



Degree Program in
Data Science and Management

Course of Big Data and Smart Data Analytics

Stock Price Prediction with Evolutionary Generative Adversarial Network

Prof. Irene Finocchi

SUPERVISOR

Prof. Alessio Martino

CO-SUPERVISOR

Miro Confalone 757441

CANDIDATE

Academic Year 2022/2023

Abstract

Generative Adversarial Networks (GANs) represent a powerful tool for generating realistic data samples, they have been used in multiple settings, both for regression and classification tasks and for time series, even if the application in this field has proven challenging due to the complex and dynamic nature of such data. In this thesis, it is proposed an evolutionary approach to GAN optimization, using genetic algorithms. GANs are inherently unstable for regression because of the adversarial part, and this research tries to investigate if evolutionary algorithms can improve GAN performances. Overall, this work contributes to the development of GAN-based methods for time series analysis and applies the optimization to two types of GAN, a Wasserstein GAN, and a standard GAN. The proposed method has the potential to be applied in various domains, including finance, healthcare, and environmental monitoring.

Keywords: Generative Adversarial Networks, Genetic algorithms, Time Series, Evolutionary algorithms.

Contents

1	Introduction	6
1.1	Research Objective and Motivation	7
1.1.1	Research Question	8
1.1.2	Thesis Outline	8
2	Generative AI	9
2.1	Generative vs Discriminative Models	9
2.2	Generative AI models	11
2.3	Generative Adversarial Networks	15
2.3.1	Improving the GAN model	18
3	Time Series and Time Series Analysis	21
3.0.1	Time Series data	21
3.0.2	Statistical models for time series snalysis	22
3.0.3	Machine Learning models for Time Series analysis	24
4	Evolutionary Algorithms	27
4.1	Genetic Algoritms	29
5	Stock Price Prediction using Evolutionary GAN	33
5.1	Introduction	33
5.2	Related Work	34
5.3	Methodology	34
5.3.1	Training and results	42
6	Conclusion and further developments	50

List of Figures

2.1	Discriminative and Generative Models (Amidi and Amidi, 2023)	10
2.2	Transformer model architecture (Vaswani et al., 2017)	12
2.3	An autoencoder example. The input image is encoded to a compressed representation and then decoded (Bank et al., 2020)	13
2.4	The structure of a VAE (Rocca, 2019)	14
2.5	Typical GAN architecture	16
2.6	Data having a low dimension in high dimension space can hardly have overlapped. (Left) Two lines in a three-dimension space. (Right) Two surfaces in a three-dimension space.(Weng, 2019)	19
3.1	Stationary vs non stationary time series (Bauer, 2021)	22
3.2	The LSTM chain(Olah, 2015)	26
4.1	The various fields of application of EAs are shown(Olah, 2015)	28
4.2	Population, Chromosome, Gene, and allele(Tutorialpoints, 2022)	30
4.3	Different types of crossovers(Sobeih et al., 2010)	31
4.4	Different types of mutation(Liu, 2014)	32
5.1	The Evolutionary GAN as written by(Wang et al., 2018)	34
5.2	The relative Fourier Transformation, sine waves produced approximate the original price line in (<i>red line</i>), hence helping the model to represent the trend better	35
5.3	Here the real price is added to the output of the Generator to enhance data length and offer more context, the discriminator receives only prices as input and not all the other variables Lin et al. (2021)	38
5.4	The Diagram of the proposed model	40
5.5	The result on the test set with the Baseline LSTM	42
5.6	The plot of train and validation loss for LSTM	42

5.7	The result on the test set with the Baseline GRU	43
5.8	The plot of train and validation loss for GRU	43
5.9	The result on the test set with the Baseline GRU	44
5.10	Plot of losses for the Discriminator and the Generator	44
5.11	The result on the test set for the Basic GAN optimized with GA	45
5.12	Left: the tSNE plot of the Basic GAN with no optimization, Right: the one optimized with the evolutionary approach	46
5.13	The result on the test set for the Wasserstein GAN with Gradient Penalty .	46
5.14	Plot of losses for the Discriminator and the Generator	47
5.15	The result on the test set for the Wasserstein GAN with Gradient Penalty optimized with GA	48
5.16	Left: the tSNE plot of the WGAN GP with no optimization, Right: the optimized one	48
6.4	Left: the tSNE plot of the Basic GAN with no optimization, Right: the one optimized with the evolutionary approach	65

List of Tables

- 5.1 Explanation of Features 36
- 5.2 Comparison of Original and Optimized Hyperparameters of Basic GAN . . . 45
- 5.3 Comparison of Original and Optimized Hyperparameters WGAN GP 47
- 5.4 Comparison of Models Performance 49
- 5.5 JS and KL Divergence of the various models, the asterisk underline where the optimization reduced one of the two metrics 49

Chapter 1

Introduction

Forecasting the market has always been an area that involved many techniques and different approaches. Today, institutional investors are heavily investing in artificial intelligence and machine learning cutting-edge technologies to achieve the most accurate results and predictions. For instance, JPMorgan Chase invests 14 billion USD a year in Artificial Intelligence technologies(Kahn, 2023).

Machine learning and artificial intelligence (AI) have a long history, with the concept of a machine mirroring human cognition and behavior being a topic of discourse for many decades. This discourse has led to remarkable advancements in diverse domains, ranging from natural language processing to computer vision.

Arthur Samuel, as early as 1959, developed a program proficient in playing checkers, rivaling human players. This program was grounded in the principle of reinforcement learning, where the computer leveraged its past experiences to enhance its performance. (Samuel, 1959)

Post Samuel's pioneering work, machine learning evolved significantly. In the 1980s, the emergence of neural networks, a class of algorithms inspired by the architecture of the brain, initiated a revolution in computer science, leading to the current state-of-the-art image and speech recognition systems.

The dawn of the 21st century witnessed the rise of Big Data and novel algorithms capable of harnessing it, ushering in a new epoch of machine learning and AI. Deep learning, a variant of machine learning rooted in multi-layered neural networks, laid the foundation for advancements in image and speech recognition, natural language processing (NLP), and other applications.

In parallel, the field of evolutionary computation, particularly genetic algorithms, has also

made significant strides. These algorithms, inspired by the principles of natural selection and genetics, have been used to solve optimization problems in various domains, including financial forecasting. They work by generating a population of possible solutions and iteratively refining them through processes analogous to mutation, crossover, and selection.

Innovative algorithms and techniques, such as generative adversarial networks (GANs), reinforcement learning, and genetic algorithms, are continually expanding the capabilities of machine learning systems. These techniques have empowered machines to learn from a handful of labeled examples and even without any labeled data.

The more the results of a model are close to the true ones, the more confident institutional investors can be in their decision-making process. Accurate predictions can lead to more effective investment strategies, better risk management, and potentially higher returns. In the context of price prediction, this could mean more profitable trades, improved portfolio performance, and a significant competitive advantage in the fast-paced financial market.

1.1 Research Objective and Motivation

The primary motivation behind this thesis is to explore a novel approach to optimizing Generative Adversarial Networks (GANs) using genetic algorithms. This innovative approach seeks to contribute to the existing body of knowledge by extending the application of GANs to time-series data, a domain that has yet to be thoroughly explored.

In their work, Wang et al. (2018) employed a similar strategy, but their research did not extend to the realm of time-series data. Time-series data possess unique characteristics that set them apart from non-temporal data. They encapsulate patterns, trends, and seasonality that require specialized techniques for identification and analysis. In contrast, non-temporal data can be examined using traditional statistical techniques, such as measures of association or group differences. Moreover, financial time series have high value for various industries.

The objective of this research is to continue on the research made by Lin et al. (2021), specifically on the hyperparameters tuning of their models, which has been performed through genetic algorithm. This comparison aims to provide a comprehensive understanding of the performance of the proposed approach in relation to established models and investigate the use of genetic algorithms applied to regression problems and neural network optimization, since complex neural networks especially, GANs can suffer from instability and inconsistency. (Nguyen et al., 2023)

The findings from this study could potentially pave the way for future research in this

area, and open up new avenues for the application of GANs in time-series analysis.

1.1.1 Research Question

The development of this thesis is guided by the following research question:

Can the performances of a Generative Adversarial Network be improved by Genetic Algorithms and better predict the stock price?

1.1.2 Thesis Outline

In Chapter 2 there is an introduction to Generative AI and the various models.

Chapter 3 describes the peculiarity of time series data and the various approaches that are applied when dealing with them and in Chapter 4 while in Chapter 5 the methodologies that have been used to carry out the objective of the research, the end part of Chapter 5 and Chapter 6 will discuss the results and the discussion on further improvements and approaches.

Chapter 2

Generative AI

This chapter delves into the realm of Generative Artificial Intelligence, the main models belonging to this family, their characteristics, and what distinguishes them from discriminative ones. A significant focus is placed on Generative Adversarial Networks (GANs), the neural network utilized in the empirical research of this thesis. The chapter serves as an introduction to Generative AI and a precursor to the in-depth exploration of GANs in the empirical research section

2.1 Generative vs Discriminative Models

The realm of Artificial Intelligence (AI) is in a state of perpetual evolution, continually pushing the boundaries of what is possible.

Today, AI has progressed beyond merely processing vast amounts of data for regression or classification tasks. It has ventured into the domain of generating new data from noise or other examples fed into a model.

This new frontier is known as Generative AI, which includes models capable of producing text, images, soundtracks, videos, or simply synthetic data. Generative models stand distinct from discriminative models due to their explicit assumptions about the class conditional distributions $p(\mathbf{x} | y)$. These assumptions inherently translate to assumptions about the marginal $p(x)$ as well as the joint distribution $p(\mathbf{x}, y)$. On the other hand, discriminative models explicitly model the posterior $p(y | \mathbf{x})$, making assumptions about the parametric form of the same $p(y | \mathbf{x})$, and utilize the training process to learn the parameters (Banerjee, 2007).

Among the various discriminative models, one can find linear and logistic regression, decision

trees, Artificial Neural Networks, Recurrent Neural Networks, etc.

In this specific class of models statistical bias is a concept that warrants attention in this context. Suppose we repeatedly draw training samples S_1, \dots, S_l , each of size m , and apply our learning algorithm A to construct hypotheses $\hat{f}_{S_1}, \hat{f}_{S_2}, \dots, \hat{f}_{S_l}$, where \hat{f}_{S_i} denotes the hypothesis $A(S_i)$. We can amalgamate all of these different hypotheses into a mean of hypothesis:

$$\bar{\hat{f}}(x) = \lim_{l \rightarrow \infty} \frac{1}{l} \sum_{i=1}^l \hat{f}_{S_i}(x).$$

The value $\bar{\hat{f}}(x)$ represents the expected predicted value of $f(x)$, where the expectation is taken over all possible training samples of fixed size m (Dietterich and Kong, 1995). Statistical bias is then defined as the error between $f(x)$ and $\bar{\hat{f}}(x)$, serving as a measure of systematic error for a given sample size. As this error is systematic, it can be reduced.

Generative models, due to their stronger assumptions about the data given an output variable, can lead to higher bias, resulting in higher asymptotic error (Banerjee, 2007). This potential for higher bias underscores the need for careful consideration when choosing between generative and discriminative models, depending on the specific requirements and constraints of the task at hand. In figure 2.1 the difference between the two classes is shown:

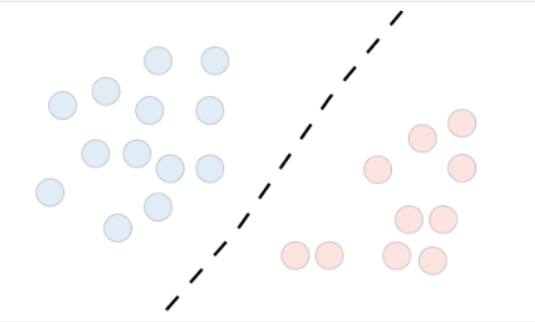
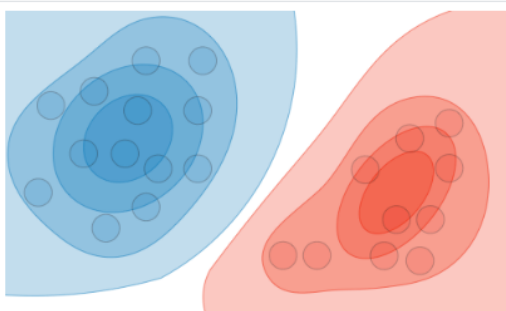
	Discriminative model	Generative model
Goal	Directly estimate $P(y x)$	Estimate $P(x y)$ to then deduce $P(y x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Figure 2.1: Discriminative and Generative Models (Amidi and Amidi, 2023)

2.2 Generative AI models

Among the myriad of generative AI models, we find those based on game theory, such as the family of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), autoregressive models like GPT, PixelRNN, WaveNet, and other approaches such as diffusion models, Variational Auto Encoders (VAE), Hidden Markov Machines (HMM), and Latent Dirichlet Allocation (LDA).

Autoregressive Models and GPT

Autoregressive models, including the Generative Pre-trained Transformer (GPT), have garnered significant attention in recent years. Often referred to as Large Language Models (LLMs), these models excel in performing complex natural language processing (NLP) tasks. They are built upon deep neural networks and are typically trained on vast amounts of data using unsupervised learning techniques. The advent of LLMs has been facilitated by the exponential increase in computational power and the availability of large-scale datasets.

GPT, introduced by Radford et al. (2018), is an autoregressive and unidirectional model. This means that it generates information in the order the text is presented. GPT models are based on Transformer models, which were introduced by Vaswani et al. (2017).

Transformers are models that make exclusive utilization of attention mechanisms. These mechanisms enable the model to assign different weights to specific parts of the input sequence based on their relevance to the task. This weighted approach allows the model to focus selectively on the most pertinent information.

One of the most common types of attention mechanisms employed is “self-attention” or “intra-attention”. In this setup, the model generates a set of “query,” “key,” and “value” vectors for each position in the input sequence. The dot product between the query and key vectors is then used to compute a weight for each position. This process culminates in a weighted sum of the values, which forms the output of the self-attention mechanism, effectively providing a summary or representation of the input (Vaswani et al., 2017).

The key point to understand is that the attention mechanism allows the model to focus more on the important parts of the input sequence, and less on the less important parts. This is done by assigning higher weights to the important parts, and lower weights to the less important parts. The weighted sum of the values then provides a summary of the input sequence that takes into account the relative importance of each part. In figure 2.2, the model architecture of a transformer.

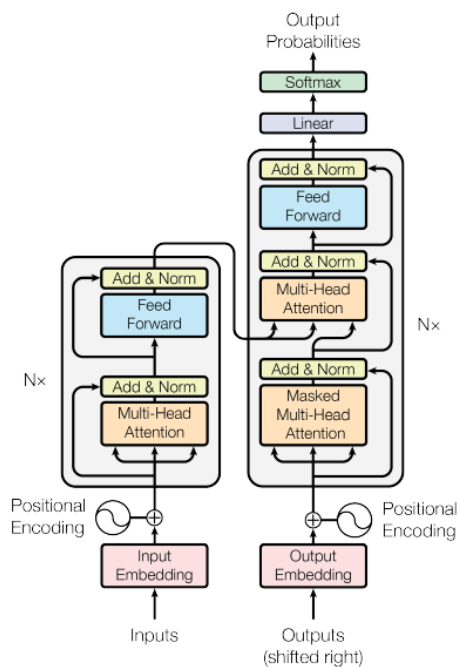


Figure 2.2: Transformer model architecture (Vaswani et al., 2017)

One of the key advantages of LLMs is their ability to perform a wide range of NLP tasks, including language translation, question answering, text classification, and sentiment analysis. These models can learn the underlying patterns and structures of language, allowing them to generate coherent and grammatically correct text.

LLMs are also highly flexible and can be fine-tuned for specific tasks. For example, a pre-trained language model like GPT can be fine-tuned to generate human-like text in a particular domain, such as legal documents or medical reports. This makes LLMs a powerful tool for a wide range of applications, including chatbots, virtual assistants, and automated content creation.

Autoencoders and Variational Autoencoders

Autoencoders (AEs) are a type of neural network frequently used for unsupervised learning, dimensionality reduction, and generative modeling. The fundamental architecture of an AE includes an encoder network, which maps the input data to a lower-dimensional latent space, and a decoder network, which maps the latent representation back to the original input space. The latent space is defined as *“In artificial intelligence ‘Latent Space’ refers to a mathematical space which maps what a neural network has learned from training images. Once it has been trained it understands all images of trees as existing in a specific area, and*

all images of birds in another.” (Elwes, 2021).

Autoencoders are trained by minimizing a reconstruction loss, which measures the discrepancy between the original input and the reconstructed output, as illustrated in Figure 2.3.

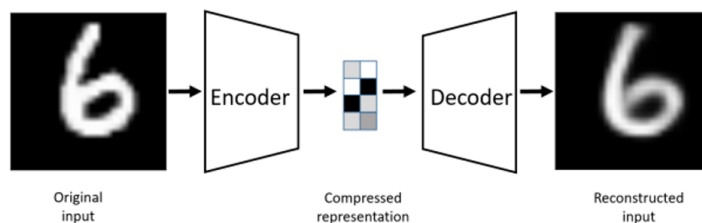


Figure 2.3: An autoencoder example. The input image is encoded to a compressed representation and then decoded (Bank et al., 2020)

The loss function for an AE can be written as follows, which is also called Reconstruction Loss:

$$\mathcal{L}_{\text{recon}} = \sum_{i=1}^N \|x_i - \hat{x}_i\|^2$$

Here, x_i is the original input data, \hat{x}_i is the output of the decoder for the corresponding input, and N is the number of training examples. \mathcal{L} can be any metric such as Mean Squared Error or Cross-Entropy.(Jordan, 2018)

One drawback of standard AEs is that they typically learn an unstructured latent space that may not be useful for downstream tasks such as classification or generation. This is where Variational Autoencoders (VAEs) come in.

VAEs introduce a probabilistic component to the latent space by modeling it as a probability distribution, typically a Gaussian distribution. (Bank et al., 2020)

VAEs hence are a type of unsupervised learning model that differently from simple autoencoder can generate new data samples from a given dataset. VAEs are based on the same mechanism of AEs and depend on the quality and the distribution of the latent space to generate new data, it can happen that the quality of the latent space is not optimal, hence, VAEs introduce regularization during the training, aiming at avoiding overfitting and ensuring the latent space has good properties.(Rocca, 2019)

The encoder network in a VAE maps the input data to the parameters of the latent distribution, which are the mean and standard deviation of the Gaussian. The decoder network then samples from this distribution to generate a latent representation, which is then decoded back into the original input space.

VAEs are trained by minimizing a loss function that consists of two parts: a Reconstruction Loss and a KL divergence term that measures the difference between the learned latent distribution and the prior distribution.

$$\text{Loss} = \mathcal{L}(x, \hat{x}) + \sum_j KL(q_j(z | x) || p(z))$$

We want our KL divergence and Reconstruction Loss to be as small as possible, with KL Divergence > 0 . Figure 2.4 shows the VAE is shown with the addition of the probabilistic component.

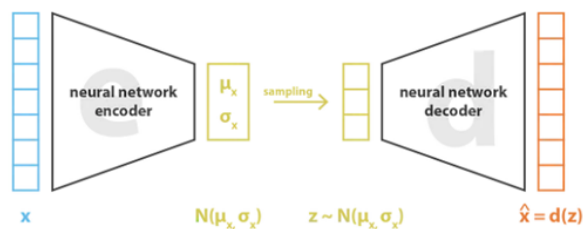


Figure 2.4: The structure of a VAE (Rocca, 2019)

One advantage is that VAEs can generate new data samples that are similar to the training data, while also exploring new regions of the data space. This is because VAEs are able to interpolate between different regions of the latent space to generate new samples.

Hidden Markov Models

Another type of generative model is represented by **Hidden Markov Models** (HMMs). HMMs are a statistical model that can be used to generate sequences of observations based on a set of hidden states. HMMs have been used in a wide range of applications, such as for NLP tasks, computational finance, but also speed analysis.

As the name suggests Hidden Markov Models are based on Markov Chain, in which future observations are independent of the past but only dependent on the present. In other words, given the present observation, the future is independent of the past.

$$P(x_{t+1} | x_1, x_2, \dots, x_t) = P(x_{t+1} | x_t)$$

In HMM both observed data and hidden states are considered causal factors in the probabilistic model. The hidden states represent the underlying structure of the data sequence. (Daniel Ju-

rafsky, 2023; Rabiner and Juang, 1986)

HMMs have several advantages over other generative models. One advantage is that HMMs can model sequences with complex temporal dependencies. They are able to capture long-term dependencies in the data sequence through the sequence of hidden states.

HMMs can generate new data sequences by sampling from the probability distribution over the hidden states, while also recognizing existing sequences by computing the probability of the observed data sequence given the model parameters.

2.3 Generative Adversarial Networks

In his seminal paper, Goodfellow et al. (2014) introduced a novel type of generative model known as Generative Adversarial Networks (GANs). This marked a significant breakthrough in the field of deep learning. GANs are grounded in the concept of game theory and employ a dual neural network architecture, consisting of a generator and a discriminator.

These models are based on the adversarial modeling framework, where both models are multilayer perceptrons. The generative model, or the generator, is put against an adversary: a discriminative model, or the Discriminator. The discriminator learns to recognize whether a sample originates from the generator or from the original data. The generator can be likened to a counterfeiter, whose goal is to deceive the discriminator that the data it generates are real.

During training, both models improve until the products generated by the generator become indistinguishable from genuine data (Goodfellow et al., 2014).

A prior distribution on input noise variables, $p_{\mathbf{z}}(\mathbf{z})$, is defined to learn the generator's distribution over data x . This represents a mapping to data space as $G(\mathbf{z}; \theta_g)$, where G is a differentiable function with parameters θ_g .

A second deep neural network, $D(\mathbf{x}; \theta_d)$, with θ_d , is defined and represented by the Discriminator. This outputs a number. Here, $D(\mathbf{x})$ is the probability that x comes from real input rather than from the generator $p(g)$.

The discriminator is trained to maximize the probability of assigning the correct label to both generated and real data. Simultaneously, G tries to make $\log(1 - D(G(\mathbf{z})))$ as little as possible, implying that we want $D(G(\mathbf{z}))$ to be as small as possible (Goodfellow et al., 2014). Figure 2.5 depicts the classical GAN architecture

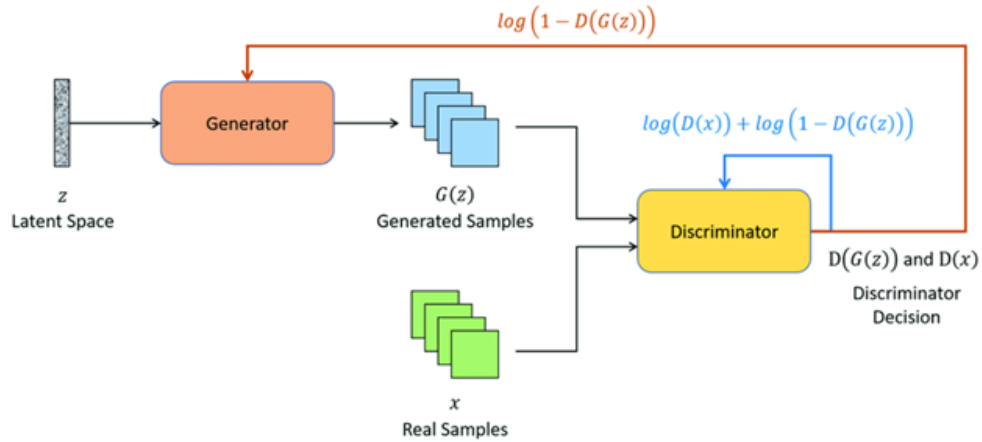


Figure 2.5: Typical GAN architecture

In Goodfellow et al. (2014) paper the idea is that D and G play a two-player minimax simultaneously $V(G, D)$.

$$\min_G \max_D V(D, G) = E_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + E_{z \sim p_z(z)}[\log(1 - D(G(z)))] \quad (1)$$

Equation 1 that reaches the optimum when $p(z) = p(\text{data})$.

In Algorithm from Goodfellow et al. (2014) 1 is shown how the GAN is optimized.

If the discriminator is trained until it reaches its optimal state before every update of the generator's parameters, then the process of minimizing the value function effectively becomes equivalent to minimizing the Jensen-Shannon divergence between the real data distribution (P_{data}) and the generated data distribution (P_z)

Reaching the optimality $P(z) = P_{\text{data}}$

First, the optimal discriminator D for any given output of G is given:

Proposition 1. For G fixed, the optimal discriminator D is

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_z(\mathbf{x})}$$

Inserting $p_{\text{data}}(x) = p_z(x)$ we get

$$D_G^*(x) = \frac{p_z(x)}{p_z(x) + p_z(x)} = \frac{1}{2}$$

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets

Require: Number of training iterations, number of discriminator steps k , batch size m , generator G , discriminator D

1: **for** number of training iterations **do**

2: **for** k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$
- Sample minibatch of m examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$
- Update the discriminator by ascending its stochastic gradient:

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

3: **end for**

- Sample minibatch of m noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(1 - D(G(z^{(i)})))$$

4: **end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

So, if the generator is able to trick the discriminator into producing data that are so similar to real input then the discriminator is only able to label those data based on randomly guessing the classification with a probability equal to 0.5.

The foundational concept of Generative Adversarial Networks (GANs) draws from the idea of undirected graphical models featuring hidden variables. This encompasses models like Restricted Boltzmann Machines (RBMs) and Deep Boltzmann Machines (DBMs).

They use a probabilistic approach to model the distribution of data and they are trained using a method called maximum likelihood estimation (MLE). The key difference between GANs and RBMs or DBMs is that GANs use a discriminative approach to train the generator, while RBMs and DBMs use a generative approach to learn the underlying distribution of the data. (Bengio et al., 2012) GANs have been shown to be highly effective in generating realistic data, but they can be more difficult to train than RBMs or DBMs due to the minimax game that is used to train the generator and discriminator.

2.3.1 Improving the GAN model

Wasserstein GAN and Wasserstein GAN with Gradient Penalty

As said before the architecture of the GAN implicitly tends to minimize the Jensen-Shannon Divergence, so, various methods have been introduced to make the model more stable, since as shown by Arjovsky et al. (2017) GAN training is delicate and unstable, and since the whole base of these networks is the adversarial part, it is difficult for the model to achieve Nash Equilibrium.

Another problem that affects GANs is low dimensionality, meaning that the dimension of real data are kind of stuck in low dimension but they appear to be high, for example, a certain object like a human face has to have some pre-defined characteristics and this pushes also the generated data to have low dimensionality. (Weng, 2019)

When $P(g)$ and P_{data} are in low dimensional space they hardly overlap, they are disjoint, meaning that can be a discriminator that always and with certainty will detect fake samples, this is shown in Figure 2.6.

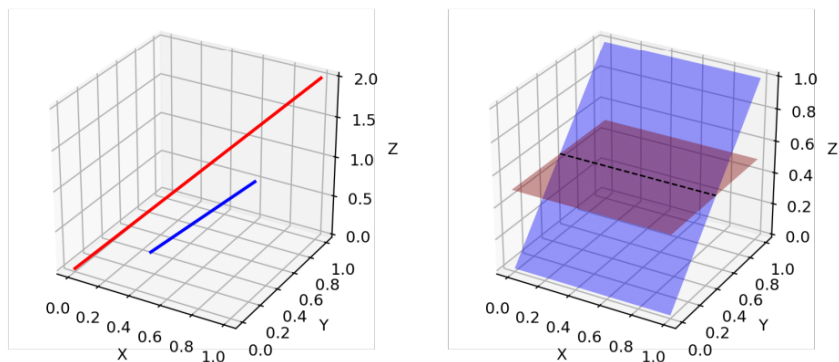


Figure 2.6: Data having a low dimension in high dimension space can hardly have overlapped. (Left) Two lines in a three-dimension space. (Right) Two surfaces in a three-dimension space.(Weng, 2019)

Another problem is given by the vanishing gradient problem, when the discriminator is perfect, the loss function falls to zero, leading to no gradient for updating the loss during learning iterations. This creates a dilemma: if the discriminator performs poorly, the generator lacks accurate feedback, and if the discriminator performs too well, the learning becomes extremely slow or even halts.

The final problem to note is mode collapse, a scenario in which the generator might default to a state where it consistently produces identical outputs. This represents a frequent pitfall for GANs, as they struggle to accurately capture the intricacies of real-world data distributions, thus getting trapped in a confined space exhibiting negligible variety. To counter these challenges, Arjovsky et al. (2017) introduced an improved to classic GAN called Wasserstein GAN which instead of minimizing the JS Divergence has the objective of minimizing the Wasserstein Distance or Earth Mover’s distance, and it can be informally interpreted as there are two different piles of dirt representing two different distributions. Now, you want to transform the first pile to look exactly like the second one. The EMD would be the minimum amount of work you need to do to achieve this, where “work” is defined as the amount of dirt you move multiplied by the distance you move it. (Weng, 2019)

Hence, the loss function is the following: from the “discriminator” w is learn to find a good f_w loss is set as measuring the Wasserstein distance between p_r and p_g

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} E_{x \sim p_r} [f_w(x)] - E_{z \sim p_r(z)} [f_w(g_\theta(z))]$$

Hence, the “discriminator” has not more the task of telling the fake output from the real

ones. Instead, it learned a K -Lipschitz continuous function to help computing Wasserstein distance. A function $f : R \rightarrow R$ is said to be Lipschitz continuous if there exists a constant $K \geq 0$ such that for all $x, y \in R$,

$$|f(x) - f(y)| \leq K|x - y|$$

The smallest such L is called the Lipschitz constant of f . As the loss function diminishes during the training phase, the Wasserstein distance contracts, bringing the output of the generator model increasingly in line with the actual data distribution.

One big problem is to maintain the K -Lipschitz continuity of f_w during the training one trick presented by Arjovsky et al. (2017), is that following each gradient update, the weights w are confined within a small interval, for instance, $[-0.01, 0.01]$. This results in a confined parameter space W , ensuring that f_w retains its lower and upper limits to maintain Lipschitz continuity.

Still vanilla WGAN suffers from instability, slow convergence and vanishing gradient.

Different GAN models

- **DCGAN: Deep Convolutional GAN:** A type of GAN architecture that uses convolutional neural networks (CNNs) in both the generator and discriminator. It was introduced by Radford et al. (2016) and has been widely used for generating high-quality images.
- **CycleGAN (CGAN):** Can be used for image-to-image translation tasks, such as converting images from one domain to another. It was introduced by Zhu et al. (2017) and uses a cycle-consistency loss to ensure that the mapping is consistent in both directions.
- **StyleGAN:** It can be used for generating high-quality images with diverse styles. It was introduced by Karras et al. (2019) and uses a mapping network to control the styles of the generated images.
- **TimeGAN:** A type of GAN that is designed to generate real time series data. It was introduced by Yoon et al. (2019) and uses an autoregressive model in the generator and a bidirectional RNN in the discriminator. TimeGAN also uses a modified form of the Wasserstein loss function to train the model.

Chapter 3

Time Series and Time Series Analysis

This chapter discusses time series data, the inherent characteristics of this special class of data, and the various methods that have been utilized, both from statistics and machine learning and Artificial Intelligence

3.0.1 Time Series data

Time series data differ from other kinds of data since they are collected sequentially over time at regular or irregular intervals. Time series analysis is usually made to detect patterns, anomalies, and trends in the data in order to make predictions.

Each data point is associated with a timestamp which can be of any magnitude possible, seconds, weekly, daily, or in a different order. The order of the data points is crucial for understanding the temporal dynamics and dependencies. Many time series exhibit repeating patterns or cycles over fixed time periods, such as daily, weekly, or yearly patterns, this can take the name of *seasonality*. An example of a repeating cycle or seasonality can be represented by agricultural production, holiday seasons, energy consumption, etc.

Another important factor to take into consideration is the exhibition of long-term increases or decreases in the mean level, known as trends, this can be upward or downward and can last for a random period of time. Real-world time series data often contains random variations, measurement errors, or other unexplained fluctuations, collectively referred to as noise.

Many techniques to analyze time series require that the temporal data that are being analyzed are stationary. A time series is considered stationary if its statistical properties remain constant over time, hence they don't present trends or seasonality patterns. The differences are shown in Figure 3.1 Stationarity is often achieved through differentiation techniques. Time series data often have correlations between successive observations, known as autocor-

relation. Autocorrelation can help identify periodicities, seasonality, and persistence in the data and inform appropriate models for forecasting or anomaly detection.

In conclusion, time series data and their particular characteristics can lead to extracting meaningful insights and making informed decisions across various fields. There exists a multitude of models and techniques that can be applied to time series.

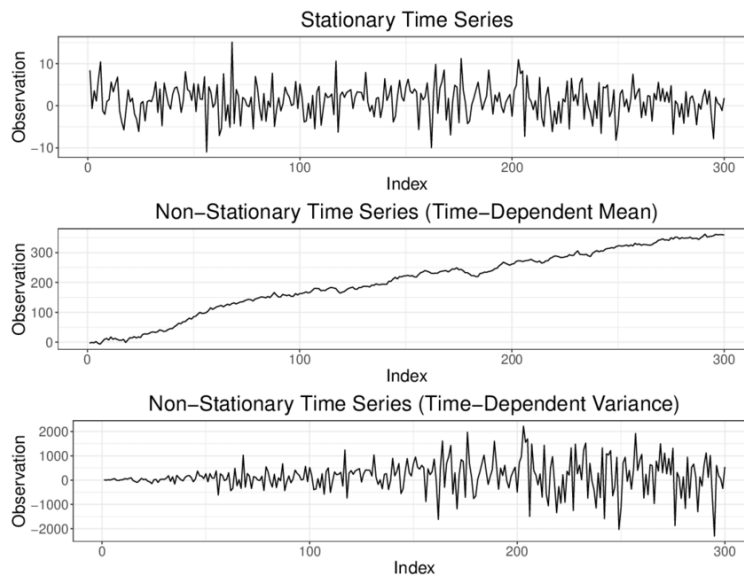


Figure 3.1: Stationary vs non stationary time series (Bauer, 2021)

3.0.2 Statistical models for time series analysis

There exist a multitude of models and techniques that can be applied to time series, classical methods for time series analysis include autoregressive (AR) models, moving average (MA) models, and autoregressive integrated moving average (ARIMA) models. These models capture linear relationships between successive observations and they usually function well for stationary time series data.

Autoregressive models

An autoregression model of order p bases its fundamentals on regressing the time series over itself and can be written as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t$$

where ϵ is white noise. This can be referred to as **AR**(p) model.

Moving Average Models

A moving average term in a time series model is a past error time coefficient.

Let $w_t \stackrel{iid}{\sim} N(0, \sigma_w^2)$, w_t are iid, each with a normal distribution with 0 mean and equal variance.

Here is a moving average model of order q or **MA**(q) model.:

$$x_t = \mu + w_t + \theta_1 w_{t-1} + \theta_2 w_{t-2} + \cdots + \theta_q w_{t-q}$$

Autoregressive moving average model (ARMA)

As the name suggests ARMA models combine the strengths of **AR**(p) model and **MA**(q) model, allowing it to capture a wider range of temporal dependencies in the data. The ARMA(p, q) model is represented as:

$$X_t = c + \phi_1 X_{t-1} + \cdots + \phi_p X_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q}. \quad (3.1)$$

By integrating both AR and MA components, the ARMA model can better describe complex autocorrelation structures, such as those with both short and long-term dependencies. In addition, ARMA models allow for choice between the order of q and the order of p .

Autoregressive Integrated Moving Average (ARIMA)

ARIMA models are usually denoted as ARIMA (p, d, q) where p is the order of the AR model, d is the degree of differencing, and q is the order of the MA model.

ARIMA models can be written as:

$$y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where y'_t is the differenced series, the degree of y_t depends on how much is differenced, and there is no limit on the amount of differencing. It can be said that the ARIMA(p, d, q) model is defined as the ARMA(p, q) model applied to the d -th order differenced time series.

So, moving from ARMA to ARIMA it can be noted that by incorporating differencing, the ARIMA model can handle a much larger range of time series data, including those with trends and other non-stationary components.

3.0.3 Machine Learning models for Time Series analysis

Machine learning and deep learning models have risen in popularity in dealing with temporal data.

In deciding which model to use it needs to be kept in mind that it is still a heuristic process and the performance of a certain model varies with data and number of training iterations, in statistics and machine learning the *No Free Lunch* theorem by Wolpert (1996) always apply.(Cerqueira et al., 2019)

Among the several machine learning models used in working with time series data can be found models such as Random Forests, Support Vector Machines, Gradient Boosting Machines, Deep Neural Networks such as ANN and mostly used Recurrent Neural Networks like Long-Short Term Memory (LSTM).

Support Vector Machines (SVM)

Support Vector Machines were introduced by (Cortes and Vapnik, 1995), not initially thought for time series analysis they can be extended to temporal data, they are a set of supervised learning methods.

The main idea is to find a hyperplane that separates the different classes in the case of classification, while in the case of regression that hyperplane minimizes the error.

The hyperplane in n -dimensional space is of dimension $n-1$, for example for a 2 dimension, the hyperplane is a straight line. Since is not always possible to linearly separate data, SVMs use something called the *kernel trick*, the kernel trick involves mapping the input data to a higher-dimensional space where it becomes linearly separable. There are several types of kernel functions, such as:

- Linear Kernel: $K(x, y) = x^T y$
- Polynomial Kernel: $K(x, y) = (x^T y + c)^d$, where c is a constant and d is the degree of the polynomial.
- Radial Basis Function (RBF) or Gaussian Kernel: $K(x, y) = \exp(-\gamma \|x - y\|^2)$, where γ is a positive parameter.

Random Forests

Random forests were introduced by (Breiman, 2001) and are an ensemble learning method that generates multiple decision trees and aggregates their predictions. In building different

trees on bootstrapped training samples but differently than in pure bagging, at each split it chooses a random percentage of predictors k among all the possible predictors p , reducing variance and thus making the model more stable and less susceptible to outliers.

Boosting and Gradient boosting machines

Like bagging boosting requires the combination of many decision trees, here as the name suggest is not to fit a single tree but instead make a slow learning, here instead of fitting trees on Y , they are fit on the *residuals*.

By fitting smaller trees to the residuals and updating the fitted function accordingly, the boosting algorithm adapts progressively, and this augments the robustness of the model.

The shrinkage parameter λ , which usually takes a value of 0.001 or 0.10, further slows down the learning process in boosting, enabling the algorithm to use more and diverse tree structures to tackle the residuals. Generally, slow-learning methods yield better performance. It's important to note that, in boosting, the construction of each tree is strongly dependent on the previously grown trees, unlike in bagging.

Gradient boosting is an advancement since it employs a gradient descent algorithm to minimize the loss when adding models to the ensemble. Instead of adjusting weights, subsequent models are fitted to *pseudo-residuals*, defined as the negative gradient of the loss function with respect to the model's predictions.

In other words, they indicate the direction and magnitude of changes needed to minimize the loss function.

With each iteration, the model improves by building an enhanced model that incorporates an estimator, denoted as h , to deliver superior performance. This optimum h implies:

$$F_{m+1}(x) = F_m(x) + h(x) = y$$

Where y is the target. Or simply:

$$h(x) = y - F_m(x)$$

This is an error between the target and the predictions from the previous iteration's model. Chadha (2020)

Hence, the formula for pseudo residuals depends on the specific loss function being used.

$$\eta_{j-1,t} = - \left. \frac{\partial L(y_t, f_{j-1}(x_t) + h)}{\partial h} \right|_{h=0}$$

Among the various gradient boosting methods, there are *XGBoost*, which uses stochastic gradient descent, *Light GBM*, and *CatBoost*.

Neural networks

Neural networks have proven also effective in dealing with temporal data, and among the various type of neural networks, some are suited for time series data more than others.

Feedforward Neural Networks shallow or deep for example have been used for time series forecasting but they may not be ideal for capturing long-term dependencies in the data.

Also, Convolutional Neural Networks have been used in forecasting time series data, this type of model is mostly used in computer vision, classification, and recognition, but also used in Sezer and Ozbayoglu (2018) use a Convolutional Neural Networks in algorithmic trading, creating a 15×15 image, with each image composed by using 15 financial indicators and 15 different intervals of these indicators. The images are labeled as “Hold”, “Buy”, or “Sell” using a sliding window logic, the images are composed respecting a specific order of the indicators.

Then the images are passed through a CNN for final classification.

Probably the most used type of neural network is represented by the family of Recurrent Neural Networks or RNN for short, specifically Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU).

LSTM is a robust model to approach time series data since they have a more sophisticated memory cell structure that enables them to learn and retain information from earlier time steps. This fact of taking in account past information when trying to forecast future outcomes makes the LSTM a good model.

This structure allows the LSTM to deal better than VAR models or ARIMA models in the case of non-stationary time series data. Figure 3.2 shows the LSTM architecture.

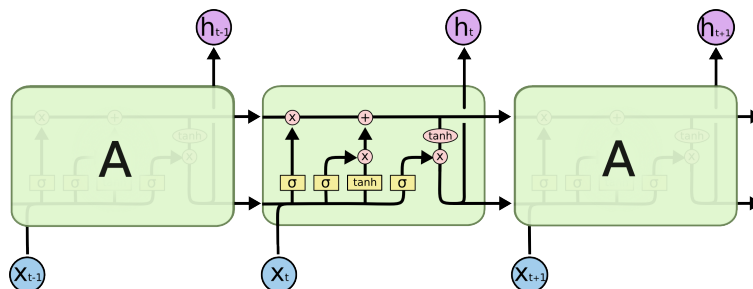


Figure 3.2: The LSTM chain(Olah, 2015)

Chapter 4

Evolutionary Algorithms

This chapter provides an in-depth introduction to the concept of EAs, and the fundamental principles that guide these algorithms, including the concepts of selection, mutation, and crossover. The chapter also discusses the various types of EAs, their unique characteristics, and their applications in different fields.

Evolutionary algorithm will be used to optimize the algorithms in the empirical section of this thesis

Evolutionary algorithms (EAs) are computational strategies that imitate aspects of natural evolution. Over time, they have evolved to incorporate a variety of search mechanisms, blending nature-inspired concepts with practical engineering needs. The basic function of all EAs is to sustain and iteratively refine a pool of potential solutions, a process akin to artificial 'evolution'. Among the various types of EAs, Genetic Algorithms (GAs), Genetic Programming (GP), and Evolution Strategies (ES) are some of the most recognized. These algorithms have shown remarkable success in real-world applications, particularly in addressing combinatorial problems. The adaptability of EAs is one of their most significant advantages. They can be customized to address any optimization problem without the need for simplification or reformulation, unlike other methods.

However, this adaptability also presents a challenge. Fine-tuning the configuration and parameters of an EA to achieve optimal performance for a specific set of tasks can be complex and time-consuming. This process of fine-tuning is a key area of ongoing research in the field of EAs.(Corne and Lones, 2018; Yu and Gen, 2010) In Figure 4.1 the various fields of application are shown.

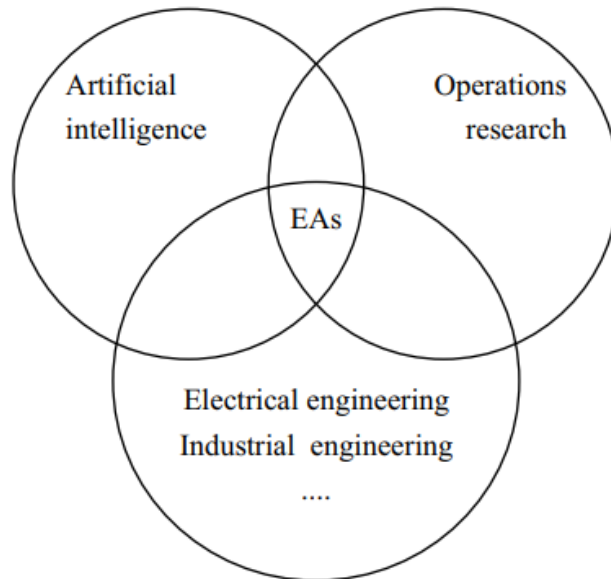


Figure 4.1: The various fields of application of EAs are shown(Olah, 2015)

The concept of bionics, which involves applying principles from nature to human-made systems, has led to numerous significant inventions. This approach has been used to create radar systems inspired by bats and submarines modeled after fish. Similarly, the process of natural evolution, which can be seen as a species learning to adapt and optimize its fitness, has been mimicked in the design of optimization and learning algorithms. This has led to the development of Evolutionary Algorithms (EAs).

EAs are unique in their ability to evolve while performing optimization or learning tasks. They possess three main characteristics:

- **Population-based:** EAs operate on an ensemble of solutions, known as a population, allowing them to optimize or learn about the problem in an alternative manner. This population-based approach is a foundation principle of the process of evolution
- **Geared Towards Fitness:** Every solution within a given population has a unique genetic representation (its own code) and a performance measure (its fitness score). The basis for algorithm optimization lies in favoring instances with superior fitness scores.
- **Variation-centric:** To emulate genetic changes, individual entities are subject to different variation procedures. This is a vital aspect in exploring the solution landscape.

(Yu and Gen, 2010) In essence, EAs represent a bionic approach to designing optimization and learning algorithms, drawing inspiration from the principles of natural evolution. (Yu and Gen, 2010)

4.1 Genetic Algorithms

Genetic Algorithms (GAs) are a group of Evolutionary Algorithms (EAs) that draw inspiration from the principles of genetics and natural selection. They are utilized to find approximate solutions to problems of optimization. The fundamental components of a GA include a population of chromosomes, genes, allele, fitness function, and operators such as selection, crossover (recombination), and mutation.

- **Population:** A GA operates on a population of potential solutions to the problem at hand. Each potential solution, also known as an individual or a chromosome, is encoded as a string of genes. These genes can be binary, real-valued, or represent more complex data structures, depending on the problem domain. The starting population is randomly selected.
- **Chromosome:** Each potential solution.
- **Gene:** A gene is one element position of a chromosome.
- **Allele:** The value for a gene takes on a certain solution (chromosome).
- **Fitness Function:** The fitness function evaluates each individual in the population to determine its quality or fitness. It quantifies the optimality of a solution in the problem domain, thus guiding the search for the best solution. The fitness of an individual is used to decide its likelihood of being selected for reproduction.
- **Selection:** The selection operator is used to select individuals from the current population to contribute to the next generation. Selection is typically biased towards individuals with higher fitness, mimicking the survival of the fittest principle in natural evolution.

In Figure 4.2 population, chromosomes, genes, alleles are depicted

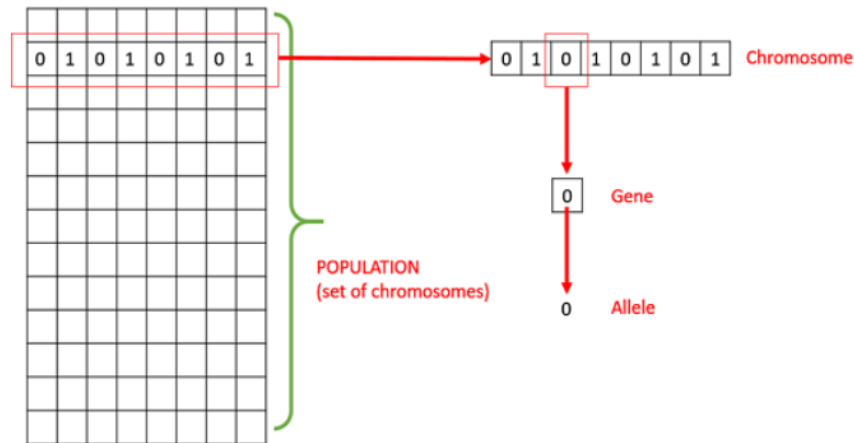


Figure 4.2: Population, Chromosome, Gene, and allele(Tutorialpoints, 2022)

Crossover (Recombination):

Crossover is a process utilized to amalgamate the genetic details of two progenitor entities to spawn new offspring. It promotes the exploration of new regions in the search space by creating individuals that are different from their parents, there are three primary methods of crossover which are also shown in Figure 4.3:

- **One-point crossover:** In this method, a random index of a chromosome is selected at each generation, and a swap of the solution between the parent chromosomes occur at this point. This results in two offspring that carry genetic information from both parents.
- **Two-point crossover:** This method involves the random selection of two indices of a chromosome at each generation. The genes located between these two indices are swapped between the parent chromosomes, resulting in two offspring that carry a mix of genetic information from both parents.
- **Uniform crossover:** In this method, each gene can either be swapped or remain in its current solution with a uniform probability at each generation. This is the only crossover method in which each gene is treated separately from the others.(Sobeih et al., 2010)

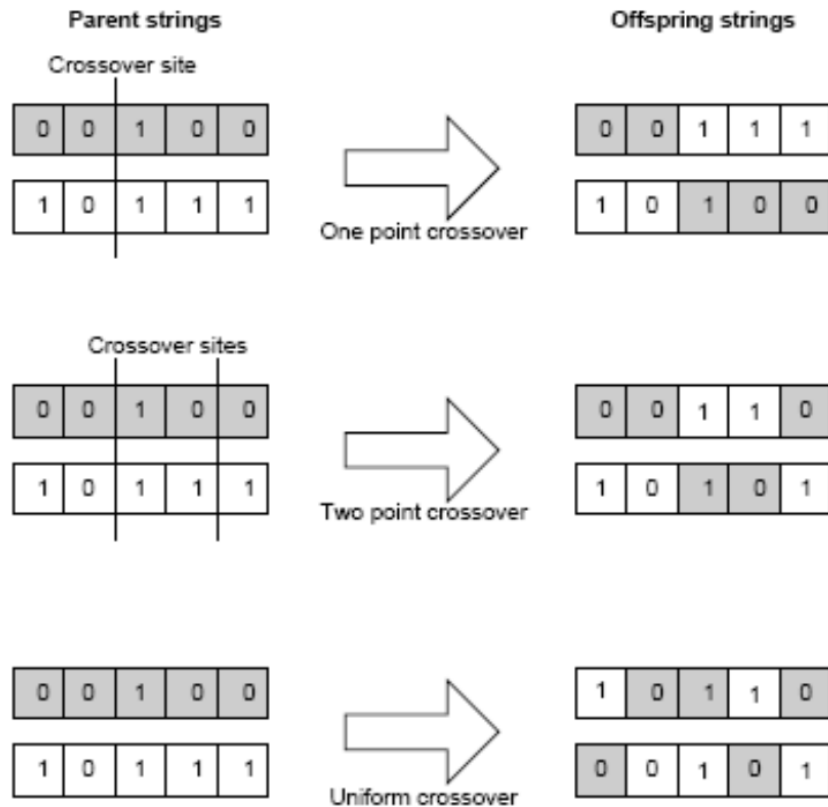


Figure 4.3: Different types of crossovers(Sobeih et al., 2010)

Mutation:

The mutation is used to introduce little random changes in the individual's genes. It ensures to maintain genetic diversity in the population and prevents premature convergence to sub-optimal solutions by allowing the exploration of new regions in the search space, mutation is also a solution to increase randomness. Figure 4.4 shows various types of mutation.

- **Adaptive mutation:** In this method, a subset of genes is selected at each generation, and their order is shuffled randomly.
- **Inversion mutation:** This method involves the inversion of the string of a subset of the genes at each generation.
- **Random mutation:** At each generation, the algorithm changes the value of a subset of genes in a random way, multiplying their value for a random value close to 1.
- **Swap mutation:** In this method, the algorithm interchanges the value of two different

randomly selected genes at each generation. (Liu, 2014)

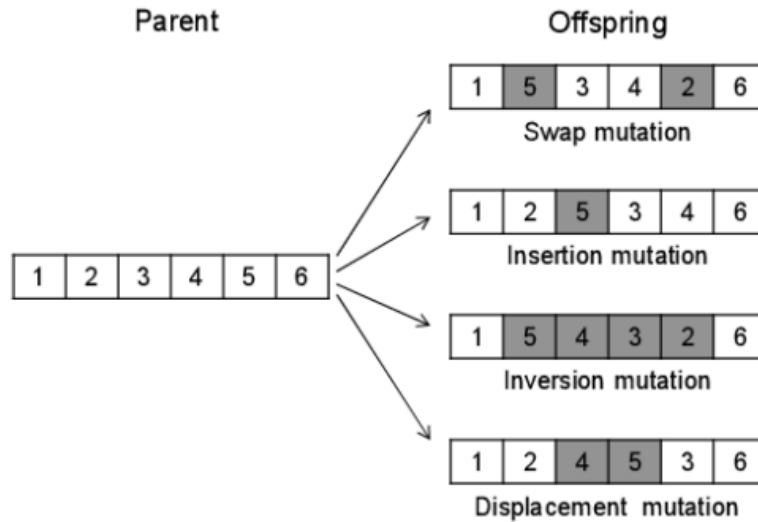


Figure 4.4: Different types of mutation(Liu, 2014)

Search termination:

The algorithm can terminate either if the difference between the maximum fitness function between two consecutive generations is less than an ϵ , if the number of generations exceeds a certain value, or if the computation time expires.

So, the GA begins with a randomly generated initial population. It then iteratively applies the selection, crossover, and mutation operators to create a new population. The performance or 'fitness' of each entity within the population is assessed, and this procedure continues until a predetermined termination criterion is fulfilled, such as hitting a specific generation count or achieving an acceptable level of fitness. (Melanie, 1996)

In conclusion, genetic algorithms provide an interesting solution to optimize algorithms in various fields but they are heavily dependent on the initial solution and implementing a genetic algorithm to a problem could be difficult due to the increase of computational resources required with the increment of generations or the size of the population taken in the exam.

Chapter 5

Stock Price Prediction using Evolutionary GAN

This chapter will describe step-by-step how the empirical part was carried out, including the details on the various software and hardware used, the description of the data science workflow implemented in this problem, the results and the discussion about them

5.1 Introduction

Generative Adversarial Networks are usually utilized for unsupervised tasks and developing GANs for regression introduces two major challenges: (1) inherent instability in the GAN and (2) performing regression and obtaining stability at the same time Nguyen et al. (2023). Since the setting in which neural networks operate is an infinite space, Nash (1950) said that infinite spaces need not have an equilibrium at all, and since the adversarial part is what differentiates GANs from other architectures the network may never reach the optimality. Hence, is interesting to investigate the use of these particular networks for regression problems, especially to time series data characterized by consecutive observations that could have different degrees of dependency on one with the other. Moreover, the addition of an evolutionary algorithm such as a genetic algorithm to optimize the performance of the network, especially with time series, is an innovative approach.

5.2 Related Work

Wang et al. (2018) introduced Evolutionary GAN that uses Genetic Algorithm to optimize the network, the author didn't focus on temporal data but instead on image data namely the CIFAR-10, LSUN bedroom, and CelebA, but the author didn't use mutation and crossover, so is not a genetic algorithm in the strict sense, the approach is shown in Figure 5.1

Bate (2022) have used genetic algorithm on time series for the same problem, but the author only focuses on the basic GAN, while this thesis will apply the optimization on 2 different architectures of Generative Adversarial Network, the Vanilla GAN and the Wasserstein GAN with Gradient Penalty.

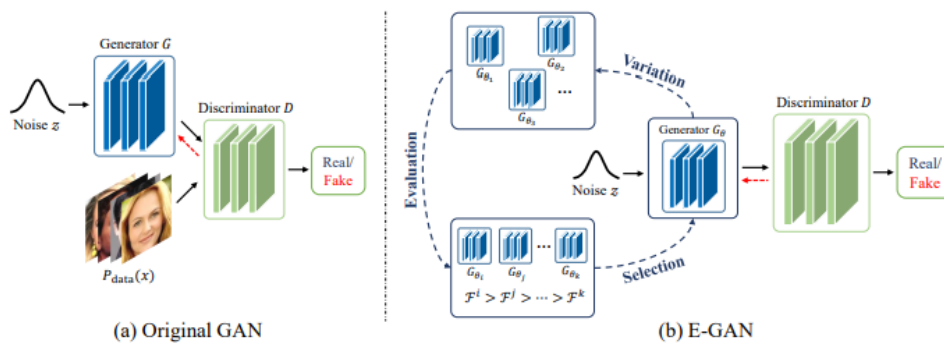


Figure 5.1: The Evolutionary GAN as written by(Wang et al., 2018)

The steps of this research are mainly inspired by the paper from Lin et al. (2021) *Stock Price Prediction using Generative Adversarial Networks*

5.3 Methodology

Materials and libraries used

All the operations were performed on Python, data pre-processing were made with Pandas and scikit-learn, the Fourier transform was made thanks to NumPy and the sentiment score was obtained thanks to FinBERT (Araci, 2019), the GAN implementation was performed with Tensorflow (Abadi et al., 2015). The entire script was run on the Colab platform leveraging the T4 GPU from Nvidia.

Data collection

The data were obtained from the Github Repository published by Hungchun Lin the author of the paper (Lin et al., 2021) <https://github.com/hungchun-lin/Stock-price-prediction-using-GAN>.

Dataset description

The dataset is composed of 2517 observations from 1st July 2010 to 24 June 2020, there are 37 features:

Open, High, Low, Close and Volume of Apple stock, then other columns where added: the price of NASDAQ, NYSE, S&P500, FTSE100, Nikki225, BSE SENSEX, RUSSELL2000, HENGSENG, SSE, CrudeOil, Gold, VIX, USD index, Amazon, Google, and Microsoft. Then some financial indicators were built on Apple stock: MA7, MA21, 20SD, MACD, upper, Lower, EMA, logmomentum.

Through Fourier transformation absolute of 3 comp, angle of 3 comp, absolute of 6 comp, angle of 6 comp, absolute of 9 comp, and angle of 9 comp were obtained, then a last column which is News.

Financial data were obtained through Yahoo Finance, the Dollar Index is downloaded from the FRED website, news data were obtained from Seeking Alpha and the sentiment score was computed as written before. The Fourier transform was performed with NumPy and is done because the sine waves produced by the transformation can approximate the original function and help the network to produce better outputs. Technical indicators were obtained with Pandas. Figure 5.2 shows the Fourier transformation

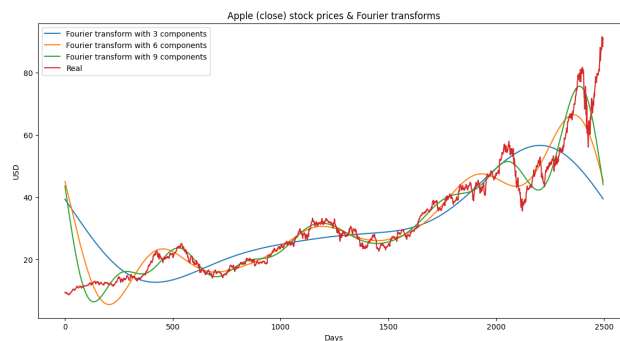


Figure 5.2: The relative Fourier Transformation, sine waves produced approximate the original price line in (*red line*), hence helping the model to represent the trend better

In Table 5.1 there is the description of the various variables

Feature Name	Feature Explanation	Feature Name	Feature Explanation
Open	Opening price in the trading day	Amazon	Amazon stock Price
High	Highest price in the trading day	Google	Google stock Price
Low	Lowest price in the trading day	Microsoft	Microsoft stock Price
Close	Closing price in the trading day	MA7	7-day simple moving average
Volume	Volume in the previous trading day	MA21	21-day simple moving average
NASDAQ	NASDAQ Index Closing Price	20SD	Bollinger bands mid-rail
NYSE	NYSE Composite Index Closing Price	MACD	Moving average convergence/divergence
S&P500	S&P 500 Index Close Price	Upper	Bollinger Band upper track
FTSE100	FTSE100 Index Close price	Lower	Bollinger Band lower track
Nikkei225	Nikkei Index Close Price	EMA	Exponential moving average
BSE SENSEX	BSE Sensitive Index Closing Price	logmomentum	Logarithmic momentum indicator
RUSSELL 2000	RUSSELL 2000 Index Closing Price	abs of 3 comp	3-order reconstruction(absolute)
HANG SENG	HK Hang Seng Index Closing Price	angle of 3 comp	3-order reconstruction(angle)
SSE	SSE Composite Index Closing Price	abs of 6 comp	6-order reconstruction(absolute)
CrudeOil	Crude Oil Closing Price	angle of 6 comp	6-order reconstruction(angle)
Gold	Gold Closing Price	abs of 9 comp	9-order reconstruction(absolute)
VIX	CBOE Volatility Index	angle of 9 comp	9-order reconstruction(angle)
USD index	The US dollar index	News	Sentiment value of financial news

Table 5.1: Explanation of Features

Data Preprocessing

First values equal to zero are transformed in null values and then forward and backward-filled, the date column in the dataset is converted to a DateTime format and set as the index of the dataset. The data is then rescaled using the MinMaxScaler from the sklearn library, which scales each feature individually such that it is in the given range, i.e between -1 and 1 in this case. Then data are transformed into sequence data, based on the number of input and output steps for the model, and creates the input and output datasets accordingly, in this case, the amount of input steps is equal to 30, while the number of output steps is equal to 3, namely we are trying to predict the price for the next 3 days considering the previous 30 days. Finally, data are split into training and testing sets with a ratio of 70/30.

Model outline

Four models have been developed:

- **Basic GAN with GRU (G) + CNN (D)**: This model starts with a GRU layer with 1024 units, followed by another GRU layer with 512 units, and a third GRU layer with 256 units. Each of these layers uses recurrent dropout for regularization. After these GRU layers, the model includes two dense layers, one with 128 units and the other with 64 units. The generator concludes with a final dense layer that matches the desired output dimension.

The discriminator, on the other hand, is a Convolutional Neural Network (CNN). It begins with a 1D convolutional layer with 32 filters, followed by another 1D convolutional layer with 64 filters, and a third 1D convolutional layer with 128 filters. Each of these layers uses the LeakyReLU activation function. After these convolutional layers, the model flattens the data and passes it through a dense layer with 220 units. This is followed by a LeakyReLU activation function and another dense layer with 220 units using a ReLU activation function. The discriminator concludes with a final dense layer with a single unit and a sigmoid activation function, which outputs the probability that the input data is real or fake. In Figure 5.3 the structure of this GAN is shown

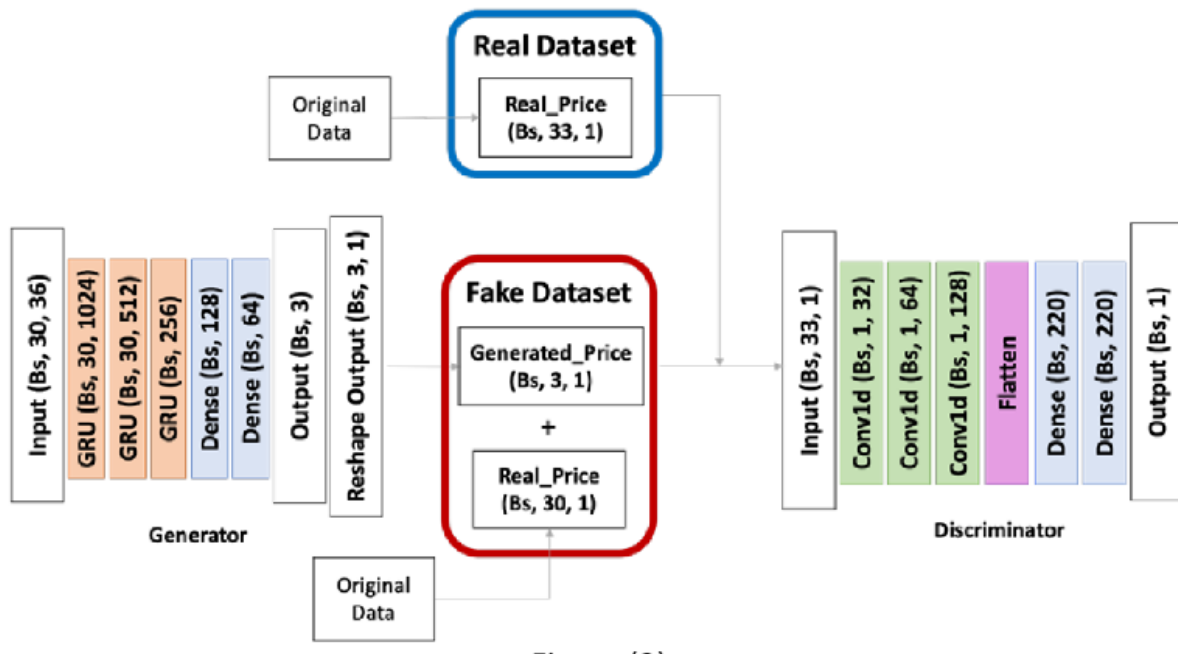


Figure 5.3: Here the real price is added to the output of the Generator to enhance data length and offer more context, the discriminator receives only prices as input and not all the other variables Lin et al. (2021)

- **Wasserstein GAN + GP with GRU (G) + CNN (D):** It starts with a GRU layer with 256 units, followed by another GRU layer with 128 units. Each of these layers uses recurrent dropout and L2 regularization for regularization. After these GRU layers, the model includes three dense layers with 64, 32, and the same units respectively, each with L2 regularization, and concludes with a final dense layer that matches the desired output dimension. The Discriminator has the same structure as the previous model, what is changed between this model and the other is that here the Discriminator is trained 3 times more than the Generator in this case, as for (Arjovsky et al., 2017). Moreover, here a gradient penalty is applied, first, the discriminator gradient is computed and then subtracted from 1, this difference is then squared to ensure it's always positive, and then it's multiplied by a penalty coefficient (a hyperparameter that you can adjust). This value is then added to the original loss function of the discriminator.

In practice, this means that if the discriminator's function has gradients that are too steep or too flat (i.e., not close to 1), it gets penalized, and its loss increases. This encourages the discriminator to have gradients of roughly 1, helping to satisfy the 1-Lipschitz condition and making the training process more stable and reliable. (Gulrajani

et al., 2017)

$$L = \underbrace{E_{\tilde{\mathbf{x}} \sim P_g} [D(\tilde{\mathbf{x}})] - E_{\mathbf{x} \sim P_r} [D(\mathbf{x})]}_{\text{Original critic loss}} + \lambda \underbrace{E_{\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}} [(\|\nabla_{\hat{\mathbf{x}}} D(\hat{\mathbf{x}})\|_2 - 1)^2]}_{\text{Our gradient penalty}}$$

- **Baseline GRU and Baseline LSTM:** The model is composed by two GRU layers, one with 128 units and another with 64 units. After the GRU layers, a dense (fully connected) layer with 32 units is added. This layer can help the model learn more complex representations. Finally, a dense layer with a number of units equal to the output dimension is added.

The model is compiled with the Adam optimizer, the other baseline model is a Long Short Term Memory with a Bidirectional LSTM as a first layer and two dense layers. These will be the two models that Lin et al. (2021) used as a baseline of its GANs.

Genetic Algorithm implementation

The genetic algorithm implemented here operates on the hyperparameters of the Generator and just on the learning rate of the Discriminator. The purpose of the genetic algorithm is to optimize these hyperparameters with respect to the Root Mean Square Error (RMSE) metric.

The algorithm starts by initializing a population of hyperparameters. Each individual in the population is a set of hyperparameters, defined by the Hyperparams class. The size of the population is set by the *popsiz*e parameter. The algorithm then proceeds over a certain number of generations.

For each generation, the algorithm assesses the fitness of each individual (set of hyperparameters) in the population. This is done by constructing the GAN model with the given hyperparameters, training it on the training data, and evaluating the model’s performance in terms of RMSPE, here the following hyperparameters are evaluated: the number of neurons for each layer, batch size, discriminator, and generator learning rate.

After assessing the fitness of the individuals, the algorithm selects a subset of the population to survive to the next generation. The top 20% of individuals, i.e., the individuals with the lowest RMSE, are chosen. The rest of the individuals are discarded.

The algorithm then replenishes the population by generating new individuals. Each new individual is created either by mutating an existing individual (survivor) or by crossing over two existing individuals. The choice between mutation and crossover is made randomly, with

a 50% chance of each.

Mutation involves selecting an individual from the population and modifying its hyperparameters. Each of the parameters is multiplied by a random factor between 0.9 and 1.1, effectively allowing the parameter to slightly decrease or increase. The new mutated parameters are then returned in a new Hyperparams object. This slight mutation allows the genetic algorithm to explore a localized area in the hyperparameter space around the parent individual. Crossover takes in hyperparameters from two-parent individuals and applies a crossover operation. It randomly selects each parameter from either parent1 or parent2, effectively creating a new set of hyperparameters that is a mix of the two parents. This new set of hyperparameters is returned in a new Hyperparam object. The crossover operation allows the genetic algorithm to combine the traits of two well-performing individuals in the hope of creating an even better-performing offspring.

The algorithm repeats this process for a number of generations specified by the generations parameter. In the end, it returns the hyperparameters of the individual with the lowest RMSE. In this case, the population size is set to 10 and the number of generations to 5, so the process of creating new individuals, selection, etc. will be performed 5 times. The number of epochs is set to 25, meaning that each training sample will be passed $25 * 5 * 10$ times or 1250 times. This low number was a tradeoff between the time available for the research and the computation capacity of the hardware available. In Figure 5.4 is process is summarized.

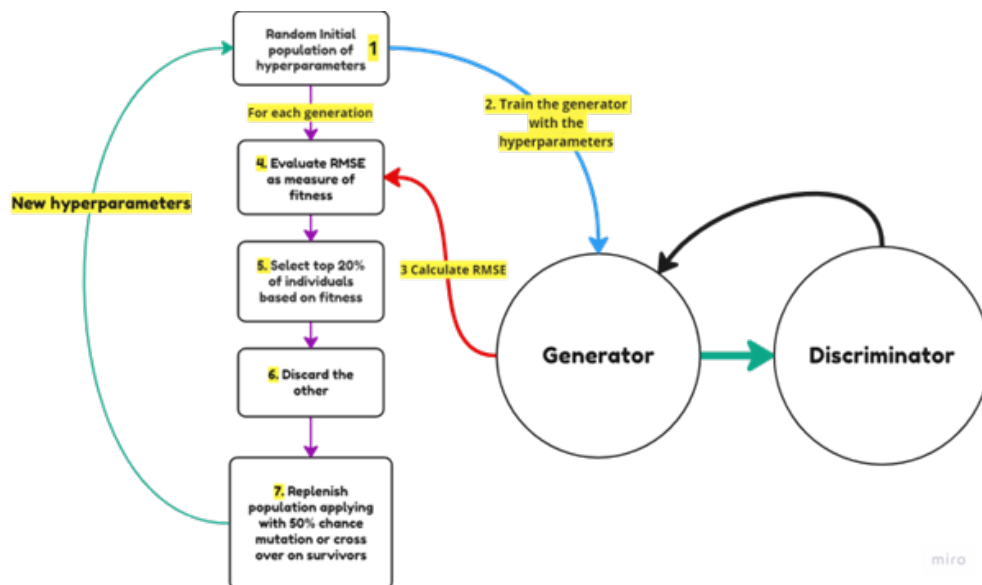


Figure 5.4: The Diagram of the proposed model

Evaluation metrics

Each GAN and baseline model is evaluated based on the RMSE:

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (x_i - \hat{x}_i)^2}{N}}$$

Other metrics used include the Jansen-Shannon Divergence that was introduced by Lin (1991) and has its fundamentals on the KL (Kullback-Leibler) Divergence that measures how one distribution p is far from a second expected distribution q . (Weng, 2019)

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

D_{KL} reaches the zero when $p(x) == q(x)$ everywhere.

So, Jensen-Shannon Divergence is another measure of similarity between two probability distributions $\in [0, 1]$. JS divergence is symmetric and smoother.

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}\left(p||\frac{p+q}{2}\right) + \frac{1}{2}D_{KL}\left(q||\frac{p+q}{2}\right)$$

Another tool often employed is t-Distributed Stochastic Neighbor Embedding (t-SNE), a popular method for reducing the dimensions and enhancing the visualization of datasets with high dimensionality. t-SNE accomplishes this by preserving the similarities between data points in a low-dimensional space, usually two or three dimensions, making it particularly useful in the evaluation of different distributions.(Van der Maaten and Hinton, 2008)

t-SNE aims to reduce the discrepancy between two key sets of data: one which analyzes the mutual characteristics of the original input entities, and the other that assesses mutual characteristics of the related points in a simpler, low-dimensional representation. This goal is accomplished by leveraging the principles of Kullback-Leibler (KL) divergence.(van der Maaten, 2014)

5.3.1 Training and results

Baseline LSTM

The model is run for 50 epochs and has a batch size of 64 and a learning rate of 0.001. The result of the test set is shown in Figures 5.5 and 5.6.

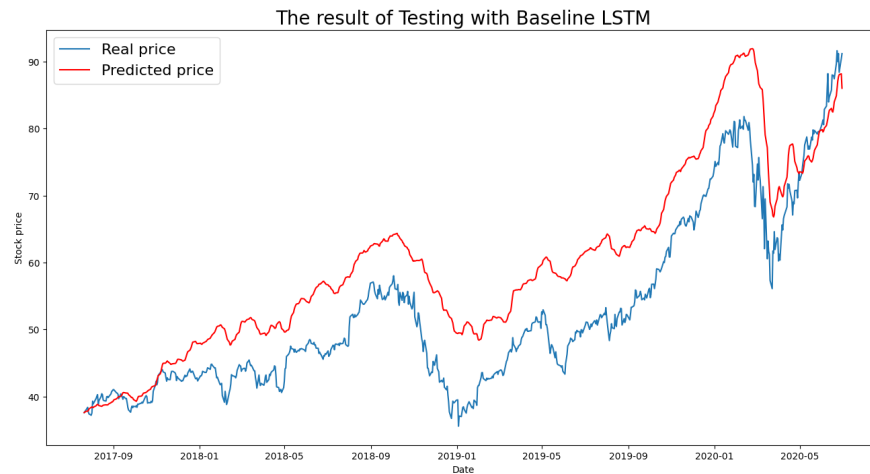


Figure 5.5: The result on the test set with the Baseline LSTM

It achieved a test RMSE of 8.34

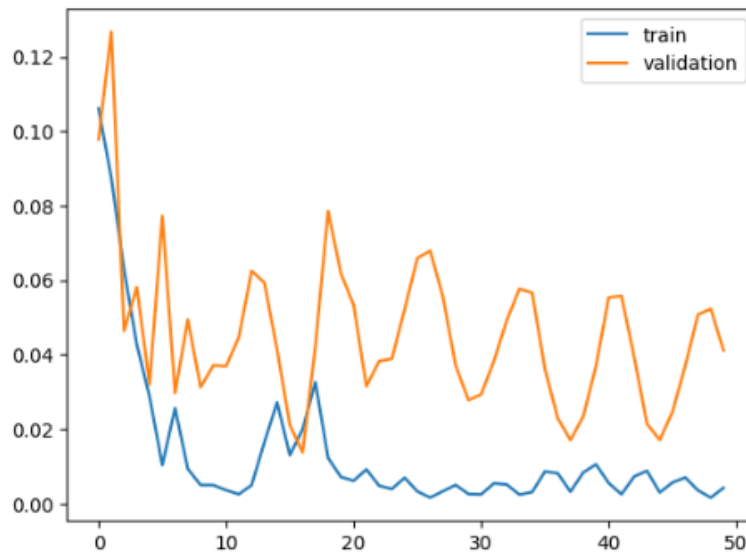


Figure 5.6: The plot of train and validation loss for LSTM

Baseline GRU

The model is run for 50 epochs and has a batch size of 128 and a learning rate of 0.0001. The result of the test set is shown in Figures 5.7 and 5.8.

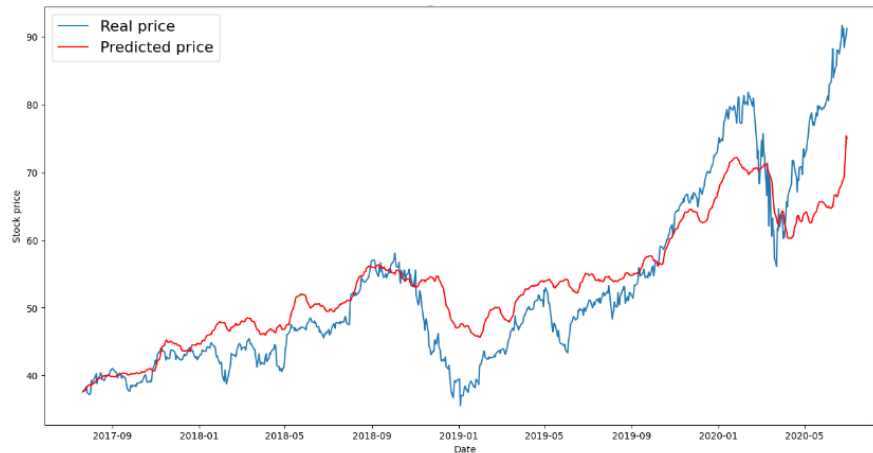


Figure 5.7: The result on the test set with the Baseline GRU

It achieved a test RMSE of 6.02

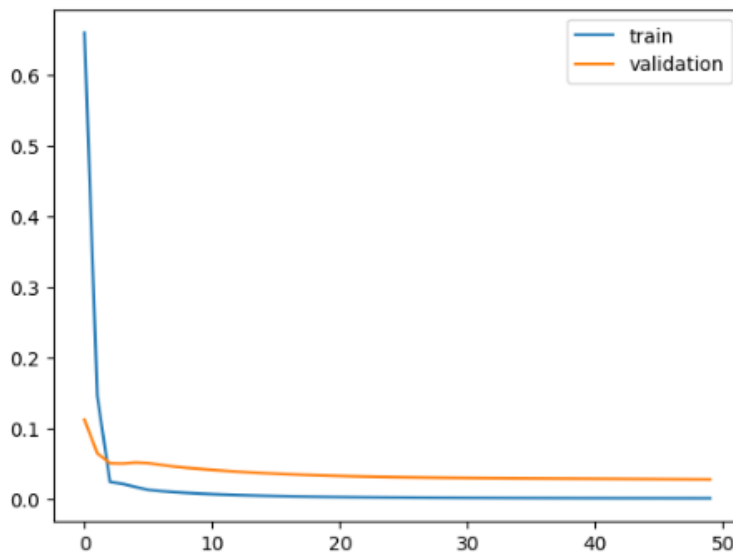


Figure 5.8: The plot of train and validation loss for GRU

Basic GAN

Here, the GAN has a 0.00016 learning rate for both discriminator and generator, the batch size is 128 and the model is trained for 165 epochs. In Figures 5.8 and 5.9 the predicted time series and the losses are shown.

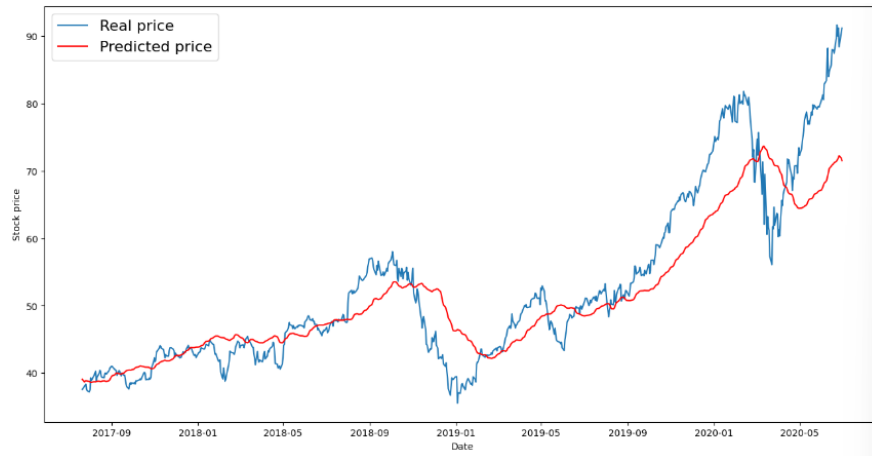


Figure 5.9: The result on the test set with the Baseline GRU

The model achieved a 5.82, beating the baseline

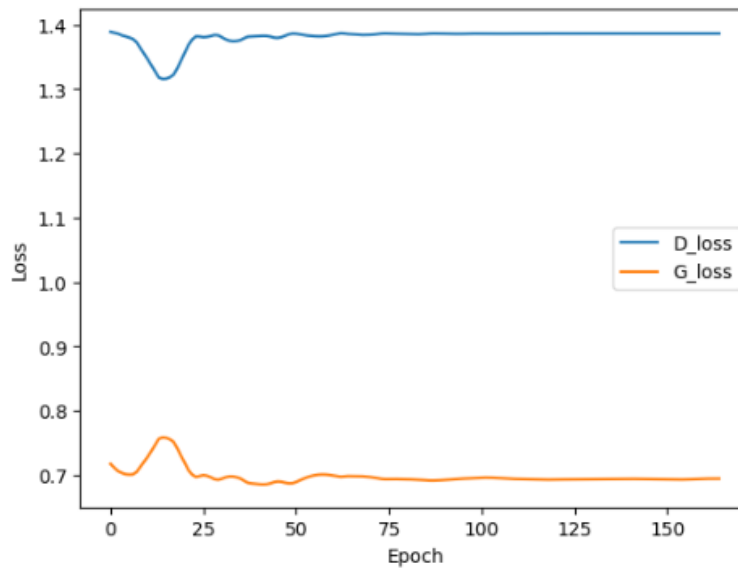


Figure 5.10: Plot of losses for the Discriminator and the Generator

Here the losses tend to stay flat over the entire number of iterations.

Optimization with Genetic Algorithm - Basic GAN

As explained in the subsection regarding the proposed model, the hyperparameters that are going to be examined are the neurons in the several layers of the Generator, except the last Dense layer, the batch size, the learning rate of both the Discriminator and the Generator. After the optimization process, the new hyperparameters are printed:

Hyperparameter	Original	Optimized
GRU 1 Units	1024	1152
Dense 1 Units	128	116
Batch Size	128	140
Discriminator LR	0.0001	0.0001
Generator LR	0.0001	0.0000916
GRU 2 Units	512	477
Dense 2 Units	64	57
GRU 3 Units	256	269

Table 5.2: Comparison of Original and Optimized Hyperparameters of Basic GAN

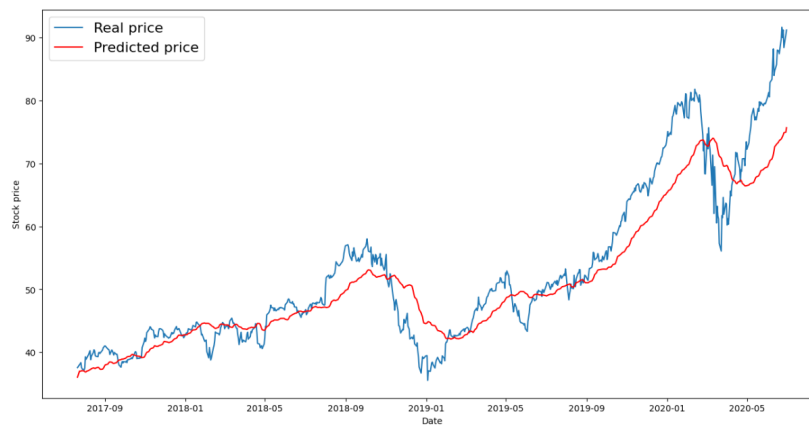


Figure 5.11: The result on the test set for the Basic GAN optimized with GA

Figure 5.11 shows the predicted vs. real price

The model achieved a 5.02 RMSE on the test set, with an improvement of 13% with respect to the non-optimized one, the loss plot behaves in the same way, from the tSNE plot it appears that the generated data are closer to the true distribution reflecting the lower RMSE as shown in Figure 5.12.

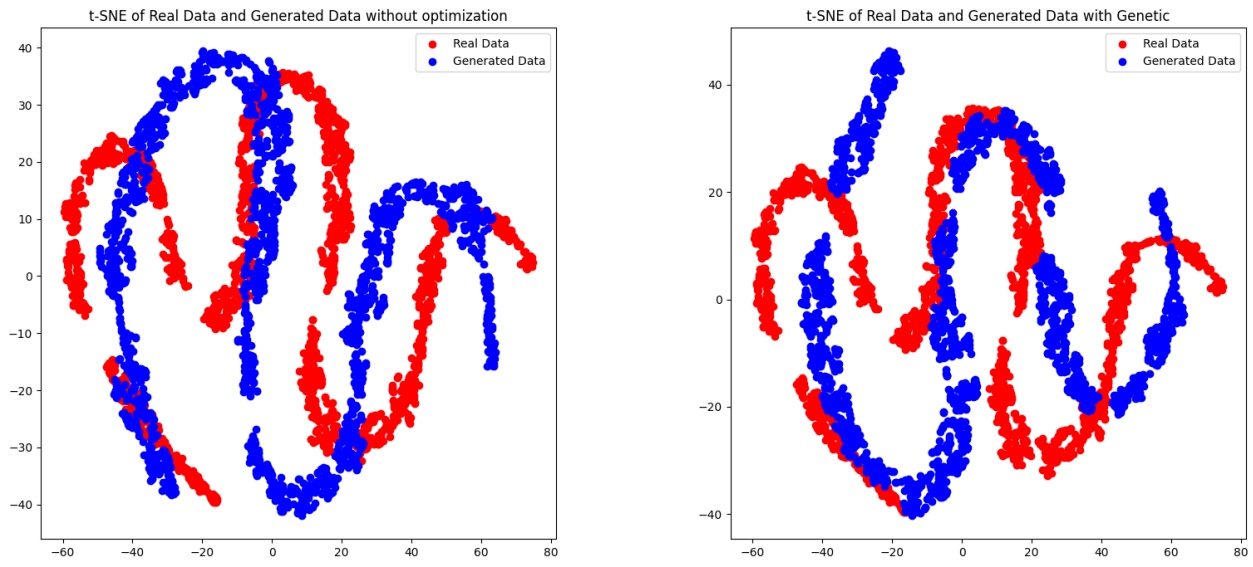


Figure 5.12: Left: the tSNE plot of the Basic GAN with no optimization, Right: the one optimized with the evolutionary approach

Wasserstein GAN with Gradient Penalty

Here, the WGAN GP has a 0.0001 learning rate for both the discriminator and generator, the batch size is 128, and the model is trained for 100 epochs. Here the Generator is trained 3 times more than the Discriminator. In Figures 5.13 and 5.14 the predicted time series and the losses are shown.

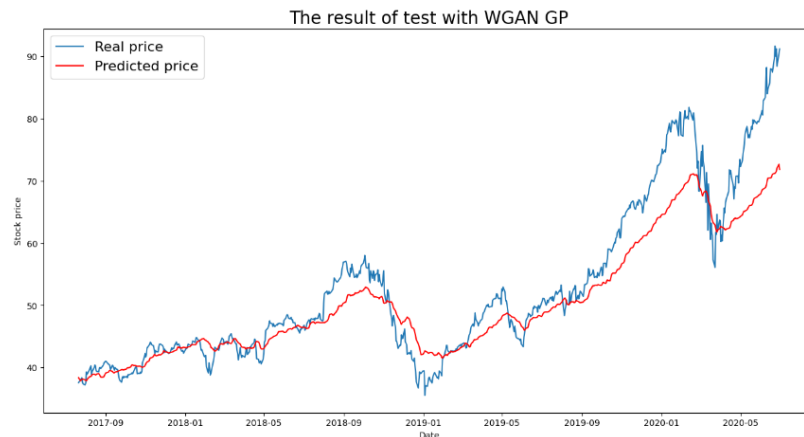


Figure 5.13: The result on the test set for the Wasserstein GAN with Gradient Penalty

The model achieved a 5.15, beating the baseline and the previous GAN

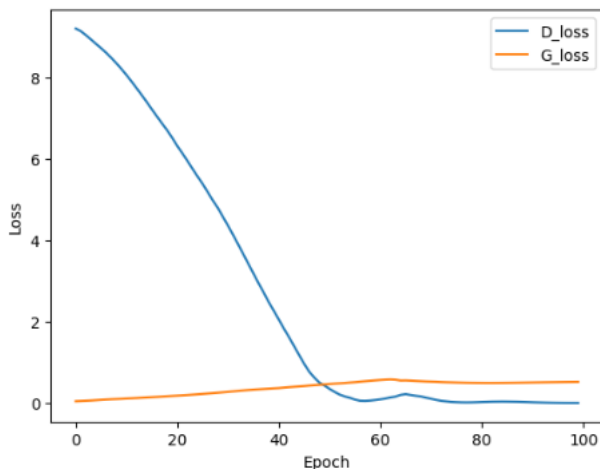


Figure 5.14: Plot of losses for the Discriminator and the Generator

Here the loss of the Discriminator collapses after 50 epochs, this is a totally different behavior with respect to the basic GAN.

Optimization with Genetic Algorithm - Wasserstein GAN GP

As explained in the subsection regarding the proposed model, the hyperparameters that are going to be examined are the neurons in the several layers, except the last Dense Layer, of the Generator, the batch size, the learning rate of both the Discriminator and the Generator. After the optimization process, the new hyperparameters are printed:

Hyperparameter	Original	Optimized
GRU 1 Units	256	255
Dense 1 Units	64	64
Batch Size	128	119
Discriminator LR	0.0001	0.000096
Generator LR	0.0001	0.0001
GRU 2 Units	128	138
Dense 2 Units	32	26

Table 5.3: Comparison of Original and Optimized Hyperparameters WGAN GP



Figure 5.15: The result on the test set for the Wasserstein GAN with Gradient Penalty optimized with GA

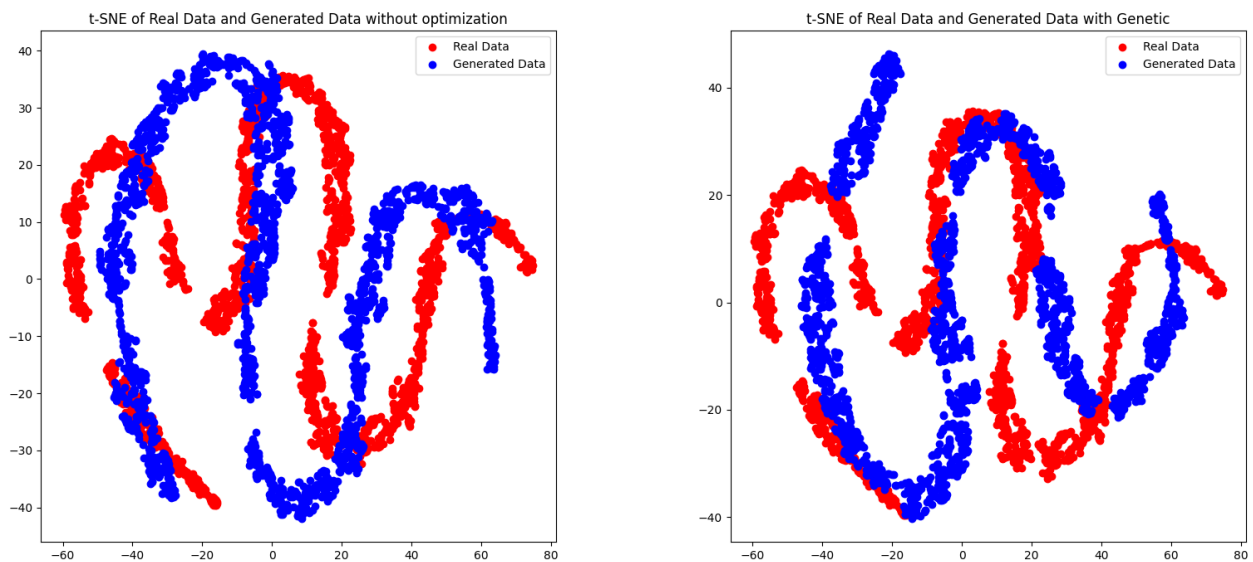


Figure 5.16: Left: the tSNE plot of the WGAN GP with no optimization, Right: the optimized one

Figure 5.15 shows the plot of the predicted and the real price.

The model achieved a 4.99 RMSE on the test set, with an improvement of 3% with respect to the non-optimized one, the loss plot behaves in the same way, also from these two tSNE plots in Figure 5.16 it can be observed how the distribution after the optimization seems to better overlap the real one.

Summary of results and discussion

To summarize, these are the final results both on the train and the test set.

Model	Train RMSE	Test RMSE
WGAN GP with GRU	2.36	5.15
WGAN GP with GRU + GA	2.13*	4.99*
Basic GAN GRU	1.61	5.82
Basic GAN GRU + GA	1.829	5.02*
Baseline GRU	1.055	6.02
Baseline LSTM	4.08	8.34

Table 5.4: Comparison of Models Performance

Metric	Basic GAN	Basic GAN + GA	WGAN GP	WGAN GP + GA
JS Divergence t_1	0.2264	0.1732*	0.2293	0.1449*
JS Divergence t_2	0.2585	0.2218*	0.2065	0.2275
JS Divergence t_3	0.1716	0.2199	0.1658	0.1887

Table 5.5: JS and KL Divergence of the various models, the asterisk underline where the optimization reduced one of the two metrics

The results confirm the findings from Lin et al. (2021), that WGAN with penalty generalizes better on unseen data on this time series, but what can be observed is that the proposed method improves the performance of the GAN on both train and test for the Wasserstein GAN. It can also be observed that the optimization with the Genetic Algorithm drastically reduced the JS Divergence for the next day's closing price, while it didn't have a significant effect on t_2, t_3, it even significantly increases on t_3 for the optimization on Wasserstein with Gradient Penalty. This result can mean that a model that tries to predict the next day's observation can benefit more from this type of optimization. Hence genetic algorithms can be a solution for the optimization of GANs applied to the regression of time series data.

Chapter 6

Conclusion and further developments

The present study has explored the potential and efficacy of Generative Adversarial Networks (GANs) in predicting stock prices, further enhanced by the application of genetic algorithms. With the aim of improving the performance of GANs, the findings point to an interesting and promising direction for enhancing machine learning models for financial market predictions. The results are significant for the computational finance field, highlighting how AI and machine learning techniques can be harnessed to model and predict complex and volatile financial markets.

Genetic algorithms (GAs) have proven to be effective in enhancing GANs' performance. This approach is grounded in the principles of natural selection and evolution, encouraging the development of optimal GAN models by iterating through generations. The research implemented GAs for five generations, with an iteration limit of 25 epochs, which yielded notable improvements in prediction accuracy and stability.

However, the application of GAs, albeit successful, is not the solitary path to optimization. There are still other evolutionary computation techniques that might be beneficial. For instance, Particle Swarm Optimization (PSO) is an evolutionary algorithm that could potentially enhance the GANs' performance. Its unique trait of emphasizing social interaction within the swarm for determining the best solution could be a worthwhile exploration, considering the swarm's interdependencies could mirror market complexities and dynamics. (Kennedy and Eberhart, 1995)

Another promising development in GAN optimization is the implementation of fuzzy logic. Nguyen's 2023 research on FuzzyGAN proposed the integration of a differentiable fuzzy logic system in a GAN, which was reported to improve its regression capabilities. The fuzzy logic system uses the output of either the generator or the discriminator to predict output and

evaluate the generator's performance. Applying this concept to the current model could enhance its ability to handle uncertainty and noise, common characteristics in financial time series data.

While the present study has successfully applied GAs to improve GANs performance in predicting stock prices, several limitations arose. The computational resources available placed constraints on the number of generations and epochs that could be executed. Increased computational resources might allow for a more extensive exploration of the model's genetic and evolutionary parameters, potentially yielding even greater accuracy and robustness.

Moreover, the present study focused on a specific financial time series, which might limit the generalizability of the results. Future work could involve applying the model to different time series with varying characteristics, such as volatility and the Hurst exponent. This will offer a broader perspective on the applicability and adaptability of the model, further establishing its robustness in varying market scenarios.(Qian and Rasheed, 2004)

Lastly, there is a promising avenue in hybridizing fuzzy logic and evolutionary algorithms in optimizing GANs. Fuzzy logic could enhance the model's ability to handle uncertainty and noise, while evolutionary algorithms could improve the model's adaptability and robustness. The fusion of these techniques could possibly lead to a high-performing GAN model capable of accurate and robust financial market predictions.

In conclusion, the current study underscores the remarkable potential of GANs in financial forecasting, demonstrating their ability to be improved by genetic algorithms. As we advance, it is evident that harnessing the power of other optimization techniques such as PSO, Fuzzy Logic, and possibly, a hybrid approach, offers a fascinating direction for future research. Undoubtedly, the application and continuous refinement of these computational techniques will shape the future of financial forecasting, aiding in the mitigation of risks and harnessing potential economic opportunities.

Bibliography

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.
- Amidi, A. and Amidi, S. (2023). Supervised learning cheatsheet.
- Araci, D. (2019). Finbert: Financial sentiment analysis with pre-trained language models.
- Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan. In *International conference on machine learning*, pages 214–223. PMLR.
- Banerjee, A. (2007). An analysis of logistic models: Exponential family connections and online performance. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, pages 204–215. SIAM.
- Bank, D., Koenigstein, N., and Giryas, R. (2020). Autoencoders.
- Bate, H. (2022). Genetic algorithm-based optimization of generative adversarial networks and its applications.
- Bauer, A. (2021). *Automated Hybrid Time Series Forecasting: Design, Benchmarking, and Use Cases*. PhD thesis.
- Bengio, Y., Yao, L., Alain, G., and Vincent, P. (2012). Generalized denoising auto-encoders as generative models. *Journal of Machine Learning Research*, 13(1):223–268.
- Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

- Cerqueira, V., Torgo, L., and Soares, C. (2019). Machine learning vs statistical methods for time series forecasting: Size matters.
- Chadha, C. (2020). Bagging, Boosting, and Gradient Boosting. <https://towardsdatascience.com/bagging-boosting-and-gradient-boosting-1a8f135a5f4e>.
- Corne, D. and Lones, M. A. (2018). *Evolutionary Algorithms*, pages 1–22. Springer International Publishing, Cham.
- Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20:273–297.
- Daniel Jurafsky, J. H. M. (2023). Hidden markov models.
- Dietterich, T. G. and Kong, E. B. (1995). Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Citeseer.
- Elwes, J. (2021). Latent space.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, 27:2672–2680.
- Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. (2017). Improved training of wasserstein gans. *CoRR*, abs/1704.00028.
- Jordan, J. (2018). Variational autoencoders.
- Kahn, J. (2023). Which bank is furthest ahead in a.i.? a new index will tell you.
- Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4401–4410.
- Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE.
- Lin, H., Chen, C., Huang, G., and Jafari, A. (2021). Stock price prediction using generative adversarial networks. *Journal of Computer Science*, 17(3):188–196.
- Lin, J. (1991). Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151.

- Liu, C. (2014). *Multi-Robot Task Allocation for Inspection Problems with Cooperative Tasks Using Hybrid Genetic Algorithms*.
- Melanie, M. (1996). An introduction to genetic algorithms.
- Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49.
- Nguyen, R., Singh, S. K., and Rai, R. (2023). Fuzzygan: Fuzzy generative adversarial networks for regression tasks. *Neurocomputing*.
- Olah, C. (2015). Understanding LSTM Networks. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. [Online; accessed 13-April-2023].
- Qian, B. and Rasheed, K. (2004). Hurst exponent and financial market predictability. In *IASTED conference on Financial Engineering and Applications*, pages 203–209. Proceedings of the IASTED International Conference Cambridge, MA.
- Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16.
- Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International conference on learning representations*.
- Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.
- Rocca, J. (2019). Understanding variational autoencoders (vae).
- Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229.
- Sezer, O. B. and Ozbayoglu, A. M. (2018). Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70:525–538.
- Sobeih, M., Doma, M., and Al Shouny, A. (2010). Mixture-order design of gps networks based on genetic algorithms. *ERJ. Engineering Research Journal*, 33:431–439.
- Tutorialpoints (2022). Genetic algorithms - quick guide.

- van der Maaten, L. (2014). Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15(93):3221–3245.
- Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.
- Wang, C., Xu, C., Yao, X., and Tao, D. (2018). Evolutionary generative adversarial networks. *CoRR*, abs/1803.00657.
- Weng, L. (2019). From GAN to WGAN. *CoRR*, abs/1904.08994.
- Wolpert, D. H. (1996). The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390.
- Yoon, J., Jarrett, D., and Van der Schaar, M. (2019). Time-series generative adversarial networks. *Advances in neural information processing systems*, 32.
- Yu, X. and Gen, M. (2010). *Introduction to evolutionary algorithms*. Springer Science & Business Media.
- Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232.

Final Summary

The objective of this thesis is to expand the literature on applying Generative Adversarial Networks on regression problems and especially financial time series with the utilization of genetic algorithms to optimize the architecture of the GAN and improve the performances. This thesis is organized into two main parts, the first one introduces the concept of generative AI models and the models that belong to this class, it will continue with an exam on the characteristics of time series data and the most common techniques utilized to forecast and analyze them, then this part will end with a general introduction to genetic algorithms and the main features that characterize them. The second part of this thesis is dedicated to the empirical part of this research that has applied 6 different models to the stock price forecasting, with two baseline models, two different types of GANs and finally the optimization on these two. As said the first part of my thesis explores the intricate and innovative field of Generative Adversarial Networks (GANs), a pioneering contribution to deep learning unveiled by Goodfellow. in 2014. Taking inspiration from game theory, GANs utilize a unique dual neural network architecture, featuring a generator and a discriminator. These two entities operate on the adversarial modeling framework and are characterized as multilayer perceptrons, signifying their advanced nature and capacity for complex calculations.

The dynamic between the generator and the discriminator can be thought of as an exciting adversarial interaction. The generator's role resembles that of a forger, endeavoring to produce data so convincing that it dupes the discriminator into acknowledging it as genuine. The discriminator, meanwhile, is a vigilant inspector that learns to differentiate between samples derived from the model distribution and the real-world data distribution.

Training is an evolutionary journey for both entities. As they improve and refine their capabilities, there comes a point where the discriminator can no longer distinguish between the actual data and the counterfeit data created by the generator. This signifies a significant breakthrough, highlighting the power and potential of this class of models.

The GAN model pivots around the concept of input noise variables, symbolized as $p_z(z)$.

These variables form the basis for the generator's data distribution, embodying a mapping to data space depicted as $G(z; \theta g)$. In this representation, G stands for a differentiable function, while θg constitutes its parameters.

Parallely, a secondary multilayer perceptron is defined as $D(x; \theta d)$, wherein θd is the controlling parameter. This definition represents the discriminator, producing a single scalar. The discriminator's primary objective during training is to maximize the probability of correctly labeling both generated and real data samples.

The interaction between the generator and the discriminator manifests as a two-player minimax game, with the value function $V(G, D)$ depicting the relationship between the two. The game reaches an optimum state when $p(z)$ equals $p(\text{data})$, as demonstrated by Equation 1.

The training process, as shown in Algorithm 1, is delicate and intricate, requiring careful balance. The discriminator's performance directly impacts the learning potential. If the discriminator is too good, it results in a vanishing gradient, hindering the learning process. Conversely, a poorly performing discriminator provides insufficient feedback for the generator, disrupting its learning.

GANs, fundamentally, are extensions of the concept of undirected graphical models with latent variables. This broader category includes models such as Restricted Boltzmann Machines (RBMs) and Deep Boltzmann Machines (DBMs). These models use a probabilistic approach to model the distribution of data, with a focus on maximum likelihood estimation (MLE).

The unique aspect of GANs is their discriminative approach to training the generator, which diverges from the generative approach employed by RBMs and DBMs to learn the underlying data distribution. Despite their higher complexity and potential challenges during training, GANs have demonstrated remarkable efficacy in generating realistic data.

However, GANs are not without their challenges. These include issues like low dimensionality, where the real data's dimension appears deceptively high but is actually confined to a low-dimensional space. Mode collapse, a condition where the generator consistently outputs identical or near-identical samples, also threatens the diversity and richness of the generated data.

The inception of the Wasserstein GAN aimed to address these challenges. This model tweaks the standard GAN setup by replacing the Jensen-Shannon Divergence with the Wasserstein Distance, also known as the Earth Mover's Distance. This change increases stability and robustness during training, effectively reducing the incidence of problematic issues.

Several GAN variants have emerged over the years, expanding the potential applications of this framework. These include the Deep Convolutional GAN (DCGAN), CycleGAN (CGAN), StyleGAN, and TimeGAN. Each of these models brings unique features and capabilities to the table, continually pushing the envelope of what GANs can achieve.

The advent and ongoing refinement of Generative Adversarial Networks signify a landmark development in the field of deep learning. They not only demonstrate the extent to which neural networks can mimic and reproduce complex data but also open up new avenues for generating synthetic data for a wide range of applications.

The first part continues with Chapter 3 delves into the intricate realm of time series data, a distinct category of data that is collected in a sequential manner over time, either at regular or irregular intervals. Time series data stands out from other types of data due to its temporal nature, representing observations recorded across a continuum of time. This specific form of data analysis is an essential aspect of numerous domains, ranging from financial markets to meteorology, where it is used to detect patterns, spot anomalies, predict trends, and inform decision-making processes.

Each data point within a time series is intrinsically associated with a timestamp that can vary in granularity - from seconds to days, weeks, or even longer. The order of these data points, which might seem inconsequential in other data types, is vital when dealing with time series, as it informs our understanding of the temporal dynamics and dependencies at play.

Many time series datasets exhibit repeating patterns or cycles over fixed periods, referred to as 'seasonality'. For instance, data related to agricultural production, energy consumption, and holiday seasons often demonstrate clear seasonal trends that repeat annually. On the other hand, time series data may also reveal long-term increases or decreases in the mean level, a characteristic known as 'trends'. These trends can be either upward or downward and may persist over various time spans.

Time series data often contain random variations, measurement errors, or unexplained fluctuations, which are collectively termed 'noise'. Despite their random nature, these noise components are crucial to understanding the intricate dynamics of time series data.

A central aspect of time series analysis is the concept of 'stationarity'. A time series is considered stationary if its mean and variance, don't change over time, implying the absence of trends or seasonality patterns. Stationarity is often achieved using differentiation techniques, thereby helping certain time series analysis methods to function effectively.

Moreover, time series data often exhibit 'autocorrelation', where successive observations are correlated. This property can help identify periodicities, seasonality, and persistence in the data, informing appropriate models for forecasting or anomaly detection.

The analysis of time series data and its unique characteristics can generate valuable insights, guiding informed decisions across diverse fields. To this end, there exists a multitude of models and techniques, ranging from statistical to machine learning methods, that can be applied to time series data.

The traditional methods of time series analysis encompass statistical models such as autoregressive (AR) models, moving average (MA) models, autoregressive moving average (ARMA) models, and autoregressive integrated moving average (ARIMA) models. These models capture linear relationships between successive observations and typically function well for stationary time series data.

In addition to statistical models, there has been a growing interest in the application of machine learning and artificial intelligence to time series analysis. This includes algorithms like Support Vector Machines (SVMs), Random Forests, and Gradient Boosting Machines, as well as Deep Neural Networks such as Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks.

Each of these models and techniques has its unique strengths and applicability, making them suited to address different challenges presented by time series data. However, the selection of an appropriate model for a specific task still remains a heuristic process, as it largely depends on the nature of the data, the problem at hand, and the computational resources available.

In conclusion, this chapter has provided an overview of time series data and the unique complexities it presents. From discussing its inherent characteristics to outlining various statistical and machine learning techniques used for its analysis, the chapter elucidates the significance of time series analysis in extracting meaningful insights and informing decision-making processes across a multitude of domains. While the choice of the right model or technique is still a matter of ongoing research and debate, the existing array of methods offers a promising toolset for handling the unique challenges posed by time series data.

Chapter 4 continues with a description of what are Evolutionary algorithms (EAs) that constitutes a unique class of computational methods that emulate natural evolution, creating powerful problem-solving strategies. These algorithms have continued to evolve, integrating diverse search techniques that amalgamate nature-inspired ideas with engineering requirements. They are designed to maintain and progressively refine a collection of possible solutions, a process akin to artificial evolution.

Among the assortment of EAs, Genetic Algorithms (GAs), Genetic Programming (GP), and Evolution Strategies (ES) have garnered the most recognition. These algorithms have demonstrated exceptional success in real-world applications, particularly in dealing with com-

binatorial problems. A defining feature of EAs is their adaptability. They can be customized to tackle any optimization problem, without necessitating simplification or reformulation, in contrast to other methods. However, this adaptability also presents challenges. Fine-tuning the configuration and parameters of an EA to optimize performance for a specific task can be complex and time-consuming. This fine-tuning process is a crucial area of ongoing research in the EA domain.

Bionics is a concept that applies principles from nature to human-made systems, leading to several significant inventions. This approach has been used to create radar systems inspired by bats and submarines modeled after fish. Similarly, the process of natural evolution, seen as species learning to adapt and optimize their fitness, has been mirrored in the design of optimization and learning algorithms, leading to the development of Evolutionary Algorithms (EAs).

EAs have three distinct characteristics: they are population-based, fitness-oriented, and variation-driven. EAs operate on a group of solutions, a population, allowing them to optimize or learn about the problem in a parallel manner. Each solution in a population, known as an individual, has its gene representation and performance evaluation. EAs favor individuals with higher fitness values, which forms the basis for the optimization and convergence of the algorithms. Individuals undergo various variation operations to mimic changes in genetic genes, which is vital for exploring the solution space.

Genetic Algorithms (GAs), a subset of EAs, take inspiration from the principles of natural selection and genetics. They are used to find approximate solutions to optimization and search problems. GAs operate on a population of potential solutions, with each potential solution, or individual, encoded as a string of genes. These genes can be binary, real-valued, or represent more complex data structures, depending on the problem domain. The initial population is usually randomly selected.

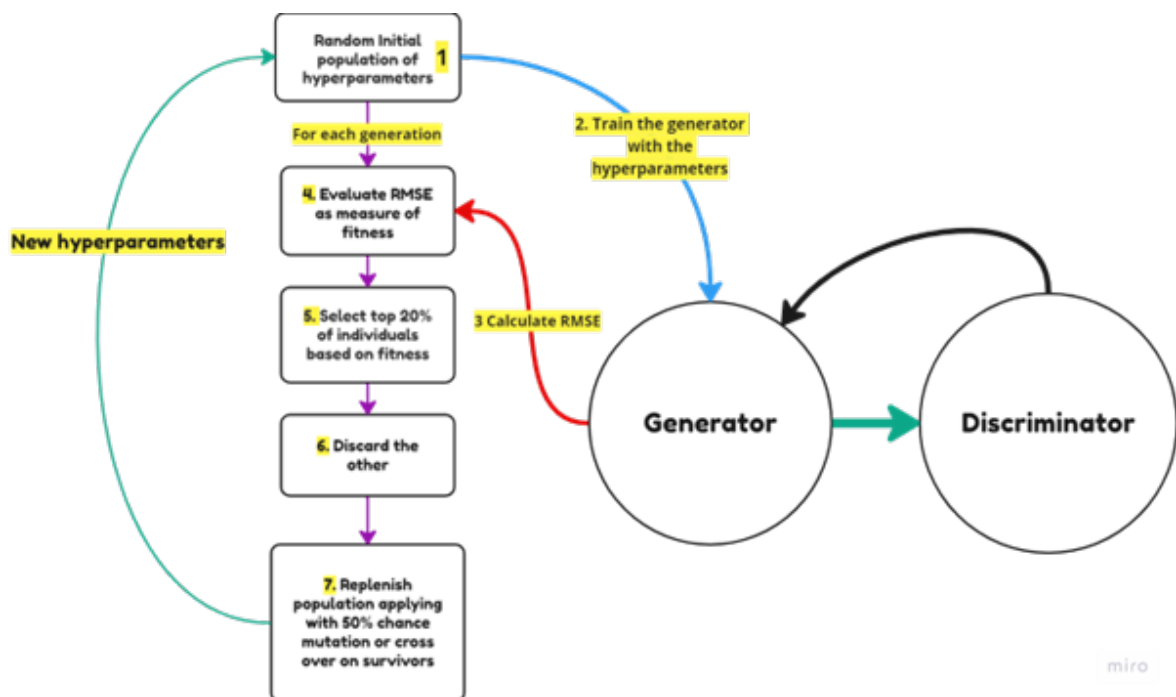
GAs have a fitness function that evaluates each individual in the population to determine its quality or fitness. This fitness quantifies the optimality of a solution in the problem domain, thus guiding the search for the best solution. The fitness of an individual is used to decide its likelihood of being selected for reproduction.

GAs employ operators like selection, crossover (recombination), and mutation. The selection operator selects individuals from the current population to contribute to the next generation. Selection is usually biased towards individuals with higher fitness, mirroring the survival of the fittest principle in natural evolution. The crossover operator combines the genetic information of two parents to generate new offspring, promoting the exploration of new regions in the search space. The mutation operator introduces small random changes in the

individuals' genes, maintaining genetic diversity in the population and preventing premature convergence to suboptimal solutions.

GAs start with a randomly generated initial population, then apply selection, crossover, and mutation operators iteratively to create a new population. The fitness of the individuals in the population is evaluated, and the process continues until a termination condition is met, such as reaching a maximum number of generations or achieving a satisfactory fitness level.

In conclusion, while genetic algorithms provide an intriguing solution for optimizing algorithms in various fields, they do have their limitations. They heavily rely on the initial solution and implementing a genetic algorithm to a problem could be challenging due to the increase in computational resources required with the increment of generations or the size of the population examined. This is the diagram of the genetic algorithm utilized in the empirical part of this thesis that is contained in Chapter 5.



The Diagram of the proposed model

The second part of this thesis contains the steps that were performed to answer to the research question:

Can the performances of a Generative Adversarial Network be improved by Genetic Algorithms and better predict the stock price?

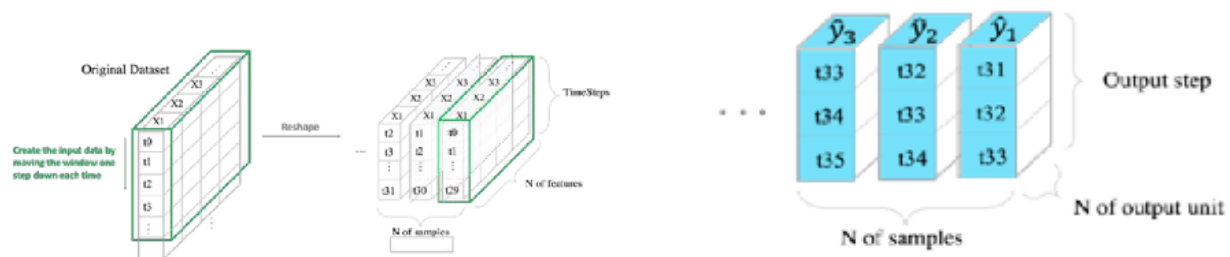
The dataset is composed of 2517 observations from 1st July 2010 to 24 June 2020, there are 37 features:

Open, High, Low, Close and Volume of Apple stock, then other columns were added: the price of NASDAQ, NYSE, S&P500, FTSE100, Nikkei225, BSE SENSEX, RUSSELL2000, HENGSENG, SSE, CrudeOil, Gold, VIX, USD index, Amazon, Google, and Microsoft. Then some financial indicators were built on Apple stock: MA7, MA21, 20SD, MACD, upper, Lower, EMA, logmomentum.

Through Fourier transformation absolute of 3 comp, angle of 3 comp, absolute of 6 comp, angle of 6 comp, absolute of 9 comp, and angle of 9 comp were obtained, then a last column which is News.

The prices regarding the financial indices, stocks, and commodities were obtained through Yahoo Finance, the Dollar Index is downloaded from the FRED website, news data were obtained from Seeking Alpha and the sentiment score was computed as written before. The Fourier transform was performed with NumPy and is done because the sine waves produced by the transformation can approximate the original function and help the network to produce better outputs.

Data were preprocessed, first normalizing them between -1 and 1 and then transformed into sequence data, specifically the previous 30 observations are taken into consideration by the model to forecast the next 3 days



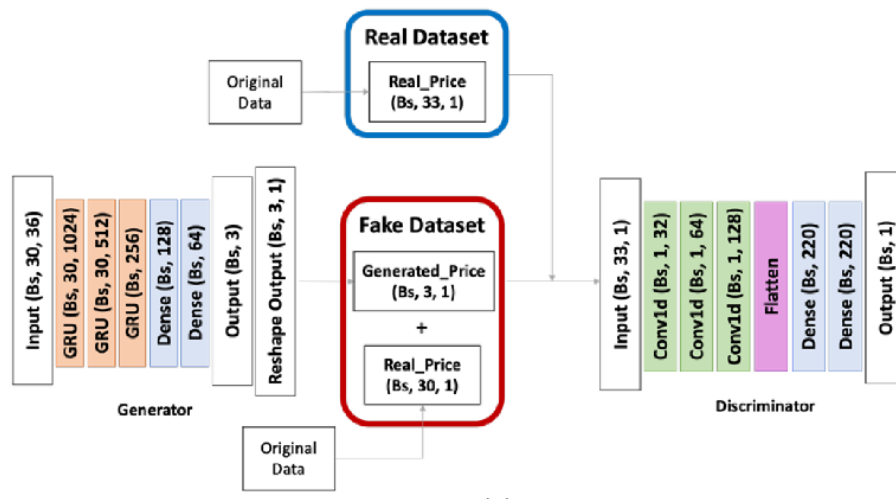
Left: the input step, previous 30 days. Right: the output step, next 3 days

Then 4 models were built, two baseline models namely a GRU and a LSTM that are commonly used in time series forecasting thanks to their ability to handle sequence data, the other 2 models were two Generative Adversarial Networks:

- a vanilla GAN with a Generator with 3 GRU layers and 2 Dense Layers and a Discriminator with 3 Convolutional Layers and 2 Dense Layers

- a Wasserstein GAN with Gradient Penalty

The main difference between the two is the utilization of a different loss function, a GAN is a network that has its roots in Game Theory, Generator and Discriminator “fight” participate in a game in which the Generator that needs to produce fake outputs tries to trick the Discriminator, and this is called the adversarial part of the game, to accomplish that the network tries to minimize the Kullback-Leibler Divergence. In the Wasserstein GAN instead is the Wasserstein distance that is being minimized, Roughly speaking let’s imagine two piles of sand representing the two probability distributions. Each grain of sand has a certain weight, and the goal is to move the sand from one pile to the other while minimizing the total amount of work done.



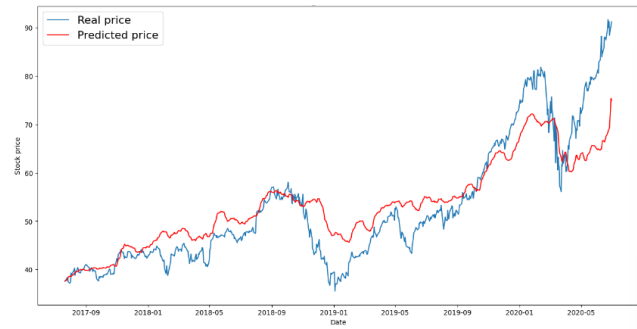
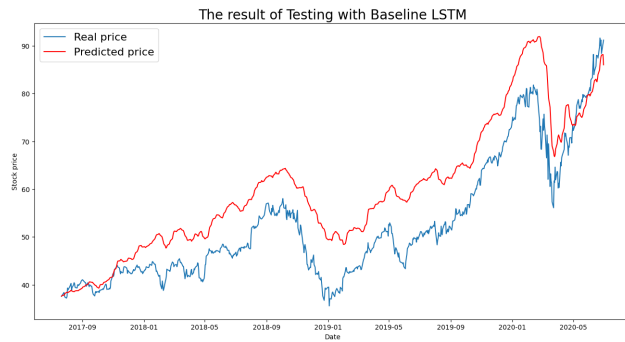
Here the real price is added to the output of the Generator to enhance data length and offer more context, the discriminator receives only prices as input and not all the other variables

These models were run for different epochs, the real price and the predicted price were plotted on a line plot, then the hyperparameters of the Generator of two GANs were tuned with a genetic algorithm, the hyperparameters chosen are the number of units for the various layers, the batch size and the learning rate, here I’ve chosen to focus on the Generator because is the member of the GAN that performs the regression.

To check if the optimization was successful a t-SNE (t-distributed stochastic neighbor embedding) plot for each model was produced, this is a robust statistical method for visualizing high-dimensional data and allows it to confront two different distribution and their similarity.

The results are shown here:

Baseline GRU and Baseline LSTM



Left: Baseline LSTM. *Right:* Baseline GRU

Basic GAN and Basic GAN + Evolutionary Algorithm



Left: Basic GAN. *Right:* Basic GAN + GA

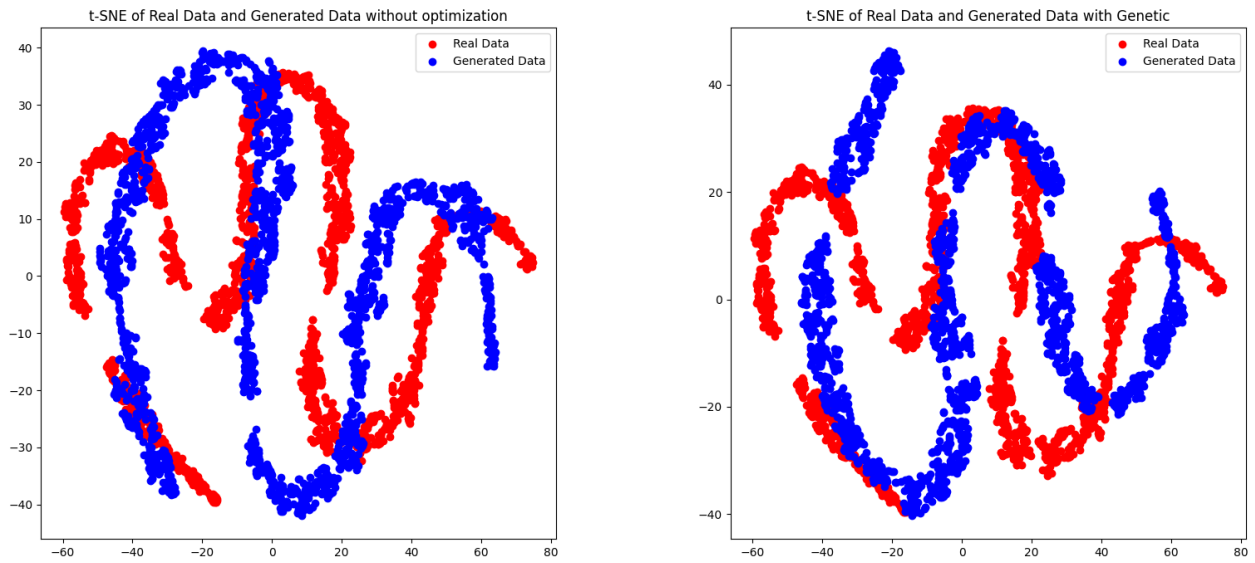
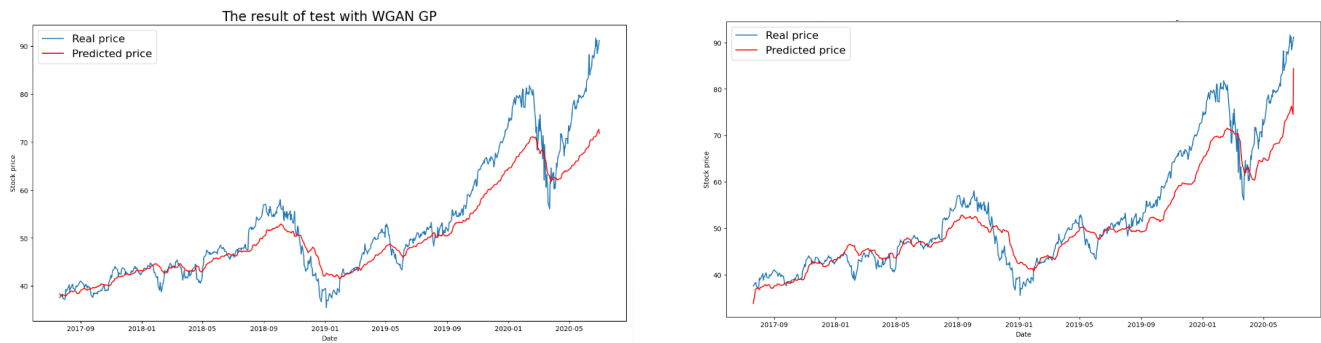
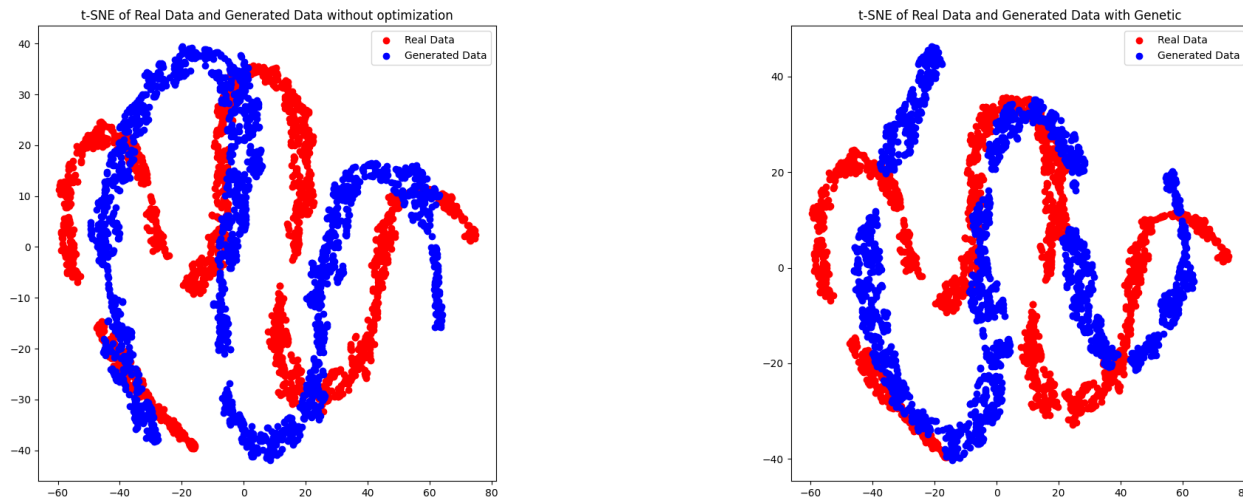


Figure 6.4: Left: the tSNE plot of the Basic GAN with no optimization, Right: the one optimized with the evolutionary approach

WGAN with Gradient Penalty and WGAN GP + Evolutionary Algorithm



Left: WGAN GP. Right: WGAN GP + GA



Left: the tSNE plot of the WGAN GP with no optimization, Right: the optimized one

Model	Train RMSE	Test RMSE
WGAN GP with GRU	2.36	5.15
WGAN GP with GRU + GA	2.13*	4.99*
Basic GAN GRU	1.61	5.82
Basic GAN GRU + GA	1.829	5.02*
Baseline GRU	1.055	6.02
Baseline LSTM	4.08	8.34

Comparison of Models Performance

Metric	Basic GAN	Basic GAN + GA	WGAN GP	WGAN GP + GA
JS Divergence t_1	0.2264	0.1732*	0.2293	0.1449*
JS Divergence t_2	0.2585	0.2218*	0.2065	0.2275
JS Divergence t_3	0.1716	0.2199	0.1658	0.1887

JS and KL Divergence of the various models, the asterisk underline where the optimization reduced one of the two metrics

In conclusion, the answer to the research question is positive and the genetic algorithm has been successful in improving the performance of the model