerslev, 1986) to models in continuous-time such as Black and Scholes (1973), Heston (1993) and rough extensions (El Euch et al., 2018). However, the development and innovation of new models is difficult: it took 20 years to extend the Black-Scholes model, which assumes that asset prices can be described through geometric Brownian motions, to the more sophisticated Heston model, which accounted for stochastic volatility and the leverage effect.

Pure data-driven modeling with NNs is a relatively new sub-field of research, which bears the potential of being able to model complicated statistical (perhaps unknown) dynamics. As a consequence, literature is rather sparse and can be summarized by four main manuscripts. Koshiyama et al. (2019) approximate a conditional model with GANs and define a score to select simulator / parameter candidates. Pardo (2019) uses Wasserstein and relativistic GANs (cf. Gulrajani et al. (2017); Jolicoeur-Martineau (2018)) and presents synthetic paths with volatility clusters and similar statistics. They also demonstrate that the use of synthetic paths can increase the accuracy of a trading strategy (cf. Pardo and López (2019)). Takahashi et al. (2019) demonstrates that GANs can approximate various stylized facts. Wiese et al. (2019a) show that recurrent architectures can be used to generate equity option markets that satisfy static arbitrage constraints by using adversarial training and discrete local volatilities (Buehler and Ryskin, 2017).

However, Pardo (2019) and Takahashi et al. (2019) both lack details on their NN architecture and do not state whether their proposed algorithm approximates the conditional or unconditional distribution. For all of the mentioned papers code is not available such that benchmarks are difficult to develop due to limited reproducibility.

3 Neural network topologies

In this section, we introduce the NN topologies that are essential to construct SVNNs. First, the *multilayer perceptron (MLP)* is defined. Afterward, we provide a formal definition of TCNs.

3.1 Multilayer perceptrons

The MLP lies at the core of deep learning models. It is constructed by composing affine transformations with so-called *activation functions*; non-linearities that are applied element-wise. Figure 3 depicts the construction of an MLP with two hidden layers, where the input is three-dimensional and the output is one-dimensional. We begin with the activation function as the crucial ingredient and then define the MLP formally.

Definition 3.1 (Activation function)**.** A function $\phi : \mathbb{R} \to \mathbb{R}$ that is Lipschitz continuous and monotonic is called *activation function*.

Remark 3.2. Definition 3.1 comprises a large class of functions used in the deep learning literature. Examples of activation functions in the sense of Definition 3.1 include $\tanh$, rectifier linear units (*ReLUs*) (Nair and Hinton, 2010), parametric ReLUs (*PReLUs*) (He et al., 2015), MaxOut (Goodfellow et al., 2013) and a vast amount of other functions in the literature Clevert et al. (2016); Klambauer et al. (2017); LeCun et al. (1998); Xu et al. (2015). Note that $\phi(0) = 0$ is a desired property for the optimization of the networks parameters (cf. Karpathy (2019)) and is satisfied by the above activation functions.

Definition 3.3 (Multilayer perceptron)**.** Let $L, N_0, \dots, N_{L+1} \in \mathbb{N}$, ϕ an activation function, Θ an Euclidean vector space and for any $l \in \{1, \dots, L+1\}$ let $a_l : \mathbb{R}^{N_{l-1}} \to \mathbb{R}^{N_l}$ be an affine mapping. A function $f : \mathbb{R}^{N_0} \times \Theta \to \mathbb{R}^{N_{L+1}}$, defined by

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

where $\circ$ denotes the composition operator,

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

and ϕ 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

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

Remark 3.4. We call a function $f : \mathbb{R}^{d_0} \times \Theta \to \mathbb{R}^{d_1}$ with parameter space Θ a *network*, if it is Lipschitz continuous.

A well-known result that justifies the high applicability of MLPs is the so-called *universal approximation theorem* for one-layer perceptrons with output dimension $d_1 = 1$, see for instance Theorem 1 and Theorem 2 in Hornik (1991) or Theorem 4.2 in Buehler et al. (2019a). Let us point out that the universal approximation theorem easily carries over to the case of MLPs with output dimension $d_1 > 1$ and more than one hidden layer, which corresponds to the situation under consideration in the present paper.

3.2 Temporal convolutional networks

As a result of volatility clusters being present in the market, the log return process is often decomposed into a stochastic volatility process 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 Bai et al. (2018). In particular, empirical results suggest that TCNs are able to capture long-range dependencies in sequences more effectively than well-known recurrent architectures (Goodfellow et al., 2016, Chapter 10) such as the LSTM (Hochreiter and Schmidhuber, 1997) or the GRU (Chung et al., 2014). One of the main advantages of TCNs compared to recurrent NNs is the absence of exponentially vanishing and exploding gradients through time (Pascanu et al., 2013), 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 perform better on supervised learning benchmarks (Bai et al., 2018).

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.

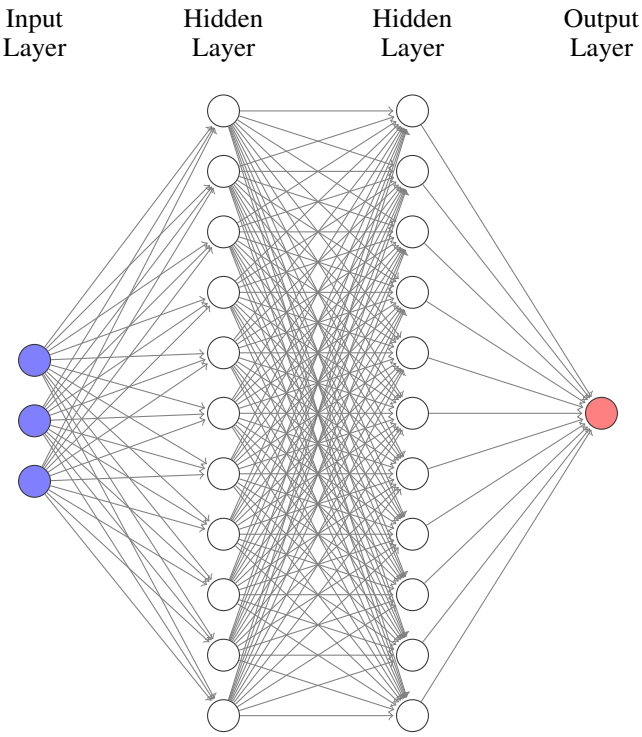


Figure 3: Two-layer MLP with a three-dimensional input space ($N_0 = 3$) and an one-dimensional output space ($N_3 = 1$). The hidden dimensions of layer one and two are eleven ($N_1 = N_2 = 11$).

Dilated convolutions, also referred to as *atrous* convolutions, are convolutions "with holes". Figure 5 illustrates a *Vanilla TCN* (cf. Definition 3.11) with four hidden layers, a kernel size of two ($K = 2$) and a dilation factor of one ($D = 1$). Figure 6 depicts a TCN with a kernel size and a dilation factor equal to two ($K = D = 2$). Note that as $D = 2$, the dilation increases by a factor of two in every layer. Comparing both networks it becomes clear that the use of increasing dilations in each layer is essential to capture and model long-range dependencies.

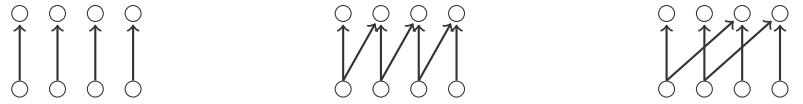
Below we define the TCN as well as related concepts formally. We first give the definition of the dilated causal convolutional operator, which is the basic building block of the *convolutional layer*. Composing several convolutional layers (together with activation functions) gives the Vanilla TCN. For the rest of this section, let $N_I, N_O, K, D, T \in \mathbb{N}$.

Definition 3.5 ($*_D$ operator)**.** Let $X \in \mathbb{R}^{N_I \times T}$ 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 $t \in \{D(K-1)+1, \ldots, T\}$ and $m \in \{1 \ldots, N_O\}$ the operator $*_D$, defined by

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

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 4. For $K = D = 1$ (see Figure 4a), each element of the output sequence only depends on the element of the input sequence at the same time step. In the case of $K = 2$ and $D = 1$, each element of the output sequence originates from the elements of the input sequence at the same and previous time step. In the case of $K = D = 2$, the distance between the elements that pass on the information is two.



(a) $K = D = 1$ (b) $K = 2$, $D = 1$ (c) $K = D = 2$

Figure 4: Dilated causal convolutional operator for different dilations D and kernel sizes K

Definition 3.6 (Causal convolutional layer)**.** Let W be as in Definition 3.5 and $b \in \mathbb{R}^{N_O}$. A function

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

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

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

is called *causal convolutional layer with dilation* D.

Remark 3.7. 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.

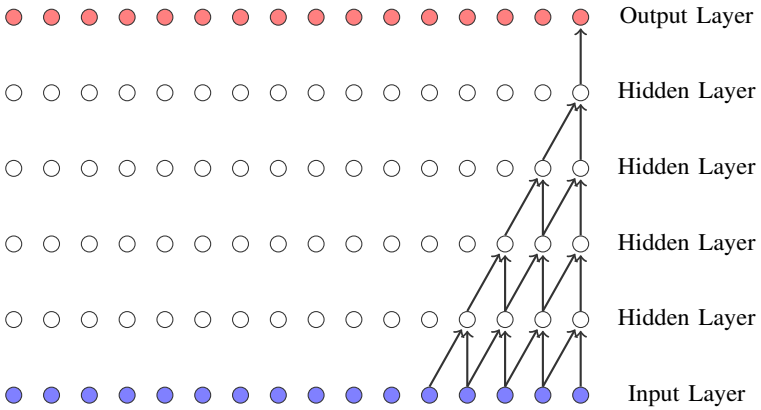


Figure 5: Vanilla TCN with 4 hidden layers, kernel size $K = 2$ and dilation factor $D = 1$ (cf. van den Oord et al. (2016)).

Example 3.8 (1×1 convolutional layer)**.** Let $X \in \mathbb{R}^{N_I \times T}$ be an N_I-variate sequence and $w : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times T}$ a causal convolutional layer with arguments $(N_I, N_O, 1, 1)$. We call such a layer a 1×1 *convolutional layer*.[2]

In the previous section we constructed MLPs by composing affine transformations with activation functions. The Vanilla TCN construction follows a similar approach: dilated causal convolutional layers are composed with activation functions. In order to allow for more expressive transformations, the TCN uses a *block module* construction and thereby generalizes the Vanilla TCN. For completeness, both definitions are given below.

Definition 3.9 (Block module)**.** Let $S \in \mathbb{N}$. A function $\psi : \mathbb{R}^{N_I \times T} \to \mathbb{R}^{N_O \times (T-S)}$ that is Lipschitz continuous is called *block module* with arguments (N_I, N_O, S).

Definition 3.10 (Temporal convolutional network)**.** Let $T_0, L, N_0, \ldots, N_{L+1} \in \mathbb{N}$. Moreover, for $l \in \{1, \ldots, L\}$ 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

$$T_L = T_0 - \sum_{l=1}^{L} S_l \geq 1 \,. \tag{1}$$

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

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

[2]Note that using a 1×1 convolution is equivalent to applying an affine transformation along the time dimension of 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 $\mathrm{TCN}_{d_0,d_1,L}$ ($d_0 = N_0, d_1 = N_{L+1}$).

Definition 3.11 (Vanilla TCN)**.** Let $f \in \mathrm{TCN}_{N_0,N_{L+1},L}$ such that for all $l \in \{1,\dots,L\}$ each block module ψ_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 ϕ, i.e. $\psi_l = \phi \circ w_l$. Then we call $f : \mathbb{R}^{N_0 \times T_0} \times \Theta \to \mathbb{R}^{N_{L+1} \times T_L}$ 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.

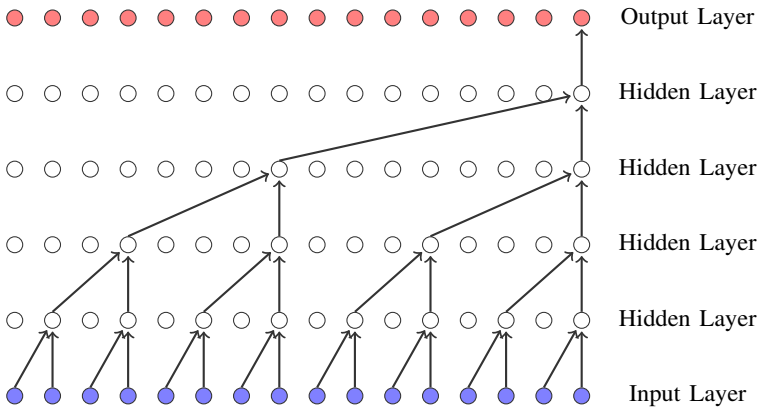


Figure 6: Vanilla TCN with 4 hidden layers, kernel size $K = 2$ and dilation factor $D = 2$ (cf. van den Oord et al. (2016)).

TCN's ability to model long-range dependencies becomes ultimately apparent when comparing the two Vanilla TCNs displayed in Figure 5 and Figure 6. In Figure 5, the network is a function of 5 sequence elements, whereas the network in Figure 6 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 3.12 (Receptive field size)**.** Let $f \in \mathrm{TCN}_{d_0,d_1,L}$ and let $S_1,\dots,S_L$ be as in Definition 3.10. The constant

$$T^{(f)} := 1 + \sum_{l=1}^{L} S_l$$

is called *receptive field size* (*RFS*).

Remark 3.13. For Vanilla TCNs with kernel size K and dilation factor $D > 1$, the RFS $T^{(f)}$ can easily be computed using the formula for the sum of a geometric sequence with finite length:

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

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

Remark 3.14. Note that an MLP can be seen as a Vanilla TCN in which each causal convolution is a 1×1 convolution. Thus, MLPs are a subclass of TCNs with an RFS equal to one.

The idea of residual connections can also be used.

Definition 3.15 (TCN with skip connections)**.** Assume the notation from Definition 3.10 and for $N_{skip} \in \mathbb{N}$ let

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

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

$$\left(X^{(l)}, H^{(l)} \right) = \gamma_l \left(X^{(l-1)} \right) \quad \text{for } l \in \{1,\dots,L\}$$

$$Y = \gamma \left(\sum_{l=1}^{L} H^{(l)} \right) ,$$

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

One of the downsides of TCNs is that the length of time series to be processed is restricted to the TCN's RFS. Hence, in order to model long-range dependencies we require a RFS $T^{(f)} \gg 1$ 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 to model long-range dependencies (cf. Dieleman et al. (2018)), we leave it as future work to develop these methods.

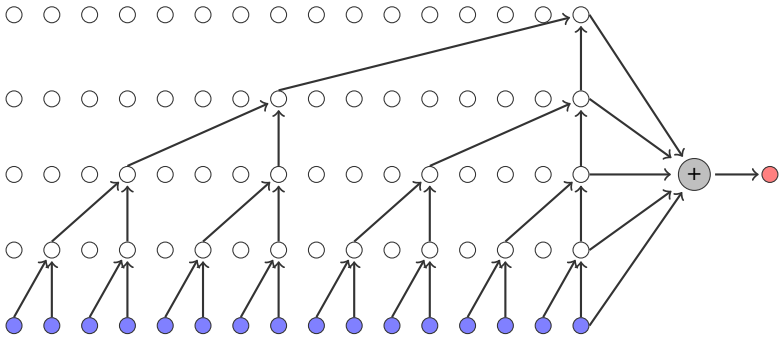


Figure 7: Vanilla TCN with skip connections.

The NN topologies introduced in this section are the core components used to achieve the results presented in Section 7. In order to train these networks to generate time series, we now formulate GANs in the setting of random variables and stochastic processes.

4 Generative adversarial networks

Generative adversarial networks (GANs) (Goodfellow et al., 2014) are a relatively new class of algorithms to learn a generative model of a sample of a random variable, i.e. a dataset, or the distribution of a random variable itself. Originally, GANs were applied to generate images. In this section, we introduce GANs for random variables and then extend them to discrete-time stochastic processes with TCNs.

Throughout this section let $N_Z, N_X \in \mathbb{N}$ and let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. Furthermore, assume that X and Z are $\mathbb{R}^{N_X}$ and $\mathbb{R}^{N_Z}$-valued random variables, respectively. The distribution of a random variable X will be denoted by $\mathbb{P}_X$.

4.1 Formulation for random variables

In the context of GANs, $(\mathbb{R}^{N_Z}, \mathcal{B}(\mathbb{R}^{N_Z}))$ and $(\mathbb{R}^{N_X}, \mathcal{B}(\mathbb{R}^{N_X}))$ are called the *latent* and the *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 $g : \mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ such that the induced random variable $g_\theta(Z) \coloneqq g_\theta \circ Z$ for some parameter $\theta \in \Theta^{(g)}$ and the targeted random variable X have the same distribution, i.e. $g_\theta(Z) \overset{d}{=} X$. To achieve this, Goodfellow et al. proposed the adversarial modeling framework for NNs and introduced the *generator* and the *discriminator* as follows:

Definition 4.1 (Generator)**.** Let $g : \mathbb{R}^{N_Z} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ be a network with parameter space $\Theta^{(g)}$. The random variable $\tilde{X}$, defined by

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

is called the *generated random variable*. Furthermore, the network g is called *generator* and $\tilde{X}_\theta$ the *generated random variable with parameter* θ.[3]

[3]The subscript θ of $\tilde{X}_\theta$ represents the dependency with respect to the neural operator's parameters θ.

Definition 4.2 (Discriminator)**.** Let $\tilde{d} : \mathbb{R}^{N_X} \times \Theta^{(d)} \to \mathbb{R}$ be a network with parameters $\eta \in \Theta^{(d)}$ and $\sigma : \mathbb{R} \to [0,1] : x \mapsto \frac{1}{1+e^{-x}}$ be the sigmoid function. A function $d : \mathbb{R}^{N_X} \times \Theta^{(d)} \to [0,1]$ defined by $d : (x, \eta) \mapsto \sigma \circ \tilde{d}_\eta(x)$ is called a *discriminator*.

Throughout this section we assume the notation used in Definition 4.1 and Definition 4.2.

Definition 4.3 (Sample)**.** A collection $\{Y_i\}_{i=1}^M$ of M independent copies of some random variable Y is called Y*-sample* of size M. The notation $\{y_i\}_{i=1}^M$ refers to a realisation $\{Y_i(\omega)\}_{i=1}^M$ for some $\omega \in \Omega$.

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 the generator distribution. In other words, the discriminator $d_\eta : \mathbb{R}^{N_X} \to [0,1]$ acts as a classifier that assigns to each sample $x \in \mathbb{R}^{N_X}$ a probability of being a realization of the target distribution.

The optimization of GANs is formulated in two steps. First, the discriminator's parameters $\eta \in \Theta^{(d)}$ are chosen to maximize the function $\mathcal{L}(\theta, \cdot)$, $\theta \in \Theta^{(g)}$, given by

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

In this sense, the discriminator learns to distinguish real and generated data. 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 refer to as the *GAN objective*.

Training The generator's and discriminator's parameters (θ, η) are trained by alternating the computation of their 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_{η^*} (Goodfellow et al., 2014, Proposition 1) it is common to compute the discriminators gradient multiple times and ascent the parameters η. Algorithm 1 describes the procedure in detail.

For further information on Algorithm 1, we refer to Section 4 in Goodfellow et al. (2014), where this algorithm was developed. In particular, regarding the convergence of Algorithm 1 we refer to Section 4.2 in Goodfellow et al. (2014).

4.2 Formulation for stochastic processes

We now consider the formulation of GANs in the context of stochastic process generation by using TCNs, as its properties are intriguing for this setting. The following notation is used for brevity.

Notation 4.4. 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 ω-realization

$$X_{s:t,\theta}(\omega) := (X_{s,\theta}(\omega), \dots, X_{t,\theta}(\omega)) \in \mathbb{R}^{N_X \times (t-s+1)}.$$

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

Definition 4.5 (Neural process)**.** Let $(Z_t)_{t \in \mathbb{Z}}$ be an i.i.d. noise process with values in $\mathbb{R}^{N_Z}$ and $g : \mathbb{R}^{N_Z \times T^{(g)}} \times \Theta^{(g)} \to \mathbb{R}^{N_X}$ a TCN with RFS $T^{(g)}$ and parameters $\theta \in \Theta^{(g)}$. A stochastic process $\tilde{X}$, defined by

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

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

Algorithm 1 GAN optimization.

INPUT: generator g, discriminator d, sample size $M \in \mathbb{N}$, generator learning rate α_g, discriminator learning rate α_d, number of discriminator optimization steps k
OUTPUT: parameters (θ, η)

while not converged **do**
  for k steps **do**
    Let $\{\tilde{x}_{\theta,i}\}_{i=1}^{M}$ be a realisation of an $\tilde{X}_\theta$-sample of size M.
    Let $\{x_i\}_{i=1}^{M}$ be a realisation of an X-sample of size 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
  Let $\{\tilde{x}_{\theta,i}\}_{i=1}^{M}$ be a realisation of an $\tilde{X}_\theta$-sample of size 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

In the context of GANs, the i.i.d. noise process $Z = (Z_t)_{t \in \mathbb{Z}}$ from Definition 4.5 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.

In our GAN framework for stochastic processes, the discriminator is similarly represented by a TCN $d : \mathbb{R}^{N_X \times T^{(d)}} \times \Theta^{(d)} \to [0, 1]$ 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

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

and $X_{1:T^{(d)}}$ and $\tilde{X}_{1:T^{(d)},\theta}$ denote the real and the 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 consider realisations $\{\tilde{x}^{(i)}_{1:T^{(d)},\theta}\}_{i=1}^{M}$ and $\{x^{(i)}_{1:T^{(d)}}\}_{i=1}^{M}$ of samples of size M of the generated neural process and of the target distribution, respectively. Each element of these samples is then inferred into the discriminator to generate a $[0, 1]$-valued output (the classifications), which are then averaged sample-wise to give a Monte Carlo estimate of the discriminator loss function.

5 The model

After we have introduced the main NN topologies in Section 3 and defined GANs in the context of stochastic process generation via TCNs in Section 4, we now turn to the problem of generating financial time series. We start by defining the generator function of Quant GANs: the *stochastic volatility neural network* (SVNN). The induced process of an SVNN will be called *log return neural process* (log return NP). The remainder of this section is devoted to answering the following theoretical aspects of our model:

– Section 5.2: How heavy are the tails generated by a log return NP?
– Section 5.3: Can the log return NP be transformed to model heavy-tails and which assumptions are implied?
– Section 5.4: Can the risk-neutral distribution of log return NPs be derived?
– Section 5.5: Can log return NPs be seen as a natural extension of already existing time-series models?

5.1 Log return neural processes

Log return NPs are inspired by the volatility-innovation decomposition of various stochastic volatility models used in practice Bollerslev (1986); Cont (2001); Heston (1993); Tankov (2003). The corresponding generator architecture, the stochastic volatility neural network, consists of a volatility and drift TCN and another network which models the innovations. Figure 8 illustrates the architecture in use.

Definition 5.1 (Log return neural process)**.** Let $Z = (Z_t)_{t\in\mathbb{Z}}$ be $\mathbb{R}^{N_Z}$-valued i.i.d. Gaussian noise, $g^{(\text{TCN})} : \mathbb{R}^{N_Z \times T^{(g)}} \times \Theta^{(\text{TCN})} \to \mathbb{R}^{2N_X}$ a TCN with RFS $T^{(g)}$ and $g^{(\epsilon)} : \mathbb{R}^{N_Z} \times \Theta^{(\epsilon)} \to \mathbb{R}^{N_X}$ be a network. Furthermore, let $\alpha \in \Theta^{(\text{TCN})}$ and $\beta \in \Theta^{(\epsilon)}$ denote some parameters. A stochastic process R, defined by

$$\begin{aligned} R : \Omega \times \mathbb{Z} \times \Theta^{(\text{TCN})} \times \Theta^{(\epsilon)} &\to \mathbb{R}^{N_X} \\ (\omega, t, \alpha, \beta) &\mapsto \left[\sigma_{t,\alpha} \odot \epsilon_{t,\beta} + \mu_{t,\alpha}\right](\omega) \,, \end{aligned}$$

where $\odot$ denotes the Hadamard product and

$$\begin{aligned} h_t &:= g_\alpha^{(\text{TCN})}\left(Z_{t-T^{(g)}:(t-1)}\right) \\ \sigma_{t,\alpha} &:= |h_{t,1:N_X}| \\ \mu_{t,\alpha} &:= h_{t,(N_X+1):2N_X} \\ \epsilon_{t,\beta} &:= g_\beta^{(\epsilon)}(Z_t) \,, \end{aligned}$$

is called *log return neural process*. The generator architecture defining the log return NP is called *stochastic volatility neural network (SVNN)*. The NPs $\sigma_\alpha := (\sigma_{t,\alpha})_{t\in\mathbb{Z}}$, $\mu_\alpha := (\mu_{t,\alpha})_{t\in\mathbb{Z}}$ and $\epsilon_\beta := (\epsilon_{t,\beta})_{t\in\mathbb{Z}}$ are called *volatility, drift* and *innovation NP*, respectively.

Remark 5.2. For simplicity, we do not distinguish the different NPs' parameters below and just write θ instead of (α, β), and Θ instead of $\Theta^{(\text{TCN})} \times \Theta^{(\epsilon)}$.

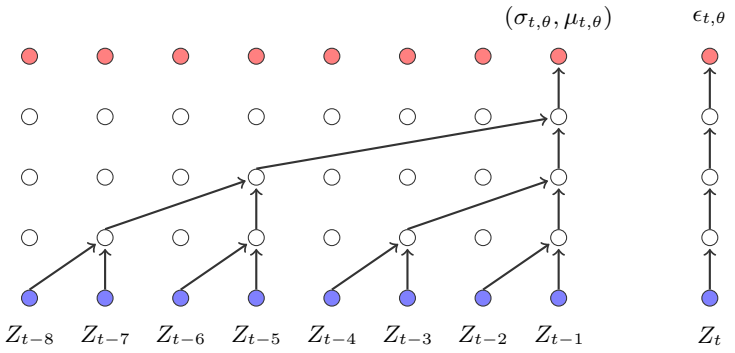


Figure 8: Structure of the SVNN architecture. The volatility and drift component are generated by inferring the latent process $Z_{t-8:t-1}$ through the TCN, whereas the innovation is generated by inferring Z_t.

Remark 5.3 (Independence of the SVNN-NPs). Denote by $(\mathcal{F}_t^Z)_{t\in\mathbb{Z}}$ the natural filtration of the latent process Z and let $t \in \mathbb{Z}$. By construction, $\sigma_{t,\theta}$ and $\mu_{t,\theta}$ are $\mathcal{F}_{t-1}^Z$-measurable. $\epsilon_{t,\theta}$ and $R_{t,\theta}$ are $\mathcal{F}_t^Z$-measurable. Observe further that the random variables $(\sigma_{t,\theta}, \mu_{t,\theta})$ and $\epsilon_{t,\theta}$ are independent, since Z is an i.i.d. Gaussian noise process. As it turns out, the proposed construction is convenient when deriving the transition to the risk-neutral distribution in Section 5.4.

5.2 L^p-space characterization of R_θ

We inspect next whether log return NPs exhibit heavy-tails. We first prove a result concerning generative networks in general and then conclude a corollary for the log return NP.

Theorem 5.4 (L^p-characterization of neural networks)**.** *Let $p \in \mathbb{N}$, $Z \in L^p(\mathbb{R}^{N_Z})$ and $g : \mathbb{R}^{N_Z} \times \Theta \to \mathbb{R}^{N_X}$ a network with parameters $\theta \in \Theta$. Then, $g_\theta(Z) \in L^p(\mathbb{R}^{N_X})$.*

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

$$\|f(x) - f(0)\| \leq L \|x\| \Rightarrow \|f(x)\| \leq L \|x\| + \|f(0)\| \tag{2}$$

as $\|x\| - \|y\| \leq \|x - y\|$ for $x, y \in \mathbb{R}^n$. Now, using the Lipschitz property of neural networks (cf. Remark 3.4), we can apply Equation 2 and as Z is an element of the space $L^p(\mathbb{R}^{N_Z})$ we obtain

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

where L is the networks Lipschitz constant and $\mathbf{0} \in \mathbb{R}^{N_Z}$ the zero vector. This proves the statement. □

Since we assume that our latent process Z is Gaussian i.i.d. noise and thus square-integrable, we conclude from Theorem 5.4 that the mean and the variance of the volatility, drift and innovation NP are finite. Additionally, these properties carry over to the log return NP as the following corollary proves.

Corollary 5.5. *Let R_θ 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 $L^p(\mathbb{R}^{N_X})$.*

Proof. The latent process Z is Gaussian i.i.d. noise. Hence, Theorem 5.4 yields $\sigma_{t,\theta}, \epsilon_{t,\theta}, \mu_{t,\theta} \in L^p(\mathbb{R}^{N_X})$. Since

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

we obtain using the binomial identity

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

where the last inequality derives from the Cauchy-Schwarz inequality. Using the independence and the L^p-property of the volatility and innovation NP (cf. Remark 5.3), we obtain for arbitrary $q \in \mathbb{N}$ that

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

□

Since financial time series are generally considered to exhibit heavy-tails (see Section 2), the existence of all moments is an undesirable property. In the next section, we present a heuristic that works well to preprocess the historical log return process and enables the log return NP to empirically generate heavy-tails.

Last, 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.

Corollary 5.6. *Under the assumptions of Theorem 5.4 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 $N_X = 1$. Using the notation from Definition 3.3 and that g_θ 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 $z \in \mathbb{R}^{Nz}$ 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, $g_{1:k-1,\theta} \coloneqq g_{k-1,\theta} \circ \cdots \circ g_{1,\theta}$ and $D^{(l)}(z) = \operatorname{diag}(\phi'(W^{(l)} g_{1:l-1,\theta}(z)))$ (compare (Goodfellow et al., 2016, Chapter 6.5)). Since the MLP is defined as a composition of Lipschitz functions Theorem 5.4 yields $g_{1:k,\theta}(Z) \in L^p(\mathbb{R}^{N_k})$ for all $k \leq L$. Similarly, the boundedness of ϕ' implies that for an induced matrix norm the random variable $\left\| D^{(l)}(Z) \right\|$ is $\mathbb{P}$-almost surely bounded by some constant $A > 0$ for all $l \leq L$. By applying both properties we obtain

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

with

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

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

Note that Corollary 5.6 does not necessarily hold for a heavy-tailed latent process. Such that using a heavier-tailed latent process may enable the generation of heavy-tails, however may come at the cost of optimization instabilities. We leave it as future work to inspect how preprocessing techniques can be used to stabilize training.

5.3 Generating heavier-tails and modeling assumptions

Theorem 5.4 implies that all moments of the log return NP exist. Furthermore, Wiese et al. (2019b) show that the tails fall at least at a square-exponential rate. Real financial time series are generally considered to exhibit heavy-tails and an unbounded p-th moment for some $p \in (2, 5]$ (Cont, 2001). The Lambert W probability transform, as mentioned in Goerg (2010), can therefore be used to generate heavier tails. The Lambert W probability transform of an $\mathbb{R}$-valued random variable is defined as follows.

Definition 5.7 (Lambert W$\times F_X$)**.** Let $\delta \in \mathbb{R}$ and X be an $\mathbb{R}$-valued random variable with mean μ, standard deviation σ and cumulative distribution function F_X. The location-scale Lambert W$\times F_X$ transformed random variable Y is defined by

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

where $U \coloneqq \dfrac{X - \mu}{\sigma}$ is the normalizing transform.

For $\delta \in [0, \infty)$ the transformation used in Equation 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.

We therefore apply the *inverse Lambert W* probability transform to the asset's log returns and use the principle of quasi maximum likelihood to estimate the model parameters (Goerg, 2010, Section 4.1). The log return NP is then optimized to approximate the inverse Lambert W transformed (lighter-tailed) log return process, from here on denoted by $R^W \coloneqq (R_t^W)_{t \in \mathbb{N}}$.

Using the Lambert W transformed log return process R^W we can formulate our model assumptions, when using SVNNs as the underlying generator.

Assumption 1. The inverse Lambert W transformed spot log returns R^W can be represented by a log return neural process R_θ for some $\theta \in \Theta$.

Assumption 1 has two important implications. First, by construction the log return NP is stationary such that the historical log return process is assumed to be stationary. Second, log return NPs can capture dynamics up to the RFS of the TCN in use. Therefore, Assumption 1 implies for an RFS $T^{(g)}$ that for any $t \in \mathbb{Z}$ the random variables $R_t, R_{t+T^{(g)}+1}$ are independent.

5.4 Risk-neutral representation of R_θ

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

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

The spot prices are then defined recursively by

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

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

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

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 $\mathcal{F}^Z_{t-1}$-measurable and get

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

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 $\mathcal{F}^Z_{t-1}$-measurable and $\epsilon_{t,\theta}$ is independent of $\mathcal{F}^Z_{t-1}$, we can write

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

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* $R^M_{t,\theta}$ as

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

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

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

and defines a martingale.

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

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

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.

5.5 Constrained log return neural processes

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 ($N_X = 1$), 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:

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

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

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

and the discounted risk-neutral price process fulfils the recursion

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

In particular, solving this recursion gives the explicit representations

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

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

Remark 5.8 (Comparison to the Black-Scholes model). In the one-dimensional Black-Scholes model, the risk-neutral distribution of the price process is given by

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

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 $\mathrm{GARCH}(p,q)$ 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.

6 Preprocessing

Prior to passing a realization of a financial time series $s_{0:T} \in \mathbb{R}^{N_X \times (T+1)}$ to the discriminator, the time series has to be preprocessed. The applied pipeline is displayed in Figure 9. We briefly explain each of the steps taken. Note that all of the used transformations, excluding the rolling window, are invertible and thu, allow a series sampled from a log return NP to be post-processed by inverting the steps 1-4 to obtain the desired form. Also, observe that the pipeline includes the inverse Lambert W transformation as earlier discussed in subsection 5.3.

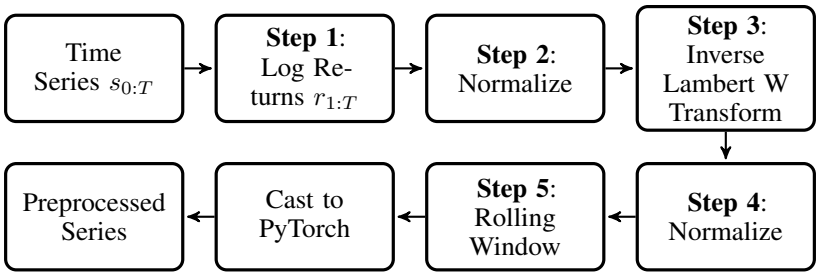


Figure 9: Condensed representation of the preprocessing pipeline.

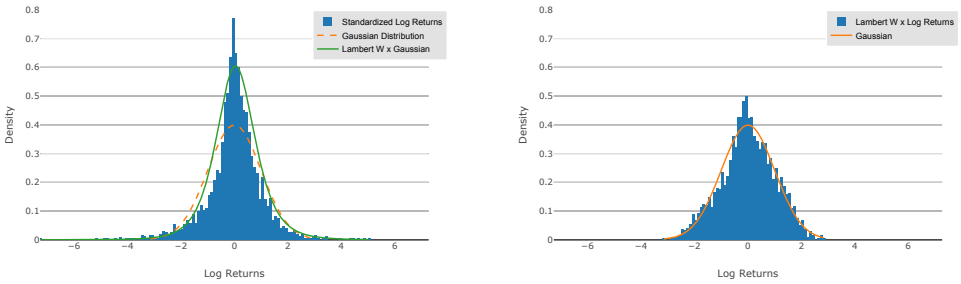
Step 1: Log returns $r_{1:T}$

Calculate the log return series

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

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 LeCun et al. (1998).



(a) Standardized Log Returns

(b) Lambert W x Log Returns

Figure 10: (a) The original S&P 500 log returns and the fitted probability density function of a Lambert W x Gaussian random variable. (b) The inverse Lambert W transformed log returns and the probability density function of a Gaussian random variable.

Step 3: Inverse Lambert W transform

The suggested transformation applied to the log returns of the S&P 500 is displayed in Figure 10. It shows the standardized original distribution of the S&P 500 log returns and the inverse Lambert W transformed log return distribution. Observe that the transformed standardized log return distribution in Figure 10b 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 one to the preprocessed log return sequence $r_t^{(\rho)}$. Hence, for $t \in \{1, \ldots, T - T^{(d)}\}$ we define the sub-sequences

$$r^{(t)}_{1:T^{(d)}} := r^{(\rho)}_{t:(T^{(d)}+t-1)} \in \mathbb{R}^{N_Z \times T^{(d)}}\,.$$

Remark 6.1. 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.

7 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 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.

7.1 Setting and implementation

The implementation was written with the programming language python. NN architectures were implemented by employing the python package pytorch (Paszke et al., 2017, 2019), an automatic differentiation library primarily used for NN computations and optimization. The training time of the NNs 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 Jones et al. (2001)[4].

The architecture of the TCN used for the generator and the discriminator of the Quant GAN models is an extension of the architecture proposed in (Bai et al., 2018, Section 3) by using skip connections (cf. van den Oord et al. (2016), Definition 3.15). A detailed description of the architecture is given in Appendix B. During training of the generator and discriminator, we applied the *GAN stability algorithm* from Mescheder et al. (2018).

7.2 Models

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

In contrast to other approaches in the literature, the proposed models are based on TCNs instead of LSTMs. Note that this is mainly motivated by the superior performance of convolutional architectures to typical recurrent architectures in many sequence related tasks (Bai et al., 2018). Therefore, we do not consider LSTM based models here and leave this as future work.

Pure TCN

To evaluate the capabilities of a pure TCN model, we model the log returns directly using a TCN with receptive field size $T^{(g)} = 127$ as the generator function, i.e. the return process is given by

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

for the three-dimensional noise prior $Z_t \overset{iid}{\sim} \mathcal{N}(\mathbf{0}, I)$, $(N_Z = 3)$.

Constrained log return NP

Assume a constrained log return NP (see Definition 5.1 and Section 5.5)

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

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. Here again the latent process is i.i.d. Gaussian noise $Z_t \overset{iid}{\sim} \mathcal{N}(\mathbf{0}, I)$ for $N_Z = 3$. The innovation process takes the form $\epsilon_{t,\theta} = Z_{t,1}$ for any $t \in \mathbb{Z}$.

GARCH(1,1) with constant drift

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

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

[4]Note that preprocessing and approximation of the Lambert W transform relevant parameters can be equivalently done by using the package pytorch.

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 Bollerslev (1986).

7.3 Evaluating a path simulator: metrics and scores

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

7.3.1 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 by

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

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 Villani (2008).

DY metric Additionally we compute the *DY metric* proposed in Drăgulescu and Yakovenko (2002). The DY metric is for $t \in \mathbb{N}$ defined by

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

where P_t^h and P_t^g denote the empirical probability 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

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

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.

7.3.2 Dependence scores

ACF score The ACF score is proposed to compare the dependence properties of the historical and the generated time series. Let $r_{1:T}$ denote the historical log return series and $\{r^{(1)}_{1:\tilde{T},\theta}, \ldots, r^{(M)}_{1:\tilde{T},\theta}\}$ a set of generated log return paths of length $\tilde{T} \in \mathbb{N}$. The autocorrelation is defined as a function of the time lag τ and a series $r_{1:T}$ and measures the correlation of the lagged time series with the series itself

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

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

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

where the function f is applied element-wise to the series. The ACF score is computed for the functions $f(x) = x$, $f(x) = x^2$ and $f(x) = |x|$ and constants $S = 250$, $M = 500$, $\tilde{T} = 4000$.

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

$$\mathcal{L}(\tau; r) = \mathrm{Corr}\left(r^2_{t+\tau}, r_t\right)$$

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

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

In the benchmark, the leverage effect score is computed for $S = 250$, $M = 500$ and $\tilde{T} = 4000$; the same as for the ACF score.

7.4 Generating the S&P 500 index

We consider daily spot-prices of the S&P 500 from May 2009 until December 2018. As expected, the GAN training was very irregular and did not converge to a local optimum of the objective function. Instead for each GAN model, several learned parameter sets were saved and the best setup was chosen based on the evaluation metrics described in subsection 7.3. For each model, we only present the results of the best performing setup. In the appendix, we show:

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

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

time series	time span	# of observations	ADF-statistic	p-value
S&P 500	May 2009 - December 2018	2413	-10.87	1.36×10^{-19}

Table 1: Considered financial time series.

We validate Assumption 1 by applying the augmented Dickey-Fuller test (Mushtaq, 2011) to the time series and obtain a test statistic of -10.87 and a p-value of 1.36×10^{-19}, which is a strong indication that the series is stationary. Further as expected, the Lambert W transformation significantly helped the reported models to capture the stylized facts of the asset returns; in particular in modeling the tails of the distribution.

7.4.1 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.

7.4.2 Constrained SVNN

The C-SVNN performs comparable to the TCN, but slightly worse in most of the evaluated metrics. This is expectable as the C-SVNN has less degrees of freedom due to the restrictions compared to the pure TCN.

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.

Despite the restrictions, the evaluated metrics of the C-SVNN in Table 2 are nearly as good as the results of the TCN. Furthermore, the C-SVNN also outperforms GARCH(1,1) model significantly.

Recall further that the SVNN has the structural advantages that the volatility can be directly modeled and a transition to its martingale distribution is known.

7.4.3 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 2), 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)
$\mathrm{EMD}(1)$	**0.0039**	0.0040	0.0199
$\mathrm{EMD}(5)$	**0.0039**	0.0040	0.0145
$\mathrm{EMD}(20)$	**0.0040**	0.0069	0.0276
$\mathrm{EMD}(100)$	**0.0154**	0.0464	0.0935
$\mathrm{DY}(1)$	**19.1199**	19.8523	32.7090
$\mathrm{DY}(5)$	**21.1167**	21.2445	27.4760
$\mathrm{DY}(20)$	26.3294	**25.0464**	39.3796
$\mathrm{DY}(100)$	28.1315	**25.8081**	46.4779
$\mathrm{ACF}(\mathrm{id})$	**0.0212**	0.0220	0.0223
$\mathrm{ACF}(\lvert\cdot\rvert)$	**0.0248**	0.0287	0.0291
$\mathrm{ACF}((\cdot)^2)$	**0.0214**	0.0245	0.0253
Leverage Effect	**0.3291**	0.3351	0.4636

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

8 Conclusion and future work

In this paper we showed that recently developed NN architectures can be used in an adversarial modeling framework to approximate financial time series in discrete-time. Although these methods have been notoriously hard to train, advances in GANs showed that they can deliver competitive results and, as GAN training procedures develop, 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 and addressed, Quant GANs offer a data-driven method that surpasses the performance of other conventional models from mathematical finance.

References

Martín Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. URL https://openreview.net/forum?id=Hk4_qw5xe.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018. URL http://arxiv.org/abs/1803.01271.

Sebastian Becker, Patrick Cheridito, and Arnulf Jentzen. Deep optimal stopping. *arXiv*, abs/1804.05394, 2018.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3):637–54, 1973. URL https://EconPapers.repec.org/RePEc:ucp:jpolec:v:81:y:1973:i:3:p:637-54.

Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986. URL https://EconPapers.repec.org/RePEc:eee:econom:v:31:y:1986:i:3:p:307-327.

Hans Buehler and Evgeny Ryskin. Discrete local volatility for large time steps (extended version). *Available at SSRN 2642630*, 2017.

Hans Buehler, Lukas Gonon, Josef Teichmann, and Ben Wood. Deep hedging. *Quantitative Finance*, pages 1–21, 02 2019a. doi: 10.1080/14697688.2019.1571683.

Hans Buehler, Lukas Gonon, Ben Wood, Josef Teichmann, Baranidharan Mohan, and Jonathan Kochems. Deep hedging: Hedging derivatives under generic market frictions using reinforcement learning-machine learning version. *Available at SSRN*, 2019b.

Anirban Chakraborti, Ioane Muni Toke, Marco Patriarca, and Frédéric Abergel. Econophysics review: I. empirical facts. *Quantitative Finance*, 11(7):991–1012, 2011. doi: 10.1080/14697688.2010.539248. URL https://doi.org/10.1080/14697688.2010.539248.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp 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. URL http://arxiv.org/abs/1511.07289.

Rama Cont. Empirical properties of asset returns: stylized facts and statistical issues. *Quantitative Finance*, 1 (2):223–236, 2001. URL https://EconPapers.repec.org/RePEc:taf:quantf:v:1:y:2001:i:2:p:223-236.

Sander Dieleman, Aaron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7989–7999. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8023-the-challenge-of-realistic-music-generation-modelling-raw-audio-at-scale.pdf.

Adrian A Drăgulescu and Victor M Yakovenko. Probability distribution of returns in the heston model with stochastic volatility. *Quantitative Finance*, 2(6):443–453, 2002. doi: 10.1080/14697688.2002.0000011. URL https://doi.org/10.1080/14697688.2002.0000011.

Omar El Euch, Jim Gatheral, and Mathieu Rosenbaum. Roughening heston. *Available at SSRN 3116887*, 2018.

Georg 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, 10 2010.

Ian Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1319–III–1327. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3043084.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Ishaan Gulrajani, Faruk Ahmed, Martín Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of wasserstein gans. *CoRR*, abs/1704.00028, 2017. URL http://arxiv.org/abs/1704.00028.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 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, pages 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123. URL http://dx.doi.org/10.1109/ICCV.2015.123.

Steven L. Heston. A closed-form solution for options with stochastic volatility with applications to bond and currency options. *Review of Financial Studies*, 6:327–343, 1993.

Sepp Hochreiter and Juergen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(91)90009-T. URL http://www.sciencedirect.com/science/article/pii/089360809190009T.

Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. *CoRR*, abs/1807.00734, 2018. URL http://arxiv.org/abs/1807.00734.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001. http://www.scipy.org/".

Andrej Karpathy. Stanford university CS231n: Convolutional neural networks for visual recognition, 2019. http://cs231n.stanford.edu/syllabus.html.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515.

Adriano Soares Koshiyama, Nick Firoozye, and Philip C. Treleaven. Generative adversarial networks for financial trading strategies fine-tuning and combination. *CoRR*, abs/1901.01751, 2019. URL http://arxiv.org/abs/1901.01751.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc. URL http://dl.acm.org/citation.cfm?id=2999134.2999257.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50. Springer, 1998. ISBN 3-540-65311-2. URL http://dl.acm.org/citation.cfm?id=645754.668382.

Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International Conference on Machine learning (ICML)*, 2018.

Rizwan Mushtaq. Augmented dickey fuller test. 2011.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress. ISBN 978-1-60558-907-7. URL http://dl.acm.org/citation.cfm?id=3104322.3104425.

Fernando Pardo. Enriching financial datasets with generative adversarial networks. 2019.

Fernando De Meer Pardo and Rafael Cobo López. Mitigating overfitting on financial datasets with generative adversarial networks. *The Journal of Financial Data Science*, 2019. ISSN 2405-9188. doi: 10.3905/jfds.2019.1.019. URL https://jfds.pm-research.com/content/early/2019/11/26/jfds.2019.1.019.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1310–1318, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/pascanu13.html.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Marco Schreyer, Timur Sattarov, Damian Borth, Andreas Dengel, and Bernd Reimer. Detection of anomalies in large scale accounting data using deep autoencoder networks. *CoRR*, abs/1709.05254, 2017. URL http://arxiv.org/abs/1709.05254.

Marco Schreyer, Timur Sattarov, Bernd Reimer, and Damian Borth. Adversarial learning of deepfakes in accounting. In *NeurIPS 2019 Workshop on Robust AI in Financial Services: Data, Fairness, Explainability, Trustworthiness, and Privacy*, Dezember 2019a. URL https://www.alexandria.unisg.ch/258090/.

Marco Schreyer, Timur Sattarov, Christian Schulze, Bernd Reimer, and Damian Borth. Detection of accounting anomalies in the latent space using adversarial autoencoder neural networks. In *2nd KDD Workshop on Anomaly Detection in Finance, 2019*, August 2019b. URL https://www.alexandria.unisg.ch/257633/. Code available at: https://github.com/GitiHubi/deepAD.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Shuntaro Takahashi, Yu Chen, and Kumiko Tanaka-Ishii. Modeling financial time-series with generative adversarial networks. *Physica A: Statistical Mechanics and its Applications*, 527:121261, August 2019. doi: 10.1016/j.physa.2019.121261. URL https://doi.org/10.1016/j.physa.2019.121261.

Peter Tankov. *Financial modelling with jump processes*, volume 2. CRC press, 2003.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv*, abs/1609.03499, 2016.

C Villani. *Optimal transport – Old and new*, pages xxii+973. 01 2008. doi: 10.1007/978-3-540-71050-9.

Magnus Wiese, Lianjun Bai, Ben Wood, and Hans Buehler. Deep hedging: learning to simulate equity option markets. *Available at SSRN 3470756*, 2019a.

Magnus Wiese, Robert Knobloch, and Ralf Korn. Copula & Marginal Flows: Disentagling the Marginal from its Joint. *arXiv*, abs/1907.03361, 2019b.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL http://arxiv.org/abs/1505.00853.

A Numerical Results

A.1 Pure TCN

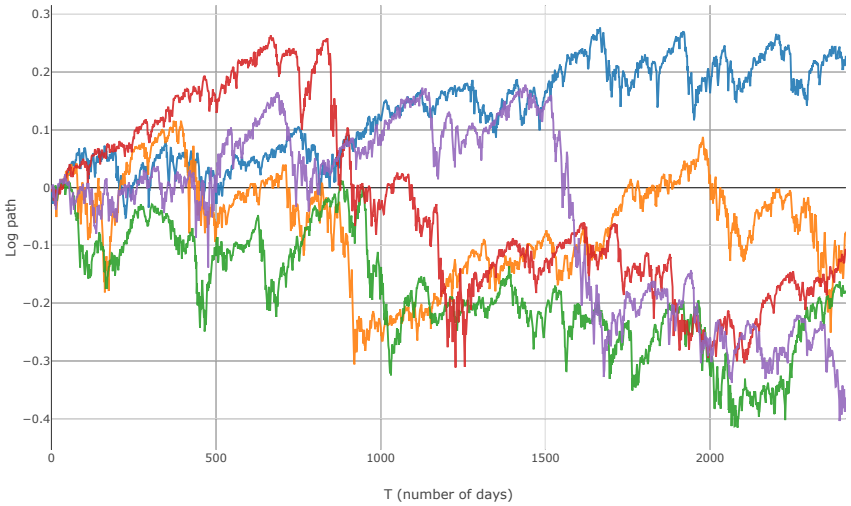


Figure A.1: 5 generated log paths

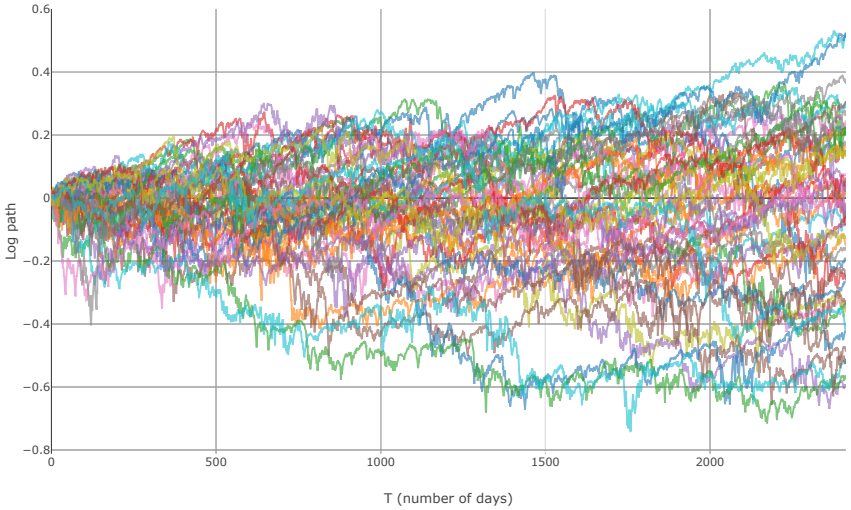
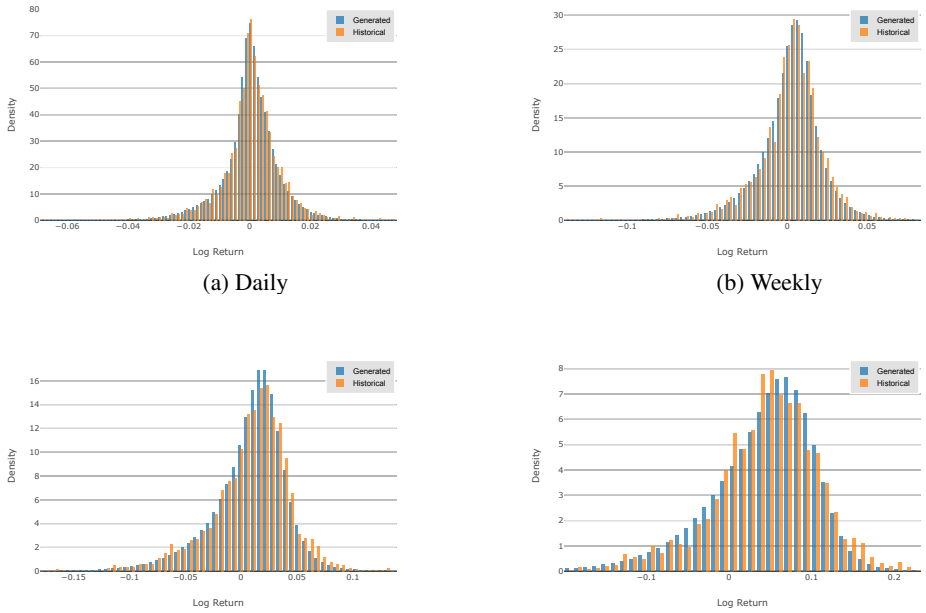


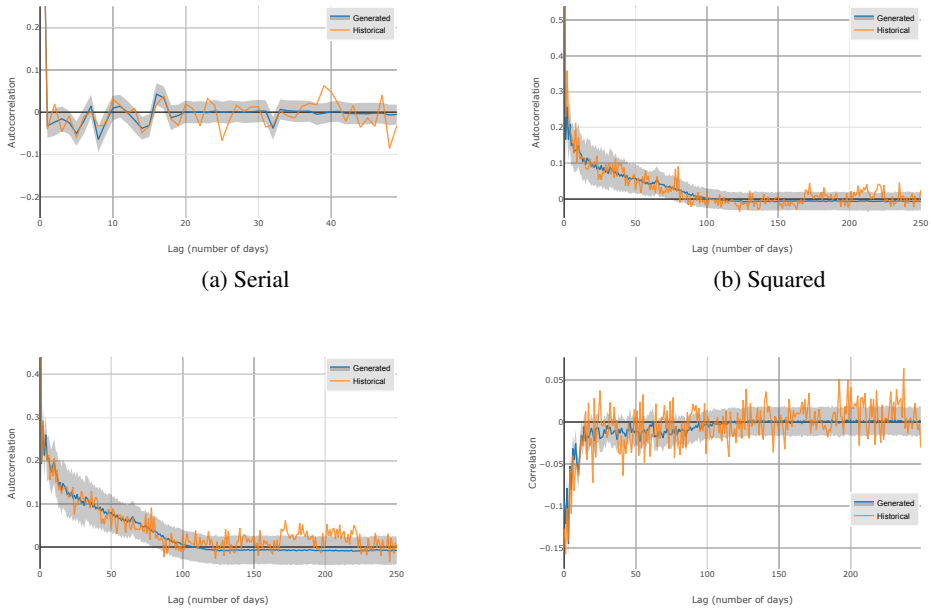
Figure A.2: 50 generated log paths



(a) Daily (b) Weekly

(c) Monthly (d) 100-day

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



(a) Serial (b) Squared

(c) Absolute (d) Leverage Effect

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

A.2 Constrained SVNN

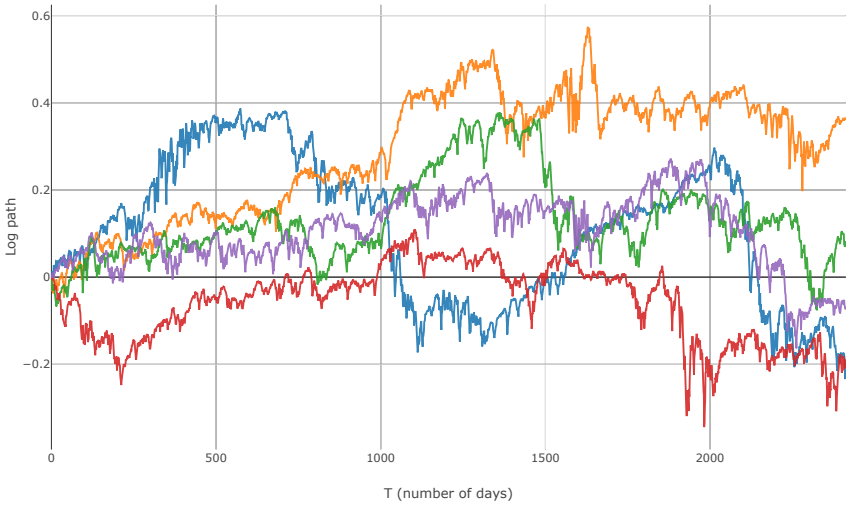


Figure A.5: 5 generated log paths

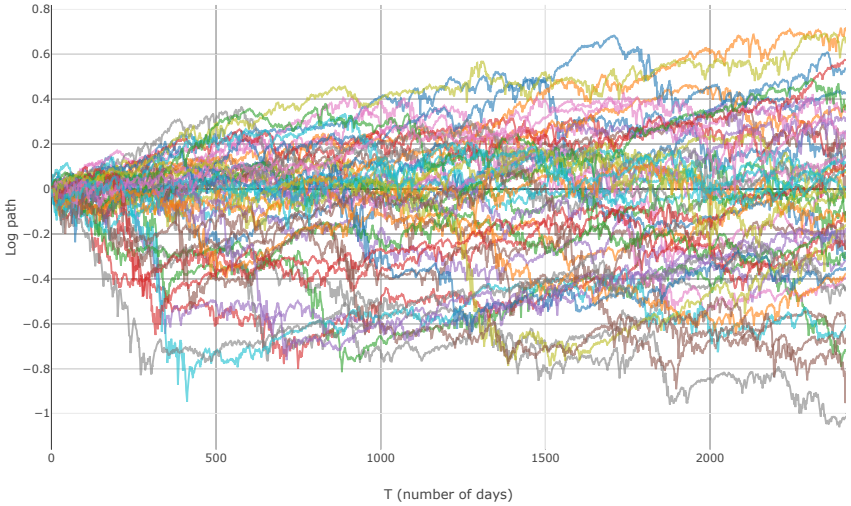
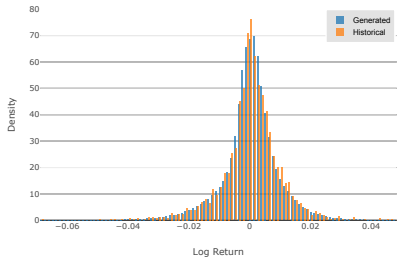
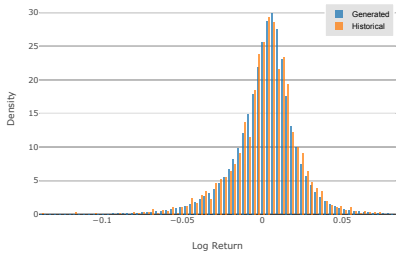


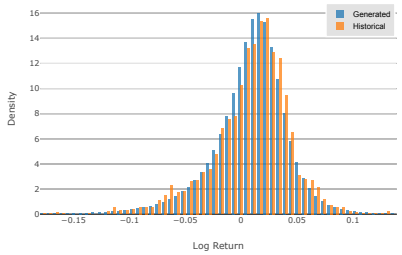
Figure A.6: 50 generated log paths



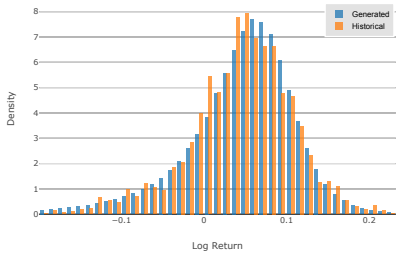
(a) Daily



(b) Weekly

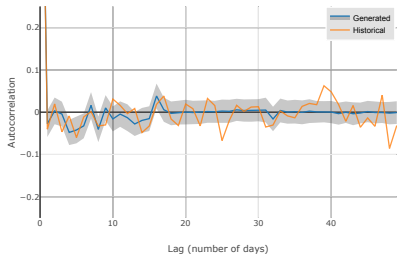


(c) Monthly

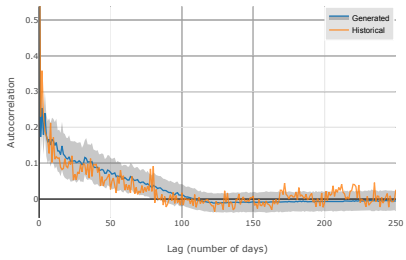


(d) 100-day

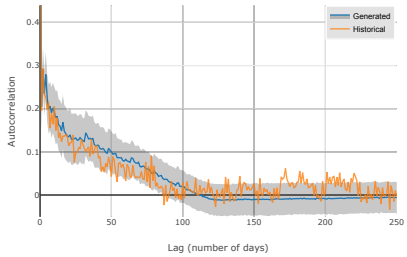
Figure A.7: Comparison of generated and historical densities of the S&P500.



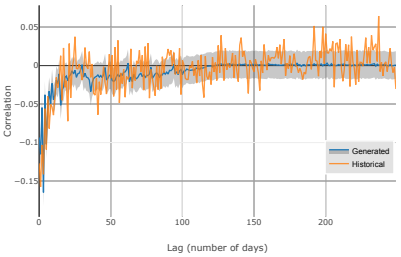
(a) Serial



(b) Squared



(c) Absolute



(d) Leverage Effect

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

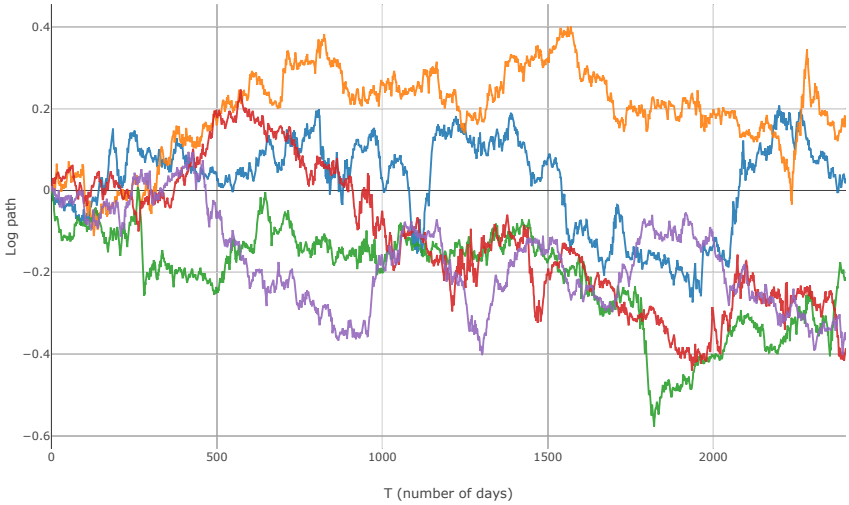


Figure A.9: 5 generated log paths

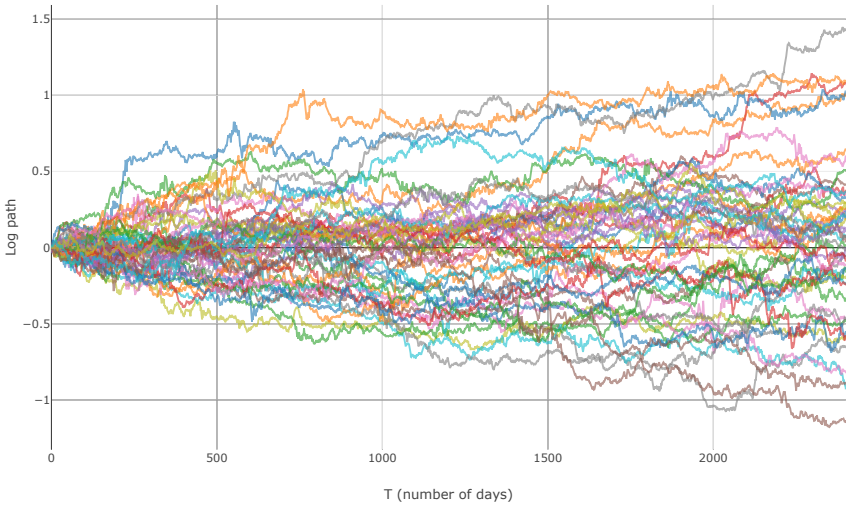
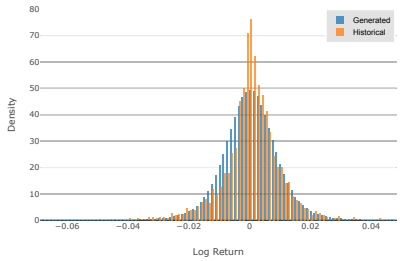
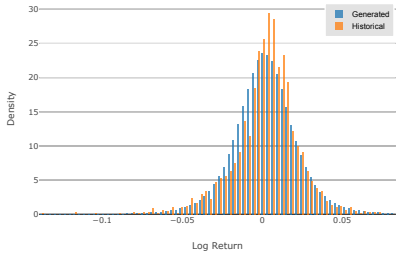


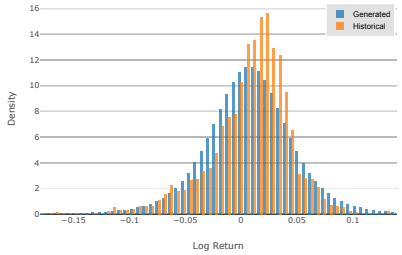
Figure A.10: 50 generated log paths



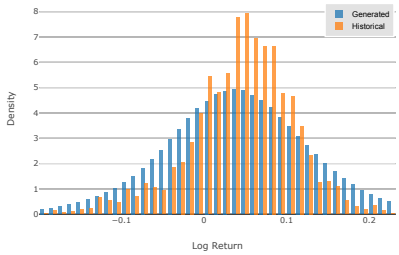
(a) Daily



(b) Weekly

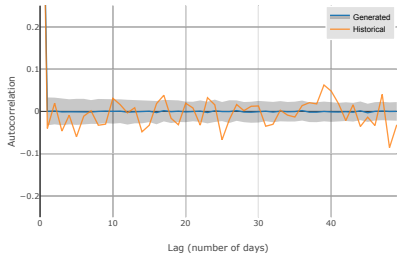


(c) Monthly

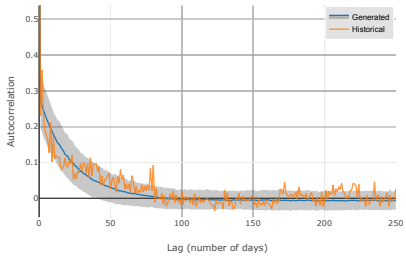


(d) 100-day

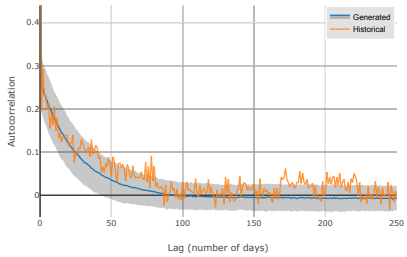
Figure A.11: Comparison of generated and historical densities of the S&P500.



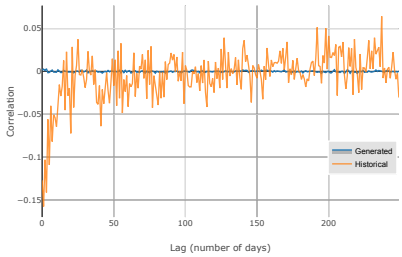
(a) Serial



(b) Squared



(c) Absolute



(d) Leverage Effect

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

B Architecture

For both the generator and the discriminator we used TCNs with skip connections. Inside the TCN architecture *temporal blocks* were used as block modules. A temporal block consists of two dilated causal convolutions and two PReLUs (He et al., 2015) as activation functions. The primary benefit of using temporal blocks is to make the TCN more expressive by increasing the number of non-linear operations in each block module. A complete definition is given below.

Definition B.1 (Temporal block)**.** Let $N_I, N_H, N_O \in \mathbb{N}$ denote the input, hidden and output dimension and let $D, K \in \mathbb{N}$ denote the dilation and the kernel size. Furthermore, let w_1, w_2 be two dilated causal convolutional layers with arguments (N_I, N_H, K, D) and (N_H, N_O, K, D) respectively and let $\phi_1, \phi_2 : \mathbb{R} \to \mathbb{R}$ be two PReLUs. The function $f : \mathbb{R}^{N_I \times (2D(K-1)+1)} \to \mathbb{R}^{N_O}$ defined by

$$f(X) = \phi_2 \circ w_2 \circ \phi_1 \circ w_1(X)$$

is called *temporal block* with arguments (N_I, N_H, N_O, K, D).

The TCN architecture used for the generator and the discriminator in the pure TCN and C-SVNN model is illustrated in Table 3. Table 4 shows the input, hidden and output dimensions of the different models. Here, G abbreviates the generator and D the discriminator. Note that for all models, except the generator of the C-SVNN, the hidden dimension was set to eighty. The kernel size of each temporal block, except the first one, was two. Each TCN modeled a RFS of 127.

Module name	Arguments
Temporal block 1	$(N_I,\ N_H,\ N_H,\ 1,\ 1)$
Temporal block 2	$(N_H,\ N_H,\ N_H,\ 2,\ 1)$
Temporal block 3	$(N_H,\ N_H,\ N_H,\ 2,\ 2)$
Temporal block 4	$(N_H,\ N_H,\ N_H,\ 2,\ 4)$
Temporal block 5	$(N_H,\ N_H,\ N_H,\ 2,\ 8)$
Temporal block 6	$(N_H,\ N_H,\ N_H,\ 2,\ 16)$
Temporal block 7	$(N_H,\ N_H,\ N_H,\ 2,\ 32)$
1×1 Convolution	$(N_H,\ N_O,\ 1,\ 1)$

Table 3: TCN architecture of the reported models. Note that the TCN architecture includes skip connections.

Models	Pure TCN - G	Pure TCN - D	C-SVNN - G	C-SVNN - D
N_I	3	1	3	1
N_H	80	80	50	80
N_O	1	1	2	1

Table 4: Configuration of the generator (G) and discriminator (D) of the *pure TCN* and *C-SVNN* model.