WASSERSTEIN GAN: DEEP GENERATION APPLIED ON FINANCIAL TIME SERIES

A PREPRINT

Pfenninger Moritz¹ pfennmor@students.zhaw.ch Bigler Nico¹ bigleda1@students.zhaw.ch

Samuel Rikli¹ riklisam@students.zhaw.ch Joerg Osterrieder^{1,2} ¹joerg.osterrieder@zhaw.ch ²joerg.osterrieder@utwente.nl

¹School of Engineering Zurich University of Applied Sciences Winterthur, Switzerland

²The Hightech Business and Entrepreneurship Group Faculty of Behavioural, Management and Social Sciences University of Twente Enschede, Netherlands

July 14, 2021

ABSTRACT

Modeling financial time series is challenging due to their high volatility and unexpected happenings on the market. Most financial models and algorithms trying to fill the lack of historical financial time series struggle to perform and are highly vulnerable to overfitting. As an alternative, we introduce in this paper a deep neural network called the WGAN-GP, a data-driven model that focuses on sample generation. The WGAN-GP consists of a generator and discriminator function which utilize an LSTM architecture. The WGAN-GP is supposed to learn the underlying structure of the input data, which in our case, is the Bitcoin. Bitcoin is unique in its behavior; the prices fluctuate what makes guessing the price trend hardly impossible. Through adversarial training, the WGAN-GP should learn the underlying structure of the bitcoin and generate very similar samples of the bitcoin distribution. The generated synthetic time series are visually indistinguishable from the real data. But the numerical results show that the generated data were close to the real data distribution but distinguishable. The model mainly shows a stable learning behavior. However, the model has space for optimization, which could be achieved by adjusting the hyperparameters.

Keywords WGAN-GP, Wasserstein, Time Series Classification, Generative Adversarial Networks, Financial Time Series, Bitcoin

^{*}Financial support by the Swiss National Science Foundation within the project "Mathematics and Fintech - the next revolution in the digital transformation of the Finance industry" is gratefully acknowledged by the corresponding author. This research has also received funding from the European Union's Horizon 2020 research and innovation program FIN-TECH: A Financial supervision and Technology compliance training program under the grant agreement No 825215 (Topic: ICT-35-2018, Type of action: CSA). Moreover, this article is based upon the work from the Innosuisse Project 41084.1 IP-SBM Towards Explainable Artificial Intelligence and Machine Learning in Credit Risk Management. Furthermore, this article is based upon work from COST Action 19130 Fintech and Artificial Intelligence in Finance, supported by COST (European Cooperation in Science and Technology), www.cost.eu (Action Chair: Joerg Osterrieder). The authors are grateful to management committee members of the COST Action CA19130 Fintech and Artificial Intelligence in Finance and Industry, which took place at Zurich University of Applied Sciences, Switzerland, in September 2020.

Contents

1	Introductions				
2	Lite	Literature review			
3	3 Neural network				
	3.1	Recurrent Neural Network	6		
	3.2	Single Layer Perceptron (SLP)	7		
	3.3	Multi layer perceptron	8		
	3.4	Backpropagation	9		
	3.5	Activation functions	9		
	3.6	Optimizers - RMSprop	10		
	3.7	Loss functions	10		
	3.8	Learning rate	11		
	3.9	Batch size	11		
	3.10) Long Short-Term Memory	11		
4	Gen	erative Adversarial Network (GAN)	12		
	4.1	Discriminator loss	13		
	4.2	Generator loss	13		
	4.3	Challenges in the training process	13		
		4.3.1 Training of the Discriminator	13		
		4.3.2 Training of the Generator	13		
		4.3.3 Mode Collapse	13		
		4.3.4 Vanishing problem	14		
	4.4	Implementation of a GAN	14		
	4.5	The Wasserstein GAN	14		
	4.6	Lipschitz Constraint	15		
	4.7	Wasserstein GAN with gradient penalty term	16		
5 Theoretical evaluation methods		oretical evaluation methods	17		
	5.1	QQ-Plot	17		
	5.2	ACF	17		
6 Descriptive Analysis		criptive Analysis	18		
	6.1 Introduction for data and Bitcoin		18		
		6.1.1 Bitcoin	18		
		6.1.2 Data	18		
	6.2	Log returns	19		
	6.3	Fat tails	19		
	6.4	Volatility Clusters	20		

	6.5	Skewness and kurtosis	21
7	Our	approach to Wasserstein GAN with gradient penalty and LSTM	22
	7.1	Data processing and Implementation	22
	7.2	Parameter settings	23
	7.3	Architecture	23
	7.4	Experimental Results	24
	7.5	Distribution Comparison	25
	7.6	Autocorrelation (ACF)	27
	7.7	Loss function	28
8	Con	clusion	28

1 Introductions

Sharing data has become more and more difficult due to stricter regulations around data management to control privacy requirements. Until recently, natural data is anonymized due to privacy risks. Nowadays, data can generated artificially using statistical parts of real data. Synthetic data generation has received considerable focus, owing to its benefits of addressing the data access restrictions by preserving the real data's multivariate relationships and statistical integrity. Synthetic data minimizes the disclosure risks and has the potential to show high usability by capturing the essential relationships and distributions from realistic data. Because of these reasons, the generation of fully synthetic data has gained more popularity and could be a next-step solution to real-world data sharing problems.

Generative Adversarial Networks (GAN) are a group of unsupervised learning algorithms in computer science. Dr. Ian Goodfellow and his colleagues first introduced it in 2014 to build a generative adversarial Network architecture that creates new data instances that resemble the training data to create images.



Figure 1: First-ever created fake images by a GAN [56]

They can also use them to make data anonymous, which can be helpful to data containing personal content. There have already been approaches such as unknown data processing. The processing of personal data is a significant hurdle in machine learning. However, GANs are a promising approach for data set anonymization and can open the door for companies to process personal data. [9] While the first GANs focused on images with faces, the GANs are meanwhile able to generate ideas of almost any kind in high resolution. The first fake-generated images are demonstrated in [Fig. 1]. It has been an important milestone in deep learning and the starting signal for further research on the topic of GAN. Over time, new implementations have been created that can testify to image quality and credibility in the authenticity of images. In the first place, it created low-resolution images, wherein the training process of the resolution is increased step by step. [10] [11] From all the GANs, there is only a fragment of the total GAN methods dealing with one-dimensional (1D) continuous-valued time series data. Throughout this paper, we reference to time series with a 1D continuous time-series database.

In this paper, we like to carry out the following:

- Introduction to the neural networks up to the GAN
- Implementation of the Wasserstein-GAN with gradient penalty and LSTM architecture applied on bitcoin.
- The results are evaluated by comparing the generated synthetic distribution against the historic data distribution.

2 Literature review

After a careful literature study on financial modeling, we understood that generating data may increase training models' performance in different situations across multidisciplinary financial activities [64]. To generate synthetic data, a model needs to learn the underlying distribution of the real data. The statistical properties of classic asset returns are a well-studied topic and referred to as Stylized Facts. The most important Stylized Facts can be captured in [59]. In this thesis, the goal is to synthesize Bitcoin's asset returns, which is the most traded cryptocurrency. It represents a high-risk investment and differs from traditional assets from a statistical point of view. Characterization of the parametric distributions of Bitcoin was provided by Stephen Chan et al. (2017) [60]. Time series [62] forecasting is a challenging task. There are many methods, such as the traditional ones like the ARIMA, AR, or the exponential smoothing, which only operate on a small-time series. Over the last decades, however, many companies have collected big sets of data [63]. The traditional methods that consider individual time series in isolation or operate on small-time series can fail to perform on such big data. This has brought the urge for new techniques to process all the available data and produce reliable forecasts. Deep neural networks have gained popularity in time series forecasting because they benefit from the massive amounts of data and may be a good alternative to many other machine learning and statistical techniques. [62] The synthetization of Bitcoin returns will be accomplished by a rather new variant of GANs generative model. It is a class of machine learning frameworks designed by Ian Goodfellow. [56] There are already some papers that focus on the implementations of GAN to synthesize financial time series. Some of them are here shown. [66–68] There have been a lot more, of which some of them already achieved satisfactory results. In Figure 2, there are all different kinds of GAN setups over time. Each implementation is somehow jointly responsible for the continuous further development of GANs.

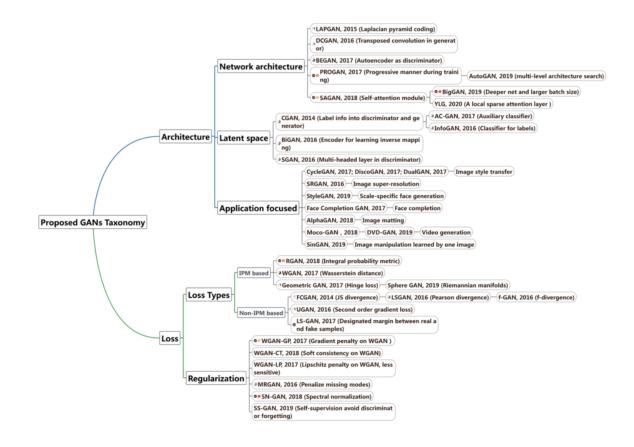


Figure 2: Different kind of GANs [57]

Because GANs often suffer from mode collapse during training, we introduce the improved GAN called Wasserstein GAN to improve learning stability. The papers [28–30] focus on implementing a Wasserstein GAN and show differences to the original GAN [57] and optimize them in specific ways. Although the WGAN makes progress toward stable training, it can sometimes fail to converge or generates poor sample quality. [28]

There are a lot of neural networks, such as the feed-forward networks. Feed-forward neural networks associate input with output and allow the signal to travel only one way, where there are no feedback loops. For normal GAN architectures, it is difficult to detect any patterns in financial time series like Bitcoin. They change with time, and it is difficult to detect any patterns due to long-term trends, seasonal and cyclical fluctuations, and random noise. However, the biggest challenge is noise and the need to make them stationary. [61][65] Recurrent neural networks have signals that travel in both directions between the input and output layers, which makes them highly complicated as powerful. However, RNNs suffer from vanishing gradients in training. This problem can cause the model to slow down or even stall. Long-Short Term Memory networks (LSTM), created in 1990, can fix this problem due to their ability to store information in a more extended period and learn from inputs separated from each other by long time lags. [65] Long Short Term Memory Networks (LSTM) can capture patterns in the time series data, making them valuable tools to predict predictions regarding the future trend of the data.

3 Neural network

Neural networks are a subset of machine learning and are at the heart of deep learning algorithms. The human brain inspires their name and structure. Deep neural networks [Fig. 3] compromises of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node (also called perceptron) is connected to another and has an associated weight and threshold. If the output of any individual perceptron is above the specified threshold value, that perceptron is activated, sending data to the next layer of the network. Otherwise, no data passes along the next layer of the network.

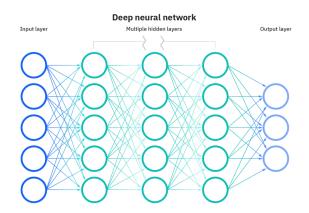


Figure 3: Structure of a deep neural network [13]

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms achieve high accuracy, they are potent tools in computer science and artificial intelligence, allowing us to classify and cluster data. [12]

3.1 Recurrent Neural Network

A recurrent neural network is a type of artificial neural network that uses sequential data or time-series data. It stores the output activations from the layers of the network. Then, the next time we feed an input example to the network, we include the previously-stored outputs as additional inputs. Like feed-forward, recurrent neural networks use training data to learn. Recurrent Neural Networks, however, can memorize previous inputs when a huge set of sequential data is fed. A deep neural network assumes that inputs and outputs are independent of each other. The output of a recurrent neural network depends on the primary elements within its sequences. [13][49]

Even though RNNs are pretty powerful, they suffer from vanishing gradient problem, which hinders them from using long term information. They are useful for storing memories of 3 or 4 instances of past iterations. More enormous numbers of instances will not provide good results. [50] That is why we use a better variation of RNNs: Long Short Term Networks (LSTM).

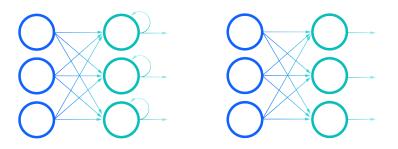
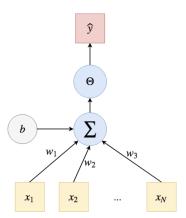


Figure 4: Recurrent neural network [13]

3.2 Single Layer Perceptron (SLP)

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers, deciding whether an input vector is some specific class. Frank Rosenblatt firstly introduced the simplest form of a neural network in 1958. It consists of a single artifical neuron with adjustable weights and a treshold that will convert an input vector into an output vector and represent a simple associative memory. [13][14] The neuron consists of two mathematical functions: A calculation of the network input and an activation function that decides whether the calculated net input now "light up" or not. Therefore, it is binary in its output: Think of a small light that depends on the input values and weights, an input (sum) is formed, and then a function decides whether the light is lit. This concept of output generation is called feed-forward propagation. [51] Let's consider the structure of the perceptron. It contains four key components:



- 1. input x_i
- 2. weights w_i
- 3. weighted sum \sum
- 4. thresholding value by using an activation function

Figure 5: Computational graph

By using the weighted summing function, the perceptron becomes a learnable parameter. By adjusting the weights, the perceptron can differentiate between two classes and therefore model the classes. The weighted sum is transmitted through an activation function θ . Activation functions decide to activate or deactivate the neurons to get a desired output. The \sum represents the linear combination of the inputs $X(x_1, x_2)$ of length n that is weighted by a weight vector w.

$$x_1 = x_1 * w_1 \tag{1}$$

$$x_2 = x_2 * w_2 \tag{2}$$

The output of the activation functions is the output of the perceptron. The larger the numerical value of the output will be, the higher the confidence of the prediction. [16–19] Let us simplify. The inputs are defined as x_1 and x_2 . Each input x_1 , x_2 gets multiplied by its assigned weight w_1 and w_2 . The input vector represents the training data on which the neural network is getting trained. Once this vector is defined, each element x_i of the vector is assigned to a weight w_i . The input elements are multiplied by their associated weight, and the resulting *n* results are summed to obtain a weighted average. The significance of the impact on individual input vector elements on the weighted average depends on the individual weights. After the weighted average is calculated, a bias is added. The bias is an element that adjusts the boundary away from origin without any dependence on the input value:

$$v = (x_1 * w_1) + (x_2 * w_2) + b \tag{3}$$

Then, perceptron's output will be presented by transforming it to a (non-)linear activation function. We explain later what the activation functions are crucial. [58]

$$y = f(x_1 * w_1 + x_2 * w_2 + b) \tag{4}$$

The SLP was, in a way, the beginning of artificial intelligence and mostly inspired the development of more complex neural networks. The main limitation of the SLP models is that those perceptron models are only accurate when working with data that is linearly separable and is therefore limited. [18]

3.3 Multi layer perceptron

MLPs are distinguished from SLPs because they are hidden layers that affect the output of the model. In the MLP, the neurons are structured into layers. They belong to the class of feed-forward neural networks (FNNs), which means that nothing ever flows back during output generation, i.e. everything flows from input to output. In Figure 6, we can see an artificial neural network with multi-layer perceptrons. The multi-layer perceptrons contain inputs x_n , hidden layers $h_n^{(n)}$, and output layers y_n .

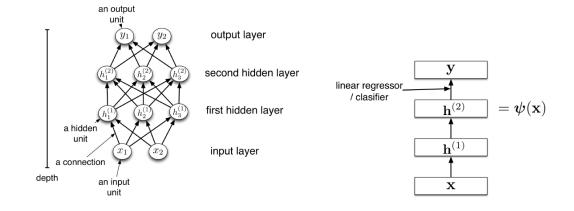


Figure 6: A multilayer perceptron with two hidden layers

Each neuron in the hidden layer $h_n^{(n)}$ is connected to each other neuron in $u_{n+1}^{(n+1)}$. The hidden layers consist of many neurons with connections between the layers. Each input value is associated with a weight connected to each in the hidden layers. The number of layers is known as the depth, and the number of units in a layer is known as the width. The last layer of neurons holds the individual classification or regression output. [6] The general neuron processing unit looks like this:

$$a = \phi(\sum_{j} w_{j} * x_{j} + b)$$
(5)

Where x_j are the inputs to the unit, the w_j are the weights, b is the bias, a is the unit's activation, and ϕ the non-linear activation function. The possibilities of machine learning are immense. Here are just some examples:

- Regression uses a linear model, so $\phi(z) = z$.
- In binary linear classifiers, ϕ is a hard threshold at zero.
- In logistic regression, ϕ is the logistic function $\phi(z) = 1 = (1 + e + z)^{-z}$

A neural network is a combination of several of these presented units. Each one takes on an effortless action. In aggregation, it is capable of powerful and useful computations. As mentioned before, every unit in one layer is connected to every unit in the next layer. Each unit has its own bias and weight for every unit's connections in the two consecutive layers. We can define the first hidden layers $h_i^{(n)}$ as follows:

$$h_{i}^{(1)} = \phi^{(1)} \left(\sum_{j} w_{ij}^{(1)} x_{j} + b_{i}^{(1)} \right)$$
(6)

$$h_i^{(2)} = \phi^{(2)} \left(\sum_j w_{ij}^{(2)} h_j^2 + b_i^{(1)} \right)$$
(7)

$$h_{i}^{(3)} = \phi^{(3)} \left(\sum_{j} w_{ij}^{(3)} h_{j}^{2} + b_{i}^{(3)} \right)$$
(8)

Consider distinguishing $\phi(1)$ and $\phi(2)$ because the different layers may have other activation functions.

Each layer contains multiple units, which represent the activations of all its units with an activation vector. There is a weight for every pair of units in two consecutive layers. We represent each layer's weight for every pair of units in two consecutive layers with a weight matrix. So the above computations could now be written in the desired vectorized form:

$$h^{(1)} = \phi^{(1)} \left(\mathbf{W}^{(1)} \mathbf{x} + \mathbf{b}^{(1)} \right)$$
(9)

$$h^{(2)} = \phi^{(2)} \left(\mathbf{W}^{(2)} \mathbf{h}^{(1)} + \mathbf{b}^{(2)} \right)$$
(10)

$$y_i = \phi^{(3)} \left(\mathbf{W}^{(3)} \mathbf{h}^{(2)} + \mathbf{b}^{(3)} \right)$$
(11)

By applying the activation function to a vector, it independently happens to all the entries. In the feed-forward neural networks, the units are arranged into a graph without cycles, so all the computation can be done sequentially. However, in a recurrent neural network, the graph contains cycles, so the processing can feed into itself and is therefore suitable for time series. [7]

3.4 Backpropagation

Backpropagation is an algorithm of supervised learning of artificial neural networks using gradient descent. With an error function, the method calculates the gradient of the error function concerning the neural network's weights. The calculation runs backward through the whole network, with the gradient of the final layer of weights being calculated first and then the gradient of the first layer of weights. Partial computations of the gradient from one layer are reused to compute the gradient for the previous layer. [47]

3.5 Activation functions

An activation function is a unit that determines which information should be transmitted to the next neuron. Each neuron in the neural network accepts the output value of the neurons from previous layers as input and passes the processed value to the next layer. In a multi-layer neural network, there is a function between these two layers. This function is called the activation function. [Fig. 7] There are several different activation functions, which can output different outputs within a range.

Using a linear function, the input of each layer will be a linear function of the output to the previous layer. The network would be limited in learning linear functions, thus modeling them. No matter how many the neural network has, the output would always be a linear combination. They determine the accuracy of a deep learning model and also the computational efficiency of training a model. Many activation functions are employed, of which ReLu, sigmoid and tanh are extensively used. [8] The tanh function outputs a natural number between [-1,1]. Because the mean value of the output is 0, it achieves a normalization to make the following layers easier to learn. The sigmoid function is a non-linear activation function used for binary classification problems by transforming the output between [0, 1]. By x approaching θ , the gradient becomes steeper. In ReLu, when x is less than 0, its function value is 0. When x is greater than or equal to 0, its function value is x itself. When x is less than 0, the gradient of ReLU is 0, which means the back-propagated error will be multiplied by 0. [8]

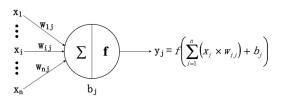


Figure 7: Structure of an activation function

3.6 Optimizers - RMSprop

Optimizers are algorithms that change the attributes of weights and learning rates to minimize their losses. [23] The way of changing the weights or the learning rate is defined by the optimizer you are using. A good example helps to imagine how an optimizer is working. A mountain climber wants to climb down from a mountain but cannot see anything. She does not know where to walk, but she begins to feel her way. By taking a step, she thinks whether it is going down or not. With every step upwards, she makes a loss of progress. With every step downwards, she makes progress, because she wants to reach the bottom. If she does this over some time and only carries out the downhill steps, she will eventually get her destination at the bottom. This process is also done by optimizing a loss function. Optimization algorithms are reducing the losses and provide the most accurate results possible.

RMSprop

RMSprop divides the learning rate by an exponentially decaying average of squared gradients. It decides how much of the gradient you update. More immense steps mean that the weights are changed more every iteration to reach their optimal value faster but possibly miss the exact optimum. So with smaller steps, the weights are changed less every iteration, so it takes more epochs to reach their optimal value, but they are less likely to miss the optimum loss function's optimum. In Figure 8, we can see that if the red dot rolls carefully with a small learning rate, we can expect consistent progress. [24]

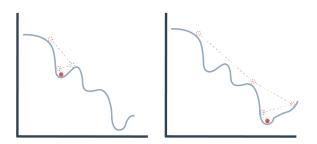


Figure 8: Learning rate

There is an option of learning rates which allows, to use of bigger steps in the first epochs and then reduce the step size when the weights have come in proximity of their optimal value. Several optimizers were introduced in the last few years, where every optimizer has its advantages and disadvantages. The players' weights are updated according to a stochastic gradient descent step with either ADAM or RMSprop optimizer to increase training convergence. It is usually used as a learning criterion of the optimization problem. [24] Remember the RMSprop optimizer, which we will later use in the Implementation. ADAM is one of the most used optimizers, but it did not give better results, which is why we do not go further into the ADAM optimizer.

3.7 Loss functions

Deep learning neural networks use a stochastic gradient descent optimization algorithm. The error of the actual state must be estimated repeatedly. That case requires a loss function to estimate the loss to update the weights, and reduce the loss on the following evaluation. The loss function takes a batch of actual samples and generated samples and then calculates the difference. The lower our loss is, the better the performance of our model. They are explained later in more detail. [27]

3.8 Learning rate

To produce stabilized GAN models, the learning rates should be low. The movement is across the gradient slope by taking smaller values, so the local minimum does not get missed. But higher learning rates might cause the gradient descent to overshoot the minimum, and as a result, the resulting model fails to converge to a minimum and leads to training failure of GANs. [55]

3.9 Batch size

The batch size determines the number of instances passed through the model, before the backpropagation for each epoch happens. A Smaller batch size results in updating the error gradients based on a smaller batch of samples and offers a regularization effect to reduce the error. Sometimes, increasing the minibatch size can improve the performance of the model. However, bigger batch size is expected to impact the performance because of the training of the Discriminator. By using a lot of samples, will end in overpowering the Generator. [55]

3.10 Long Short-Term Memory

The recurrent neural network suffers from gradient vanishing and exploding problems. But this is where the Long Short-Term Memory (LSTM) comes in. The LSTM networks are a modified version of recurrent neural networks (RNN making it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process, and predict time series. It trains the model by using backpropagation. In a LSTM network, three gates are present, as in Figure 9:

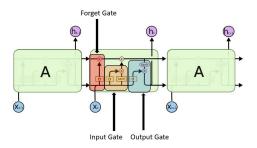


Figure 9: Structure of a LSTM

Input

The Input gate discovers which value from input should be used to modify the memory. The sigmoid function then decides which values get through [0,1]. The tanh function gives weightage to the values which are passed, deciding their level of importance ranging from [-1, 1].

$$i_{\rm t} = \sigma(W_{\rm i} * [h_{\rm t-1}, x_{\rm t}] + b_{\rm i}$$
 (12)

$$\tilde{C}_{t} = tanh(W_{c} * [h_{t-1}, x_{t}] + b_{c})$$

$$\tag{13}$$

Forget gate

The forget gate shows what details are discarded from the block. It is decided by the activation function. It considers the previous state $h_{(t-1)}$ and the content input (X_t) and outputs a number between [0,1] for each number in the cell state $C_{(t-1)}$.

$$f_{t} = \sigma(W_{f} * [h_{t-1}, x_{t}] + b_{f})$$
(14)

Output gate

The input and the memory of the block is used to decide the output. Activation functions decide which values to let through [0, 1], and then the tanh function gives weightage age to all values, which then decides its value from [-1, 1] and multiplies with the output of the activation function. [49]

$$o_t = \sigma(W_o[h_{t-1}, x_t] + B_o) \tag{15}$$

4 Generative Adversarial Network (GAN)

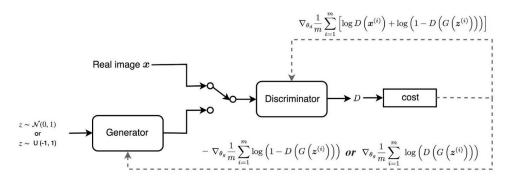


Figure 10: Generative Adversarial Network (GAN)[46]

Generative adversarial networks take up a game-theoretic approach. The network learns to generate from a training distribution. The Generator G(z) takes random noise vector z as input with corresponding network weights. The Discriminator D, on the other side, is a classifier with a containing loss function. The Discriminator has to determine if the sample is real data x or generated data G(z). Depending on whether the Discriminator has evaluated the input correctly or wrong, it gets an update on the Generator's parameters. The Generator and the Discriminator can perform almost the same tasks. Nonetheless, there is a difference in the math behind. The generative model describes how data is generated to learn any kind of data distribution using supervised learning. The Discriminator, however, models the decision boundary between the classes. Then it outputs the probability that x comes from the data rather than from the probability distribution p_g . So both models predict the conditional probability but from different probabilities. [27] Each function is differential concerning its inputs and with respect to its parameters. [26] The overall value function V(D, G) for the connection between G and D is defined as:

$$E_{\rm x}[\log(D(x))] + E_{\rm z}[\log(1 - D(G(z)))]$$
(16)

D(x) is the probability, given by the Discriminator, that real data instance x is real. E_x is the expected value over the real data instances. G(z) is the output of the Generator from a given noise z. The input z to the Generator is sampled from some simple noise, such as the uniform distribution or a spherical Gaussian distribution. They should be chosen for the following reasons [28]:

- 1. The unit variance means that each element in the noise vector can be a different feature of the output image.
- 2. With the Gaussian distribution, it will be easy to draw samples and even interpolate between two values and see a progressive change in the image.

D(G(z)) is the Discriminator's estimate of the probability that a fake instance is real. E_z is the expected value over all random inputs to the Generator. In other words, it is the expected value over all generated fake instances G(z). The Discriminator is trained to maximize the probability of choosing the real label. The Generator, however, is trained to minimize $\log(1 - D(G(z)))$. The Generator G cannot directly affect $\log(D(x))$ in the function, but it can minimize the loss, which is equivalent to reduce $\log(1 - D(G(z)))$. During training, G_z is performing not so well and is just generating some random noise, so D(x) can reject samples from G(z) with great confidence. Both optimize their weights according to their opposing loss function objective. The network improves each other until the Generator generates data that are indistinguishable from real data. [27]

4.1 Discriminator loss

When the Discriminator is trained, it decides both the real data and the fake data from the Generator. It penalizes itself for declassifying a real instance as fake, or a fake instance as real. It punishes itself if a real instance is evaluated as as fake, or a fake, created by the Generator, is evaluated as real. It can achieve it by maximizing the below function. [27]

$$\nabla \theta_{\rm d} 1/m \sum \left[\log D(x^{(i)}) + \log(1 - D(G(z^{(i)}))) \right] \tag{17}$$

 $\log(D(x))$ refers to the probability that the Generator is classifying the real image, maximizing $\log(1 - D(G(z)))$ may help to label the fake image correctly.

4.2 Generator loss

The Generator takes random noise and then produces an output from that noise. The Generator loss is calculated from the Discriminator's classification and if it successfully fools the Discriminator, it gets rewarded, otherwise, it gets penalized. [52] The following equation gets minimized to training the Generator:

$$\nabla \theta_{g}^{1/m} \sum \left[\log(1 - D(G(z^{(i)}))) \right]$$
(18)

4.3 Challenges in the training process

In the first step, the Discriminator is solid. So no adjustments to the parameters are made for the Discriminator network at this moment. The Generator get trained for a certain number of so-called training steps. The Generator is trained by using backpropagation. The training progress depends on the current state of the Discriminator, because various problems can occur in training a GAN. One of them is that the feedback of the Discriminator gets worse, as the Generator is performing better during the training. The training has to be finished in time, otherwise, the quality of the Generator and the Discriminator may decrease due to the random return values of the Discriminator. [10]

4.3.1 Training of the Discriminator

The Generator model's weights and biases are constant when creating samples for the Discriminator to get trained. The Discriminator is related to two loss functions, the Discriminator loss and Generator loss. During training of Discriminator, the Discriminator uses the Discriminator loss but ignores the Generator's loss. It attempts to classify the original data as 1 and 0 for a fake. Otherwise, it predicts a probability value. Based on the Discriminator loss, the Discriminator get penalized for misclassification. In a manner of backpropagation, the Discriminator's weights are updated using the Discriminator loss. [55]

4.3.2 Training of the Generator

Based on the Discriminator output, the Generator loss is calculated to obtain gradients and penalize the Generator for failing in fooling the Discriminator. The Generator's objective is to generate fake samples that the Discriminator classifies as original and predicts a probability of 0.5 as it cannot differentiate between fake or original samples. During Generator training, the Discriminator training is stopped, and its weights remain fixed to avoid the Discriminator model from becoming too strong. If the Discriminator training is kept on during Generator training, the Generator will never converge or be able to learn the distributions. This is because of the Discriminator, which will continue to improve and then would become too good in recognizing the fake samples. [55]

4.3.3 Mode Collapse

Another common problem is the "mode collapse". It may occurs that the Generator collapse to a setting where it always will produce the same outputs. It is a familiar failure case for a GAN. Even though the Generator might fool the Discriminator, it fails to present the complex real-world data distribution. It therefore gets stuck in a small space with a low variety. The classification performance of the Discriminator does not get affected by the low variety in the generated samples because the low variety is in the real data. The Discriminator, therefore, has no control getting the Generator to increase its variety. [10]

4.3.4 Vanishing problem

The Vanishing gradient problem is a difficulty in training artificial neural networks with gradient-based learning methods and backpropagation. In such scenarios, each of the neural network's weights receives an update proportional to the partial derivative of the error function concerning the current weight in each iteration of training. In some cases, the gradient will be little by preventing the weight from changing its value. This may stop the neural network from training and gets stuck. [50]

4.4 Implementation of a GAN

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, c = 0.01, m = 64, $n_{\text{critic}} = 5$.

Require: : α , the learning rate. c, the clipping parameter. m, the batch size. n_{critic} , the number of iterations of the critic per generator iteration.

Require: : w_0 , initial critic parameters. θ_0 , initial generator's parameters.

1: while θ has not converged do for $t = 0, ..., n_{\text{critic}} \mathbf{do}$ 2: Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data. Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples. $g_w \leftarrow \nabla_w \left[\frac{1}{m} \sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$ $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$ 3: 4: 5: 6: $w \leftarrow \operatorname{clip}(w, -c, c)$ 7:end for 8: Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples. $g_{\theta} \leftarrow -\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m f_w(g_{\theta}(z^{(i)}))$ 9: 10: $\theta \leftarrow \theta - \alpha \cdot \overline{\text{RMSProp}}(\theta, q_{\theta})$ 11: 12: end while

Figure 11: Implementation of GAN

4.5 The Wasserstein GAN

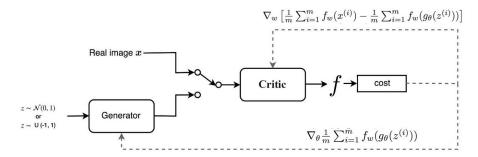


Figure 12: Model of a WGAN

The main differences between a standard GAN and a WGAN:

- A WGAN uses the Wasserstein loss.
- The WGAN is trained using labels of 1 for real and -1 for fake.
- There is no need for a sigmoid activation in the final layer in the WGAN critic.
- Train the critic multiple times for each update of the Generator.

The Wasserstein GAN was one of the first significant steps toward stabilizing GAN training. With some changes, the authors were able to show how to train GANs that have the following two properties (quoted from the paper):

- A meaningful loss metric that correlates with the Generator's convergence and sample quality
- Improved stability of the optimization process

Specifically, the paper introduces a new loss function for both the Discriminator and the Generator. It is using the Wasserstein loss function instead of binary cross-entropy from the original GAN results in a more stable convergence of the model. It is defined as the shortest average distance to move the probability mass from one distribution to another. Two distributions which are in lower-dimensional manifolds without overlaps, the Wasserstein distance can provide a smooth representation of the space in-between, which is not always the case with the JS-Divergence.

Discriminator/Critic

Generator

$$\mathbf{GAN} \qquad \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right] \qquad \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log\left(D\left(G\left(z^{(i)}\right)\right)\right) \tag{19, 20}$$

WGAN
$$\nabla_{w} \frac{1}{m} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - f\left(G\left(z^{(i)}\right)\right) \right] \qquad \nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} f\left(G\left(z^{(i)}\right)\right) \qquad (21, 22)$$

By removing the sigmoid activation from the final layer of the Discriminator, the predictions no longer fall in the range [0,1], but instead, it can now be any number in the range $[-\infty, \infty]$. For this reason, the Discriminator in a WGAN is usually referred to as a critic. [28] The WGAN value function is constructed using the Kantorovich-Rubstein duality to obtain:

$$\min_{G} \max_{D \in D} \mathbb{E}_{x \sim \mathbb{P}_{\mathbf{r}}}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_{\mathbf{g}}}[D(\tilde{x})]$$
(23)

where D are 1-Lipschitz functions and \mathbb{P}_g the model distribution. Under an optimal Discriminator, minimizing the value function with respect to the Generator's parameters will minimize $W(\mathbb{P}_r, \mathbb{P}_g)$. [28] The Wasserstein loss function trains the critic to convergence to ensure that the gradients for the Generator update are exact. Whereas in a standard GAN it is important to not let the Discriminator get too strong to avoid vanishing gradients. With WGANs, we can simply train the critic several times between Generator updates to make sure it is close to convergence. [56]

4.6 Lipschitz Constraint

It is unusual for allowing the critic to output any number in the range $[-\infty, \infty]$, rather than applying e.g., a sigmoid function to restrict the output to the usual [0, 1] range. The Wasserstein loss can, for this reason, be tremendous, which is usually unsettling, due to that large numbers in neural networks should be avoided. The authors of the WGAN paper show that for the Wasserstein loss function, it also needs to place an additional constraint on the critic. Specifically, it is required that the critic is a 1-Lipschitz continuous function. To enforce the Lipschitz constraint on the critic, it proposes to clip the critic's weights to be in a compact space. Functions that are continuously differentiable on every point are Lipschitz continuous because of its derivation. [46] The critic is a function *D* that converts an input into a prediction. A function is 1-Lipschitz if it satisfies the following inequality for any two inputs, x_1 and x_2 :

$$\frac{D(x_1) - (x_2)}{|x_1 - x_2|} \le 1 \tag{24}$$

 x_1 and x_2 are the average absolute difference between the two inputs and the critic predictions. It requires a limit on the rate at which the predictions of the critic can change. The absolute value of the gradient must be at most one everywhere. [56]

4.7 Wasserstein GAN with gradient penalty term

A main criticism of the WGAN is that the capacity in weight clipping of the critic is strongly diminished to learn. Even in the original WGAN paper the authors write: "Weight clipping is a bad way to enforce a Lipschitz constraint." A good performance of a critic is crucial of a WGAN, because, without accurate gradients, the Generator cannot learn how to update its weights to create better samples over time. Therefore, one of the most recent extensions to the WGAN is the Wasserstein GAN Gradient Penalty (WGAN-GP) framework. The WGAN-GP Generator is defined and compiled in the same way as the Wasserstein GAN Generator. It is only the critic that needs to be changed by including the penalty gradient term.

Gradient penalty

A differential function f is 1-Lipschitz if it has gradients with the norm at most 1 everywhere. So, points interpolated between the real and generated data should have a gradient norm 1 for f. Instead of applying weight clipping, the gradient penalty penalizes the model if the gradient norm moves away from its target norm value 1.

 $L = \underbrace{\mathbb{E}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_g}[D(\hat{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r}[D(\boldsymbol{x})]}_{\text{Original critic loss}} + \underbrace{\lambda_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}_{\hat{\boldsymbol{x}} \sim \mathbb{P}_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}$

Figure 13: WGAN with gradient penalty [46]

 \tilde{x} is sampled from \tilde{x} and x is sampled t-uniformly [0,1]. Lambda (λ) is usually set to 10. Batch normalization is avoided for the critic. Batch normalization creates correlations between samples in the same batch. By penalizing the norm of the gradient of the Discriminator with respect to its inputs instead of the existing weight clipping, it can improve classification accuracy. [28–30]

Implementation of Wasserstein with gradient penalty

Algorithm 1 WGAN with gradient penalty. We use default values of $\lambda = 10$, $n_{\text{critic}} = 5$, $\alpha = 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$.

Require: The gradient penalty coefficient λ , the number of critic iterations per generator iteration n_{critic} , the batch size *m*, Adam hyperparameters α , β_1 , β_2 .

Require: initial critic parameters w_0 , initial generator parameters θ_0 . 1: while θ has not converged do for $t = 1, ..., n_{\text{critic}}$ do 2: for i = 1, ..., m do 3: Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$. 4: 5: $\tilde{\boldsymbol{x}} \leftarrow G_{\theta}(\boldsymbol{z})$ $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon) \tilde{\boldsymbol{x}}$ $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda (\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$ 6: 7: 8: end for $w \leftarrow \operatorname{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^m L^{(i)}, w, \alpha, \beta_1, \beta_2)$ 9: end for 10: Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \sim p(\boldsymbol{z})$. 11: $\theta \leftarrow \operatorname{Adam}(\nabla_{\theta} \frac{1}{m} \sum_{i=1}^{m} -D_{w}(G_{\theta}(\boldsymbol{z})), \theta, \alpha, \beta_{1}, \beta_{2})$ 12:

```
13: end while
```

Figure 14: Implementation of WGAN-GP

5 Theoretical evaluation methods

5.1 QQ-Plot

The quantile-quantile (qq) plot is a graphical technique for determining if two data sets come from a common distribution. A qq-plot is a plot that sets the quantiles of the first data set against the quantiles of the second data set. By a quantile, it means the percent of points below a given value. If a reference line is plotted and the two groups come from a population with the same distribution, the points should fall close along this reference line. The greater the distance to this reference line, the higher the probability that the two given data sets come from populations with different distributions. There are several advantages of qq-Plot, such as:

- The sample sizes do not have to be the same.
- Many distributional aspects can be simultaneously tested. As example the shifts in location and scale, changes in symmetry, and the presence of outlier can be detected in a qq-Plot.

whereas questions can be answered as follows:

- Do the two distributions from populations with a common distribution?
- Are location and scale of the two datasets familiar?
- Are the shapes of the two datasets similar?
- Are there similarities in the tails?

When there are two data samples, the assumption of a similar distribution is justified. If so, the location and scale estimators can pool both data sets to obtain estimates of the common location and scale. When two samples are different, it can also be helpful to understand the differences. [34]

5.2 ACF

The random errors in the model are often positively correlated over time. Each random error is more likely to be similar to the previous random error that it would be, if random errors were independent of each other. The coefficient of correlation between two values in a time series is called the autocorrelation function (ACF). An autocorrelation of +1 represents a perfect positive correlation, which means that an increase in one time series leads to a proportionate increase in the other time series. On the other hand, an autocorrelation of -1 represents a perfect negative correlation, so an increase in one time series results in a proportionate decrease in the different time series. Even if the autocorrelation is in minus, there can still be a non-linear relationship between a time series and a lagged version of itself.

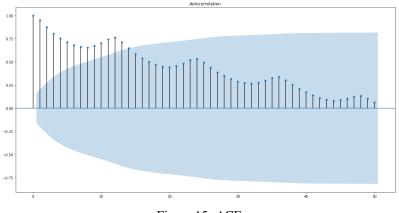


Figure 15: ACF

6 Descriptive Analysis

6.1 Introduction for data and Bitcoin

6.1.1 Bitcoin

In 2009 the Bitcoin was launched on the market and became the first publicly traded cryptocurrency. Cryptocurrencies are secure digital currencies that use encryption techniques called cryptography. Bitcoin is by far the best-known and most traded cryptocurrency with the largest market capitalization. Bitcoins are created through digital mining, in which computational puzzles are solved. The developers limited the maximum number of Bitcoins on the market to 21 million. More than 18 million of the coins have already been mined. The critical difference referring to the high risk and volatility of Bitcoin compared to traditional currencies is that Bitcoin is not supported by a central bank or backed by a government. Therefore, Bitcoin is decentralized and unregulated. Thus, monetary policy, inflation rates, and economic growth measurements, which generally impact the value of a traditional currency, do not apply to Bitcoin. In their paper, Osterrieder and Lorenz (2016) analyzed the Bitcoin exchange rates, compared them to the G10 currencies, and showed that the Bitcoin return distribution exhibits higher volatility, more substantial non-normal characteristic, and heavier tails than the G10 Currencies.

6.1.2 Data

The data used for the Descriptive Analysis and training of the Generative Adversarial Network is obtained from yahoo finance. Although the purchase of Bitcoins is publicly available since 2009, the first time a U.S regulatory agency approved a Bitcoin financial product was in September 2014 [3]. Therefore, for this Thesis, we will use the Bitcoin publicly available data from 17.09.2014–02.05.2021 from YahooFinance.com. Further, to accomplish our study, we will run a descriptive analysis on the Bitcoin and S&P500 to show how unique the financial time series of Bitcoin (or Cryptocurrencies in general) are. The S&P500 is a stock index that includes the 500 largest publicly traded U.S. companies. The index is weighted by market value capitalization and is one of the world's most widely followed stock indexes. We base our analysis on the daily close prices of the assets. Note that Bitcoin trades continuously, while the S&P500 trades on average 252 days per year. Thus, the sample size differs substantially, with 2416 data points for Bitcoin and 1666 data points for the S&P500. This difference in trading days leads to an inevitable distortion in the following calculations. Financial time series data are susceptible to random fluctuations. They are non-linear, non-stationary, and behave chaotically. Nevertheless, some observations and empirical findings are shared across various instruments, markets, and periods referred to as stylized facts. We show in Figure 16 the daily close prices of Bitcoin. We see that the asset price growths generally differ. Compared to the S&P500, Bitcoin seems to have an extremely volatile, unpredictable, and explosive growth.

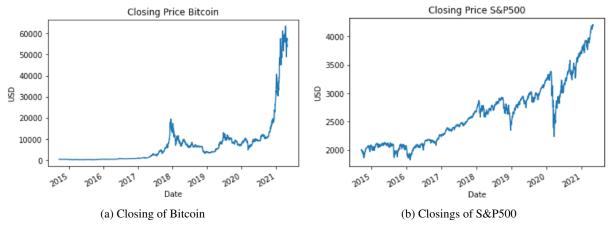


Figure 16: Closing prices of Bitcoin and S&P500

The growth dynamics of the two assets generally differ. Compared to the S&P500, Bitcoin seems to have an extremely volatile and explosive growth.

6.2 Log returns

Most methods of financial time series analysis operate with static data. A stationary time series is represented by data over time, whose statistical properties remain constant regardless of a change in the time origin. There are various ways of making time series stationary, but those ways look at the difference between values rather than the absolute values. In market data, the standard way to get stationary data is to work with log returns. They are calculated as the natural logarithm of the index today divided by the index of the day before: $\ln(\frac{V}{t-1})$. [40] Figure 17 shows the log returns as a time series of the Bitcoin and S&P500. We observe that the Bitcoin returns show more extreme behavior compared to the S&P500. The extreme movements are not surprising due to the high volatility of Cryptocurrencies.

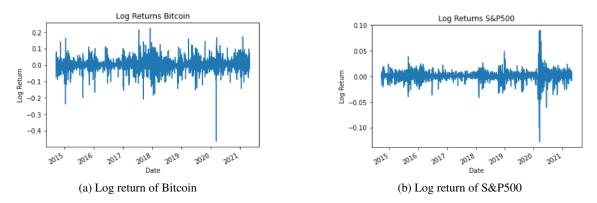


Figure 17: Log returns of Bitcoind and S&P500

The following tables (Figure 18 (a) & (b)) provide a few of the essential key measures of the investigated log returns. As you can see in the table, the log returns minimum, maximum, and quantile values of Bitcoin are much more extreme than the S&P500 values. The volatility of a financial time series is often referred to as the standard deviation (std). Comparing the standard deviations in the table shows that the Bitcoin behaved (in the observed period) over three times more volatile than the S&P500. It is interesting to note that traditional currencies typically exhibit lower volatility than stock indexes. [39][43]

Bit	tcoin_log_return
int	2416.000000
mean	0.001999
std	0.039168
min	-0.464730
25%	-0.012223
50%	0.002009
5%	0.017861
ax	0.225119
(a) Bitcoin log returns	

Figure 18: Log return tables

6.3 Fat tails

A common stylized fact of financial time series is the exhibition of fat tails. The distribution of a variable has fat tails if its outcomes are more extreme than a normally distributed variable, where the mean and variance are equal. The qq-plot is a common graphical method that is used to analyze the tails of a distribution. Figure 19 show the qq-plots, comparing the distribution of log returns with the expected normal distribution. We see that Bitcoin and S&P500 returns generally do not correspond to a normal distribution.

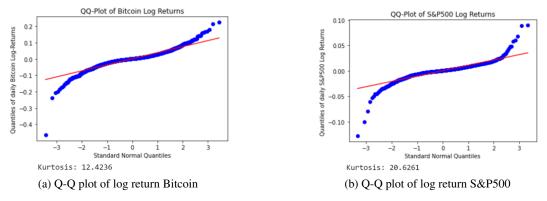
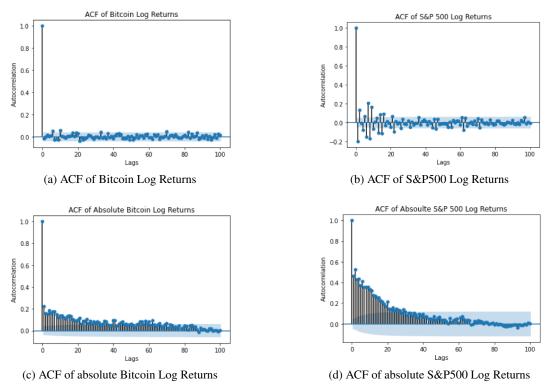
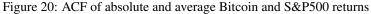


Figure 19: QQ-Plots of Log return of Bitcoin and S&P500

6.4 Volatility Clusters

A common stylized fact is volatility clustering. It means that high price volatility events tend to cluster with time. In addition, it generally means that significant price changes follow large changes in price levels, and small changes tend to be followed by small changes. Autocorrelation plots are realistic representations of volatility clustering effects. We use the (linear) autocorrelation function (ACF) to measure how the current price values and its past values correlate. In Figures 20 (a) and (b), we attest that no such linear autocorrelations exist neither among Bitcoin nor across the S&P500 data sets. This result confirms generally accepted knowledge that the price movements in financial markets are not exhibiting any significant linear correlations. If such correlation were statistically significant, the prediction of stock prices would overall deliver highly accurate results because there would be strong evidence for predictability. In Figures 20 (c) and (d), the dependency of the conditional variance can be captured using absolute log returns. Moreover it shows, that absolute log returns present significant correlations with a slow decaying trend. Therefore, we conclude that log returns are not independent, principally caused by volatility clustering. [41]





6.5 Skewness and kurtosis

Fundamental statistic and probability theory almost exclusively deal with the first and second central moment of a random variable, expectation and variance. For the statistical analysis of financial data, the third and fourth central moments called skewness and kurtosis are often meaningful.

The skewness is the third central moment and is essential for measuring another common stylized fact called the "Gain-Loss Asymmetry". It is referred to as the observation when significant drawdowns in a stock index value appear, but not equally large upward movements. The skewness measures the symmetrical gathers around its mean, and therefore are all symmetrically characterized with a skewness value of 0. Positive skewness implies that the right tail is fatter than the left, which means that positive returns tend to occur more often than significant negative returns. The kurtosis is the fourth central moment and measures the degree of peakedness of the distribution relative to its tails. The kurtosis of the normal distribution is given by the value 3. This is the reason, why a higher kurtosis value than 3 is a signal for fat tails.

The minimal kurtosis value of a distribution is equal to 1. The probability density function of the log-returns of both investigated distributions is shown in Figure 21 with their associated skewness and kurtosis values. At first glance, we can see that the log-returns of Bitcoin are more spread and the S&P500 and has a higher peak. The negative skewness values imply that both distributions have fatter left tails than right tails, whereas the S&P500 seems slightly more left-skewed. Additionally, we attest that both datasets have a higher kurtosis value than the normal distribution. Our analysis shows that the value of the S&P500 has a higher kurtosis value than Bitcoin.

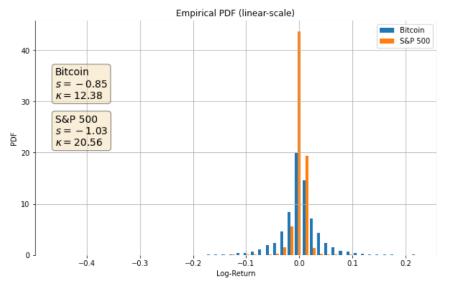


Figure 21: Empirical evaluation of Bitcoin and S&P500

7 Our approach to Wasserstein GAN with gradient penalty and LSTM

Our Implementation of the WGAN-GP written with TensorFlow 2 (Version 2.4.1 and Keras Version 2.4.3) is inspired by the Keras-Team who have created a Wasserstein GAN on images as described in paper [28]. We simply had to adjust/lower the input dimensions of tensorflow/numpy vectors and Generator and Discriminator architecture to use it on time series.

7.1 Data processing and Implementation

Note that the gradient w.r.t of the input averaged samples, not the Discriminator weights, that we are penalizing. To evaluate the gradients, we must first run samples through the Generator and evaluate the loss. Then we get the gradients of the Discriminator w.r.t. the input averaged samples. The 12 therefore calculates the norm and penalty. This loss function requires the original averaged samples as input, but Keras only supports passing y_{true} and y_{pred} to loss functions.

The keras-team made a partial of the function with the averaged samples argument, and used that for model training. The Keras API merge layer function to calculate the weighted average for the interpolation did not work on Tensorflow Versions above 1.15. So we had to create a custom Merge layer using the class tensorflow.keras.layers.Layer that can perform a weighted sum/merge of two different layers' outputs to get the weighted average. As for the training we made three different labels of vectors. Valid is the vector label for real samples with value 1, the fake is the label vector for generated samples, with value -1 and a dummy sample (not used) which is passed to the gradient penalty loss function. In particular, the Implementation has two parts. One Discriminator and one Generator. The implemented training is in two steps. The Discriminator will be trained. During the training of the Discriminator Generator layers are frozen. The Discriminator takes both real image samples and random noise seeds as input. The noise seed is run through the Generator model to get generated images.

Both real time series (processed in the pipeline as shown in Figure 22) and the generated images are then run through the Discriminator. After training of the Discriminator the Generator model is used to train the Generator layers. As such, the Implementation ensures that the Discriminator layers are not trainable. The whole training is organized in two loops. In the WGAN model, the critic model must be updated more than the Generator model. A new hyperparameter, called n_critic, is defined to control the number of times that the critic is updated for each update to the Generator model, and is set to 5. In Figure 22, we show out data processing pipeline.

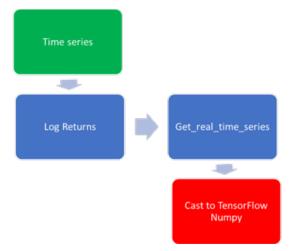


Figure 22: Data-processing

7.2 Parameter settings

Parameter	Batch size:	32
	Learning rate:	0.00005
	n_Critic:	5
	ts_length:	50
	Epochs:	3000

Figure 23: Parameter settings

In Figure 23, we demonstrate all the parameter settings that been used in our model. The WGAN model was trained over a runtime of 3000 epochs. The number of critic iterations per Generator iteration was set to 5, and to optimize the model we used a RMSprop optimizer with a learning rate of 0.00005. The random noise vector from which the Generator produces data has a length of 25 and is generated from a normal distribution. The generated time series has a size of 50, while the real and fake batches, which are fed to the Discriminator as inputs, consist of 32-time series each. The architectures of the Critic and Generator are structured as shown in Figure 24. The Generator has one LSTM block, and the output is reshaped to match the dimension of a real sample. The Discriminator has one hidden layer consisting LSTM and an output layer of dimension one. The output of the Discriminator is just without the sigmoid function and outputs a scalar score rather than a probability. This score can be interpreted as how real the input images are.

Layer (type)	Output	Shape	Param #		
lstm_4 (LSTM)	(None,	50)	10400		
reshape_2 (Reshape) Total params: 10,400 Trainable params: 10,400	(None,	50, 1)	0		
(a) Generator architecture					

Layer (type)	Output Shape	Param #			
lstm_5 (LSTM)	(None, 50)	10400			
dense_2 (Dense)	(None, 1)	51			
Total params: 10,451					

Trainable params: 10,451

(b) Critic architecture

Figure 24: Generator and critic architecture

7.3 Architecture

For the Discriminator and also for the Generator we are using vanilla LSTM. A Vanilla LSTM is an LSTM model with a single hidden layer of LSTM units and an output layer used to make a prediction. LSTMs can be used to model univariate time series forecasting problems. These are issues comprised of a single series of observations, and a model is required to learn from the series of past observations to predict the next values in the sequence. LSTM adds the explicit handling of sequence between observations when learning a mapping function from inputs to outputs, not offered by MLPs or CNNs. They are a type of neural network that adds native support for input data comprised of sequences of observations. The shape of the input in each sample is specified by the input shape argument of LSTM layer. We almost always have multiple samples. Therefore, the model will expect the input component of training data to have the dimensions or shape: **[samples, timesteps, features]**

The basic LSTM neural network cell is something like $h_t = f(W_h h_{t-1} + W_x x_t)$. [70] So, it takes previous hidden state h_{t-1} and current input x_t to produce hidden state ht and in each time step, we use the same weights. The specified

number of time steps defines the number of input variables used to predict the next-time step. We are using 50-time steps. For daily data, this is about a two-month time series sequence.

The selection of 50-time steps was inspired by the 50 day "MA" (moving average) in stock trading. The 50-day moving average indicator is one of the most important and commonly used tools in stock trading. The 50-day moving average is one of the leading indicators. It is, therefore, the first line of major moving average support in an uptrend or the first line of major moving average resistance in a downtrend. That is why we use 50-time steps to train our LSTM. [69] Since this is a univariate time series prediction, we have the time series itself. That's why the feature is set to one. It is also possible to consider as additional feature the autocorrelation of the time series itself. The LSTM network in both Generator (G) and Discriminator (D) has depth 1; each LSTM cell has 50 units. The output from each LSTM cell in D are fed into a fully connected layer with weights shared across time steps, and one output per cell is then averaged to the final decision for the sequence.

Using unidirectional LSTM only preserves the information of the past because the only inputs it has seen are from the past. Using bidirectional will run your inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backward you preserve information from the future and using the two hidden states combined, you can preserve information from both, past and future, in any point in time. Since there is no economic rational to use future information in the past and the possibility that the model starts to learn noise from the future, we did not choose bidirectional LSTM. Generator and Discriminator have both the same LSTM architecture since they need to learn the same sequence pattern in the time series.

7.4 Experimental Results

Figure 25 are the generated samples that we collected every 500 epochs after training the model during the 3000 epochs. The results show that the model can generate specific price curves for various elements of the multitude of synthetic financial time series. Figure 25 (f) shows all the generated artificial time series of epoch 3000 (in blue) and the real historical time series of the Bitcoin (in red). The real-time series is shown with an illustrative goal so we can make a comparison between the synthetic output and the actual data. There is not much visual difference in price movements between the artificial and real prices data sets. The synthetic data we obtain seems to follow generally the same trends as the real Bitcoin prices. This result shows that the GAN model we adapted for financial data did work. Therefore it is possible to generate synthetic representative financial data sets.

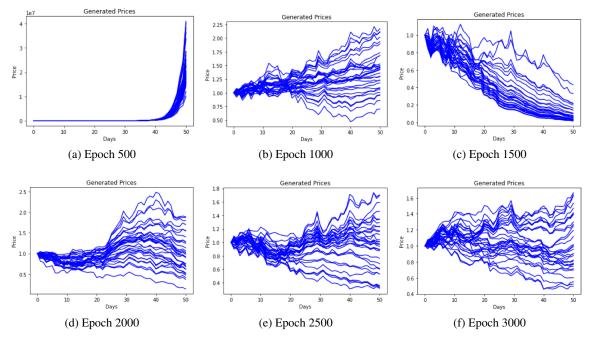


Figure 25: Generated prices all 500 time steps until epoch 3000

7.5 Distribution Comparison

Now we compare the distribution of the historical samples with the generated time series samples. Figure 26 show the probability density of the log returns from the synthetic and real data passed to the Discriminator and visualized every 500 epochs. To make it more accessible to interpret the results, we represented all the log returns of one batch in a histogram (synthetic times series in red and actual Bitcoin prices in blue). Typically, the kurtosis is similar to the variance, and it measures how a distribution is spread with a focus on the tails. The skewness value measures how symmetrical distribution is gathering around its mean. The negative skewness values obtained in Epoch 3000 imply that both financial time series, synthetic and real Bitcoin log returns, are left-skewed. In Epoch 3000, the skewness value of the historical Bitcoin data, in absolute value, is considerably higher than the synthetical times series log return distribution. The result suggests that typically, historical Bitcoin incurs proportional smaller gains and higher losses than the synthetic financial time series. The kurtosis of the historical Bitcoin log returns present more substantial non-normal characteristics than synthetic time series.

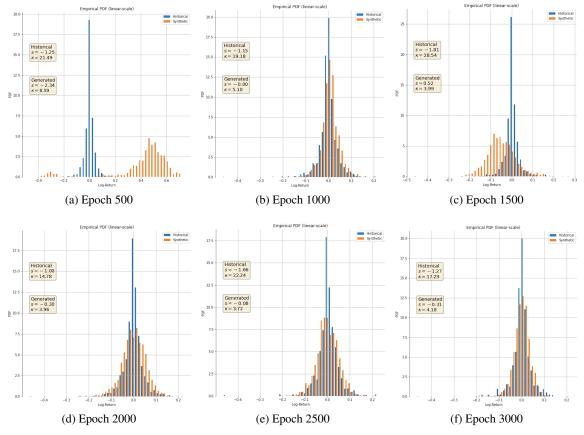


Figure 26: Empirical and historical distributions all 500 time steps until epoch 3000

As mentioned in the descriptive analysis, the historical Bitcoin log returns have fat tails as most of the stock returns. In Figure 27, each of the quantiles of the historical and synthetic log returns of a whole epoch is plotted against the quantiles of the normal distribution. On the one hand, we observe that historical data has fatter tails compared to the tails of the artificial data illustrated in the qq-Plot of epoch 3000. The result suggests that historical Bitcoin returns are more volatile, incurring a more extreme outcome than the synthetic data. Therefore, we can attest that the WGAN we adapted to the financial times series is insufficient to generate reliable, robust data that would generally represent the underlying dataset. On the other hand, the plots show that the left tail of the generated distribution is heavier than the right tail, confirming our result that the distribution of the artificial data is left-skewed. For this reason we could say that the synthetic data is approaching the real Bitcoin log return characteristics with minimal gains and significant losses.

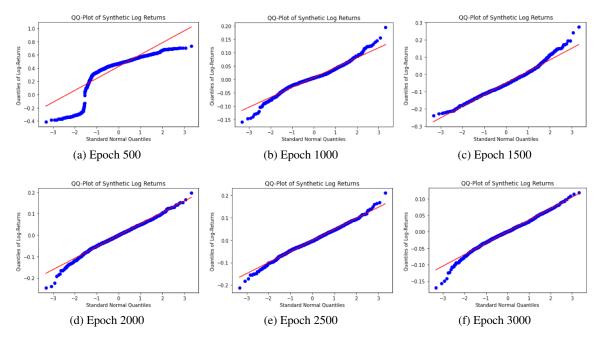


Figure 27: QQ-Plots of Synthetic Log returns all 500 time steps until epoch 3000

A PREPRINT - JULY 14, 2021

7.6 Autocorrelation (ACF)

Comparing the ACF of the log returns, we conclude that the synthetic data differs from the historical data, showing on average a higher degree of autocorrelation (see Fig. 28 (a) & (b)). In Figure 28 (c), the average autocorrelations of the absolute historical log-returns show a positive and slowly decaying trend. Figure 28 (d) shows that the artificial data has a similar tendency as the real dataset but a much more fluctuating behavior. A slow decay in the ACF is an indication that a series is non-stationary because past values heavily influence the values. Since LSTM has a memory, this could explain the distinction between synthetic log-returns and real log returns. The autocorrelation could be controlled by using the length of the time series.

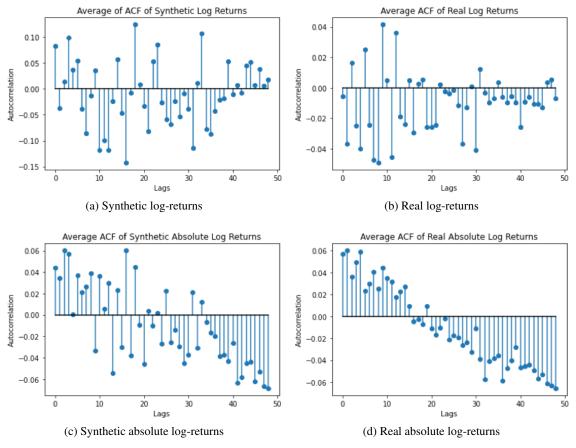


Figure 28: Average and absolute of ACF of synthetic Log returns

7.7 Loss function

In Figure 29, we plot the Loss of the Generator and Discriminator (Critic) from the adapted GAN model to the financial times series. The advantage of applied weight clipping in WGANs is that, in theory, the loss correlates with the sample quality and converges to a minimum [28]. However, these results are only proved for images, and there is no evidence of validity for the financial times series.

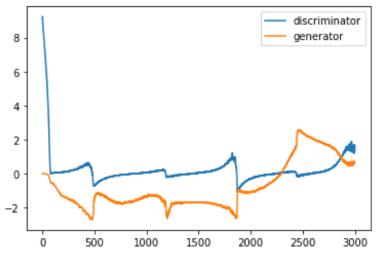


Figure 29: Loss of Generator and Critic

As illustrated in Figure 29, both the Generator loss and Discriminator loss converge and show a stable behavior while oscillating around 0. It is difficult to establish the presence of oscillation. In the absence of concrete proof, we assume the model oscillates around a minimum rather than converging to a minimum (as in theory) because the model can overrate the outliers of real time series. Besides detecting the outliers, the model strives to learn the outliers, making it challenging for the model to converge to an absolute minimum. That may be the reason why it oscillates around the absolute minimum. The loss of both gradually increases or decreases between the batches, but there is no erratic fluctuation, proving that the implemented Wasserstein metric has served its purpose. We think the stable behavior of the loss shows that the adapted WGAN model on financial time series can cope with the historical data set. The research on WGAN-GP models for generating financial time-series and evaluating their results is at an early stage. Elaborating the perfect hyperparameters of the model and setting a sufficient epoch size might lead to a decrease of oscillation or even a convergence to an absolute minimum.

8 Conclusion

This paper covered the structure and outline of a GAN and LSTMs in detail, summarizing the most relevant literature. We have trained our recurrent WGAN-GP model for generating real-valued sequential data. The implementation of the Wasserstein metric kept a stable loss of both the generator and discriminator during the whole training. The loss converged to a minimum with a cycle oscillation. It is difficult to find the cause of the oscillation. LSTM models are at an early stage and the model parameter settings. Finding the optimal parameters to fine-tune the model's performance is challenging and a time-consuming task that provides a basis for further investigation. However, the evaluation has shown that a visual distinction between the generated and real-time series is nearly impossible. In contrast, the statistical properties slightly differ from the generated data by more extreme outcomes. That is why we think it might be challenging for the model to catch up and replicate outliers.

How close a GAN can get to the stock prices and reflect their properties is still hard to demonstrate or proof, due to its high volatility and unexpected happenings on the market. Nevertheless, promising arguments and approaches are speaking in favor of further experiments in this area. GANs seem to be an excellent methodology to capture the dynamics of financial assets and forecast future movements. Applying them to derivatives pricing, Portfolio Hedging, and Risk Management may become a crucial tool in the future since we live in Big data era.

References

- Mirko Savasta "Generating synthetic financial time series with WGANs" A first experiment with Pytorch code in June, 19, 2020, url: https://towardsdatascience.com/generating-synthetic-financial-timeseries-with-wgans-e03596eb7185
- [2] Caspar Hogeboom, "Generation of Synthetic Financial Time-Series with Generative Adversarial Network", June 25, 2020, url: https://new.ingwb.com/binaries/content/assets/insights/themes/empowering-with-advanced-analytics/internship-theses-at-wbaa/generation-of-synthetic-financial-time-series-with-generative-adversarial-networks.pdf
- [3] Jake Frankenfield, "Artificial Neural Network (ANN)", Aug 28, 2020, url: https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp
- [4] "Neural Network", url: https://deepai.org/machine-learning-glossary-and-terms/neuralnetwork
- [5] Alexander Thamm, "KI, künstliche neuronale Netze, Machine Learning, Deep Learning: Wir bringen Licht in die Begriffe rund um das Thema "Künstliche Intelligenz", Mai 29, 2018, url: https://www.alexanderthamm.com/ de/blog/ki_artificial-intelligence-ai-kuenstliche-neuronale-netze-machine-learningdeep-learning/
- [6] Craig Glastonbury, "A primer on artificial neural networks and convolutions for image data.", Aug 6, 2017, url: https://glastonburyc.github.io/neuralprimer.html
- [7] Roger Grosse, "Multilayer Perceptrons", 2018, url: https://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/slides/lec05.pdf
- [8] Zewen Li, Wenjie Yang, Shouheng Peng, Fan Liu, Member, IEEE, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects", 1 Apr 2020, url: https://arxiv.org/pdf/2004.02806.pdf
- [9] David Sheehan, "Visualising Activation Functions in Neural Networks", 2020, url: https://dashee87.github. io/deep%20learning/visualising-activation-functions-in-neural-networks/
- [10] Stefan Müller, "Generative Adversarial Networks: How Data Can Be Generated With Neural Networks", Oct 15, 2020, url: https://www.statworx.com/en/blog/generative-adversarial-networks-how-data-canbe-generated-with-neural-networks/
- [11] Aakash Pydi, "Using a Generative Adversarial Network (GAN) to Create Novel Artistic Images", June 9, 2019, url: https://towardsdatascience.com/using-a-generative-adversarial-network-gan-to-createnovel-artistic-images-e06c65ff43ad
- [12] Gautam Ramachandra, "Generative Adversarial Networks", Feb 26, 2017, url: https://medium.com/ @gautamrbharadwaj/generative-adversarial-networks-85900be25425
- [13] IBM Cloud Education, "What are Neural networks", Aug 17, 2020, url: https://www.ibm.com/cloud/learn/ neural-networks
- [14] F. Rosenblatt, "The Perceptron: A probabilistic model for information storage and organization in the brain", 1958, url: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type= pdf
- [15] F. Rosenblatt, "Perceptron", April 9, 2021, url: https://en.wikipedia.org/wiki/Perceptron
- [16] SAS Institut Inc., "Neural Networks "What they are and why they matter", Aug 28, 2020, url: https: //www.sas.com/en_sa/insights/analytics/neural-networks.html
- [17] Lalithnaryan C, "Perceptron Algorithm A Hands On Introduction", Nov 3, 2020, url: https://www.section. io/engineering-education/perceptron-algorithm/
- [18] Yoav Freund, Robert E. Schapire, "Large Margin Classification Using the Perceptron Algorithm", Nov 3, 1999, url: http://cseweb.ucsd.edu/~yfreund/papers/LargeMarginsUsingPerceptron.pdf
- [19] Dipl.-Ing. (FH) Stefan Luber & Nico Litzel, "Was ist ein Perzeptron?", Aug 18, 2018, url: https:// towardsdatascience.com/https-medium-com-francesco-cicala-whats-whys-and-hows-ofperceptron-f87c66f512c5
- [20] Francesco Cicala, "Perceptron: Whats, whys, and hows", Feb 2, 2019, url: https://www.bigdata-insider. de/was-ist-ein-perzeptron-a-798367/
- [21] Jason Brownlee, "How to Choose Loss Functions When Training Deep Learning Neural Networks", Aug 25, 2020, url: https://machinelearningmastery.com/how-to-choose-loss-functions-when-trainingdeep-learning-neural-networks/

- [22] Guru99, "Back Propagation Neural Network: Explained With Simple Example", 2021, url: https://www.guru99.com/backpropogation-neural-network.html
- [23] Sanket Doshi, "Various Optimization Algorithms For Training Neural Network", Jan 13, 2019, url: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6
- [24] Nagesh Singh Chauhan, "Optimization Algorithms in Neural Networks", Jan 2020, url: https://www. kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html
- [25] Jason Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", Jan 13, 2021, url: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- [26] Rohith Gandhi, "Generative Adversarial Networks Explained", Mai 10, 2018, url: https://towardsdatascience.com/generative-adversarial-networks-explained-34472718707a
- [27] Google developers, "Loss Functions", Feb 10, 2020, url: https://developers.google.com/machinelearning/gan/loss
- [28] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, Aaron Courville, "Improved Training of Wasserstein GANs", Dez 25, 2017, url: https://arxiv.org/pdf/1704.00028.pdf
- [29] Lilian Weng, "From GAN to WGAN", April 18, 2019, url: https://arxiv.org/pdf/1904.08994.pdf
- [30] Zhiwu Huang, Jiqing W, Luc Van Gool, "Manifold-valued Image Generation with Wasserstein Generative Adversarial Nets", Jan 3, 2019, url: https://arxiv.org/pdf/1712.01551.pdf
- [31] Floydhub, "Generative Adversarial Networks The Story So Far", June 21, 2019, url: https://blog.floydhub.com/gans-story-so-far/
- [32] John Thickstun, "Kantorovich-Rubinstein Duality", June 21, 2019, url: https://courses.cs.washington.edu/courses/cse599i/20au/resources/L12_duality.pdf
- [33] Sorta Insightful, "Read-through: Wasserstein GAN", June 21, 2019, url: https://www.alexirpan.com/2017/02/22/wasserstein-gan.html
- [34] Engineering Statistics Handbook, "Quantile-Quantile Plot", url: https://www.itl.nist.gov/div898/handbook/eda/section3/qqplot.htm
- [35] John McGonagle, George Shaikouski, Christopher Williams, and 3 others contributed, "Backpropagation", url: https://brilliant.org/wiki/backpropagation/
- [36] Wikipedia, "Liste von Kryptowährungen", April 28, 2021, url: https://de.wikipedia.org/wiki/Liste_ von_Kryptowährungen
- [37] Andrew Bloomenthal, "What Determines the Price of 1 Bitcoin?", March 16, 2021, url: https://www. investopedia.com/tech/what-determines-value-1-Bitcoin
- [38] Wikipedia, "History of Bitcoin", Mai 5, 2021, url: https://en.wikipedia.org/wiki/History_of_Bitcoin
- [39] Quantivity, "Why Log Returns", Feb. 21, 2011, url: https://quantivity.wordpress.com/2011/02/21/ why-log-returns/
- [40] Mr. Karthik Jilla, Dr. Sarat Chadra Nayak, Ms. Archana Bathula, "Stylized facts of financial time series: A comprehensive analysis", 2017, url: https://www.ijrter.com/published_special_issues/01-12-2017/ stylized-facts-of-financial-time-series-a-comprehensive-analysis.pdf
- [41] Rama Cont, "Volatility Clustering in Financial Markets: Empirical Facts and Agent-Based Models", 2017, url: https://link.springer.com/chapter/10.1007/978-3-540-34625-8_10
- [42] Vaneck, "Bitcoin's Correlation to Markets Hits a Record in 2020", 2017, url: https://www.etftrends.com/ tactical-allocation-channel/Bitcoins-correlation-to-markets-hits-a-record-in-2020/
- [43] "Time Series Analysis for Log Returns of S&P500", url: https://ionides.github.io/531w18/midterm_project/project38/Midterm_proj.html
- [44] "Crypto assets: An emerging asset class", Oct. 11, 2019, url: https://www.seba.swiss/research/cryptoassets-an-emerging-asset-class
- [45] "BTC-USD", url: https://www.tradingview.com/symbols/BTCUSD/history-timeline/#Bitcoinbump-2021-03-29
- [46] Jonathan Hui, "GAN Wasserstein GAN & WGAN-GP", June 14, 2018, url: https://jonathan-hui. medium.com/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490

- [47] Jonathan Hui, "Backpropagation Step by Step", Nov 3, 2019, url: https://hmkcode.com/ai/ backpropagation-step-by-step/
- [48] Purnasai Gudikandula, "Recurrent Neural Networks and LSTM explained", Mai 27, 2019, url: https: //purnasaigudikandula.medium.com/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9
- [49] Kaggle, "Intro to Recurrent Neural Networks LSTM", 2018, url: https://www.kaggle.com/ thebrownviking20/intro-to-recurrent-neural-networks-lstm-gru
- [50] Kaggle, "Funktionsweise künstlicher neuronaler Netze ", Aug. 31, 2018, url: https://data-science-blog. com/blog/2018/08/31/funktionsweise-kunstlicher-neuronaler-netze/
- [51] Harshit Dwivedi, "Understanding GAN Loss Functions", 23rd, 2021, url: https://neptune.ai/blog/ganloss-functions
- [52] Santanu Khan, "Calculate Value-at-Risk Using Wasserstein Generative Adversarial Networks (WGAN-GP) for Risk Management System", Mar 17, 2019, url: https://chatbotslife.com/calculate-value-at-riskusing-wasserstein-generative-adversarial-networks-wgan-gp-for-risk-2b1d320fde59
- [53] Tsay Ruey, "Analysis of financial time series", Mar 17, 2005, url: https://cpb-us-w2.wpmucdn.com/blog. nus.edu.sg/dist/0/6796/files/2017/03/analysis-of-financial-time-series-copy-2ffgm3v. pdf
- [54] Manhar Singh Walia, "Synthetic Data Generation Using Wasserstein Conditional Gans With Gradient Penalty (WCGANS-GP)", 2020, url: https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1236& context=scschcomdis
- [55] David Foster, "Chapter 4. Generative Adversarial Networks", url: https://www.oreilly.com/library/ view/generative-deep-learning/9781492041931/ch04.html
- [56] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, "Generative Adversarial Nets", 2014, url: https://arxiv.org/pdf/1406.2661.pdf
- [57] Zhengwei Wang, Qi She, Tomas E. Ward, "Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy", url: https://arxiv.org/pdf/1906.01529.pdf
- [58] Victor Zhou, Machine Learning for Beginners: An Introduction to Neural Networks", url: https:// towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neuralnetworks-d49f22d238f9
- [59] Anirban Chakraborti, Ioane Muni Toke, Marco Patriarca, Frédéric Abergel, "Econophysics review: I. Empirical facts", url: https://hal.archives-ouvertes.fr/hal-00621058/document
- [60] Stephen Chan, Jeffrey Chu, Saralees Nadarajah and Joerg Osterrieder, "A Statistical Analysis of Cryptocurrencies", url: https://www.mdpi.com/1911-8074/10/2/12
- [61] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever, "An Empirical Exploration of Recurrent Network Architectures", url: http://proceedings.mlr.press/v37/jozefowicz15.html
- [62] Hansika Hewamalage, Christoph Bergmeir, Kasun Bandara, "Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions", url: https://arxiv.org/pdf/1909.00590.pdf
- [63] Rajat Sen, Hsiang-Fu Yu, and Inderjit Dhillon, "Think Globally, Act Locally: A Deep Neural Network Approach to High-Dimensional Time Series Forecasting", url: https://arxiv.org/pdf/1905.03806.pdf
- [64] Samuel Assefa, JP Morgan Chase, "Generating Synthetic Data in Finance: Opportunities, Challenges and Pitfalls", url: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3634235
- [65] Anita Yadava, C K Jha, Aditi Sharan, "Optimizing LSTM for time series prediction in Indian stock market", url: https://www.sciencedirect.com/science/article/pii/S1877050920307237
- [66] Dmitry Efimov, Di Xu, Luyang Kong, Alexey Nefedov, "Using Generative Adversarial Networks to Synthesize Artificial Financial Datasets", url: https://arxiv.org/pdf/2002.02271.pdf
- [67] Hao Ni, Lukasz Szpruch, Magnus Wiese, Shujian Liao, Baoren Xiao, "Conditional Sig-Wasserstein GANs for Time Series Generation", url: https://arxiv.org/abs/2006.05421
- [68] Kaleb E Smith, Anthony O Smith, "Conditional GAN for timeseries generation", url: https://arxiv.org/ abs/2006.16477
- [69] J.B. Maverick, "Why The 50-day Simple Moving Average Is Popular Among Traders", url: https://www. investopedia.com/ask/answers/012815/why-50-simple-moving-average-sma-so-commontraders-and-analysts.asp

[70] Arash Ardakani, Zhengyun Ji, and Warren J. Gross McGill University, Montreal, Canada, "Learning to Skip Ineffectual Recurrent Computations in LSTMs", url: https://arxiv.org/pdf/1811.10396.pdf