$DAuGAN$: An approach for augmenting time series imbalanced datasets via latent space sampling using adversarial techniques

Andrei Bratu and Gabriela Czibula
Faculty of Mathematics and Computer Science
Babeş-Bolyai University, Romania
bratu.andrei@stud.ubbcluj.ro, gabriela.czibula@ubbcluj.ro

***Abstract*—*Data augmentation* is a commonly used technique in data science for improving the robustness and performance of *machine learning models*. The purpose of the paper is to study the feasibility of generating synthetic data points of temporal nature towards this end. We propose a general approach named $DAuGAN$ (*Data Augmentation using Generative Adversarial Networks*) for identifying poorly represented sections of a time series, study the synthesis and integration of new data points, and the performance improvement on a benchmark machine learning model. The problem is studied and applied in the domain of algorithmic trading, whose constraints are presented and taken into consideration. The experimental results highlight an improvement in performance on a benchmark reinforcement learning agent trained on a dataset enhanced with $DAuGAN$ to trade a financial instrument.**

***Index Terms*—data augmentation, generative adversarial networks, reinforcement learning, algorithmic trading, time-series, gan**

1. Introduction

Data augmentation is a vast and often used method for enhancing the amount of data available for training a *machine learning* (ML) model. It is well-known that the amount and quality of data available is closely bounded by to the success of any ML project, independent of application domain. There are multiple data augmentation procedures, which are often specific to the application domain and the specific dataset that is used.

For example, image based machine learning tasks often employs operations of contrast adjustment, flipping, translation, cropping, rotation, color space manipulation etc. These present new contexts to the model, helping it to better generalize and to avoid over-fitting [1]. Another examples of data augmentation is SMOTE, or Synthetic Minority Over-sampling Technique, whose purpose is to alter the dataset presented to the algorithm by presenting minority class data points to the classifier more often than they naturally occur (oversampling), while minimising the rate at which the majority class appears (under-sampling). This increases the sensibility of the model for the sub-represented data class, and has applications such as identifying fraud credit card transactions [2].

A more recent augmentation method involves using a *generative adversarial neural network* (GAN) architecture, whose ability to reproduce a statistical distribution is repurposed for creating new, convincing examples of a poorly represented class, or generally any point of the dataset. [3]. GANs are an important machine learning paradigm. Two neural networks engage in a zero-sum game where the Generator network attempts to generate new samples, while the discriminator discerns between real samples and generated samples. The end goal is to train the generator into reproducing the initial train distribution as close as possible. GANs have been used to great effect, with examples such as reproducing the effects of dark matter on astronomical observations [4], generating photo-realistic human faces [5], or applying style transfer operations in the audio domain [6].

Identifying imbalanced classes and enhancing their presence in the time-series would not be devoid of practical applications. One such example is *securities trading*. Securities are defined as any financial instrument that can be bought or sold via an accredited intermediary, creating a supply and demand market. An example is the stock market, which allows buying and selling "shares" - discreet units of ownership in a company. While the price of any share has a correlation with the business performance, there is research that indicates the market's sentiment and domain-specific factors such as national interest rate create a cyclical effect on the price evolution [7] [8].

The current paper is based on work originating from two research questions:

RQ_1 Is it possible to improve the performance of *reinforcement learning* (RL) based trading algorithms through augmenting training data using adversarial techniques?

RQ_2 What is the performance gain of the RL agent trained on the enhanced data over a baseline RL agent trained on the initial data, without augmentation?

Our contribution introduces a general approach named $DAuGAN$ (***D**ata **Au**gmentation using **GAN**s*), to identify poorly represented sections of securities-related time-series. First, we leverage the fact that autoencoder neural network architectures can encode and decode complex temporal dependencies to and from a latent space, reducing the problem of identifying the poorly represented time series periods into

a clustering problem [9] and use a GAN to synthesise new examples of the minority class. Second, we present a method of integrating the synthesized data points into the original time series, respecting the original constraint of the data and measure the performance improvement on a benchmark reinforcement learning algorithm.

We note and acknowledge that the first stage bears resemblance to the TimeGAN method proposed by Yoon et al. for generating time-series with an arbitrary number of steps [10]. However, the paper proposes a multi-purpose flow that can be applied on a general time-series. We observe that securities-related time-series impose further constraints that are not respected using such a general method. Thus, our contribution focuses on integrating the synthesised points back in the original time-series and measuring the method's applicability using a real world benchmark.

The rest of the paper is organized as follows. Section 2 introduces the fundamental concepts used in our approach together with a literature review on time series generation. Our $DAuGAN$ approach is introduced in Section 3 along with the proposed methodology, whilst the experimental results and their analysis are presented in Section 4. The conclusions of the paper and directions to further improve and extend $DAuGAN$ are outlined in Section 5.

2. Background

This section presents a literature overview of the technical notions used in this paper. The importance and evolution of generative adversarial and autoencoder networks are presented, together with a brief review on reinforcement learning. We also include a literature review on topics related to data augmentation and data synthesis on time series.

2.1. Generative adversarial networks

GANs is a deep learning architecture that have been first introduced by Ian Goodfellow et al. [11], that has been heavily used in image based tasks, from synthesising new images [12] to reproducing the content of one image in the style of another.

At a very high level, the generative adversarial network technique pits two deep neural networks against each other in a zero sum game. One of the networks, the *generator*, acts as a map from a latent space towards a desired distribution, sampling noise from the latent space that is synthesised as closely as possible to a point in the distribution. Its counterpart, the *critic*, is fed samples from both the real distribution and from the generator, with the goal of deciding whether the sample is "real" or "fake". Over time the two networks improve at their goal, resulting in better fakes from the generator, but also a better ability to discern the fakes from the critic. Ideally the system converges towards an equilibrium where the critic can no longer separate between the two classes, i.e. it assign equal probability for any sample to be either one of the classes.

Formally, the generator attempts to minimise the value of the following loss function, while the discriminator attempts to maximise it: $E_x[log(D(x))] + E_z[log(1 - D(G(z)))]]$, where x is a random variable corresponding to the real distribution, z is a random variable assigned to the generated distribution, $G(x)$ is the generator's output, $D(x)$ is the discriminator's output and E_v denotes the expected value over all instances v.

The expected value is used to indicate that the loss is the average over all samples of the batch at a given training step. The critic assigns values from 0 to 1, estimating the probability that a sample is real. Letting x denote the real distribution and and z denote the synthesised distribution, the critic attempts to maximise this loss - the upper bound being obtained when all labels are properly assigned - while the generator attempts to minimise it by controlling the second term i.e. generating more convincing examples, signified by the $G(z)$ term. The log operations are derived from the cross entropy between the real and fake distributions.

Notorious problems affecting GANs are *mode collapse* and *vanishing gradients*. *Vanishing gradients* is a general issue in machine learning, where gradients prove insufficient for the machine learning model to update meaningfully. While this issue has classically occurred in neural models with high depth [13] or recurrent networks such as long-short term memory architecture [14]. However, the issue manifests particularly in the case of generative adversarial networks: the unadjusted loss formulation presented above will result in a critic that converges faster than the generator. Thus the critic cannot offer constructive feedback for the generator to improve on, since it perfectly discerns between real and fake.

A connected issue with vanishing gradients that is faced by generative adversarial networks is *mode collapse*. The problem manifests on the generator's side by mapping all latent points to the same synthesised sample. From the perspective of game theory, both *mode collapse* and *vanishing gradients* issues are caused by the two players, the critic and the generator, converging to a local, undesired optimum of the game, that does not offer enough incentives for any of the networks to update their weights [15].

Several improvements on the domain transfer subproblem have been addressing the mode collapse issue. CycleGAN [16] introduces the following adjustments: instead of sampling from a latent random space, the generator samples from the input space of the input domain, with output in the target domain. The critic is fed both generated images and those belonging to the target domain, thus encouraging the generator to learn a better mapping between input and target. Thus, the improvement resides in translating the task into an unsupervised task, as the generator is ideally able to map any image from the first domain to an image in the target domain. A limitation of the CycleGAN paper is the domains being required to be homogeneous [17]. An improvement over the CycleGAN is represented by TraVeLGAN [18], which adds to the classical two network architecture formed of generator and critic a third, siamese network and eliminates the domain homogeneity constraint [19]

The Wasserstein variation of the generative adversarial

network architecture (WGAN), authored by Arjovsky et al. [20] offers a robust method to train GAN architectures. WGAN improve the stability of learning, eliminate problems like mode collapse, and provide meaningful learning curves useful for hyper-parameter searches.

2.2. Autoencoders

The *autoencoder* (AE) architecture uses a two-part neural network to transform the input in a compressed and meaningful representation using the encoder part, and recreating the input using the decoder part [21]. The technique proves immensely flexible, and is of interest to the purpose of this thesis since former research proves that autoencoders are able to encode and decode complex temporal features into the latent space [9]. Furthermore, there are no special theoretical aspects to be considered over a general purpose deep learning architecture. The autoencoder is presented as two symmetrical parts, with a small, latent representation in the middle, usually trained to minimise the mean squared error between the distribution and itself.

AEs can be interpreted as an improvement over the statistical technique of principal component analysis. Principal component analysis with p dimensions identifies an orthonormal base of p vectors that best identify the variance of the input distribution [22]. This technique is limited to linear representations, unlike the manifold organized by the autoencoder. Thus, the AE is able to construct higher fidelity correlations between the original and latent spaces, preserving relationships. Furthermore, there are multiple accounts in literature in using the latent representation over the initial dataset with great effect for increased classification performance, better interpratibility of the obtained clusters, or better ability to generalise over the latent representation [23] [24] [25].

Salakhutidnov and Hinton restrict the latent representation to a binary code which is interpreted as the output of a black-box hash function modelled by the encoder. The hashing is applied in the field of document retrieval, where the hashing of the query is used to retrieve the directly associated documents plus documents from neighbouring hashes. We note that the same task has been approached from a generative approach by Hansen, Hansen et al. in *Unsupervised Neural Generative Semantic Hashing* [26].

2.3. Reinforcement learning

Reinforcement learning (RL) is a paradigm of the machine-learning field where problems are modelled around two fundamental notions: agents and environments. Agents are able to interact with the environment via a defined set of "actions" which change the environment's "state". Defining the problem's solution as a desirable environment state, the agent is conditioned via "rewards" and "punishments" to reach this favourable state. RL purpose is to teach an agent the optimal policy of acting inside an environment. The environment can at any moment be in a certain **state**, state that can be changed by the agent's **actions**. The agent receives feedback from the environment in the form of a **reward**. Using a trade-off between reward and value (future reward received by the agent by taking a certain action in a particular state), the agent learns a policy that decides the best course of action for a given state.

This flexible framework has allowed to model complex real-world situations: scheduling drug administration to patients with chronic conditions in order to minimise risk of negative interactions [27], [28], minimising energy costs associated with cooling of data centers [29] or out matching human players in games such as Go [30].

RL is facing a growing interest in the discipline of algorithmic trading [31], [32]. The interest can be explained by the ease with which the problem can be modelled: given the price fluctuation of a certain instrument, an agent's purpose is to maximise the overall profit. Current frontier in reinforcement learning focuses on improved training performance, particularly incentivising the agent to explore multiple action courses and breaking the state causality effect on training by sampling and replaying random past states [33]. Intuitively, these improvements focus on offering the agent the ability to retrospect and decide on past better courses of action.

2.4. Time series generation

GAN-based methods or generative adversarial network models have emerged as a popular technique for generating or augmenting datasets, especially with images and videos. However, GANs give poor fidelity in networking data, which has both complex temporal correlations and mixed discrete-continuous data types. Although GAN-based time-series generation exists — for instance for medical time series — such techniques fail on more complex data exhibiting poor auto-correlation scores on long sequences while prone to mode collapse.

TimeGAN architecture introduced by Yoon et al. [10] is of strong interest for our paper, as it reinforces the idea that latent spaces can be used to better understand the original time series distribution of the data. Specifically, the paper proposes using two latent spaces, H_S and H_X, where S represents the mathematical space of static features of the time series e.g. gender, while X represents the space of temporal dependencies of the time series e.g. the cholesterol level as the person ages. The paper asserts that instead of using only a generator - discriminator system for creating new samples, introducing supervised learning to the unsupervised generation will increase the fidelity of generated data. The supervised loss comes from two encoder - decoder pairs ($h : S \mapsto H_S, e : H_S \mapsto S$), ($h_X : X \mapsto H_X, e_X : H_X \mapsto X$) between the initial space and latent space, with the GAN networks learning to directly replicate the latent vectors: ($g : Z_S \mapsto H_S, d : H_S \mapsto \mathbb{R}$), ($g_X : Z_X \mapsto H_X, d_X : H_X \mapsto \mathbb{R}$), where Z_S and Z_X are the space the random noise is sampled from. Of particular interest is the use of recurrent neural networks throughout the architecture. Notably, g_X features the use of a 2-step auto-regression dependency for creating the temporal latent

vector. Recurrent neural networks are also used for encoding and decoding between X and H_X.

DoppelGANger architecture introduced by Lin et al. [34] represents a current benchmark in time series generation. It tackles a similar problem with TimeGAN as both separate the generation of static attributes from the time series measurements, although focusing on privacy over accurate reproduction of the target distribution time series. Specifically, the generation procedure for the time series implies a conditional process akin to prior work with Conditional Generative Adversarial Nets [35].

The network further improves by providing a normalization approach that avoids mode collapse on long time series: Instead of normalizing the entire data set at once, using the global minimum and maximum, the algorithm normalizes using the per-sample minimum and maximum. Furthermore, the minimum and maximum are attached as static metadata describing the associated time-series. Thus the generator is responsible for creating the static features is also in charge of creating the features that normalize the time series.

The final DoppelGANger architecture uses three networks for generating data: a generator network used for generating static attributes, a generator network used for generating the minimum and maximum of each time series, as described above. The data generated by the two networks is fed into the third, a recurrent neural network which leverages the provided information plus its internal state to generate measurements. A stacked discriminator critiques the generator's work: one model focuses on the generated static features (also called meta-data), while the other is a recurrent neural network critiquing the generated measurements.

3. Methodology

With the goal of answering our research question RQ1, this section introduces our methodology to create artificial training data using a GAN architecture, with the goal of further improving the performance of reinforcement learning based trading algorithms.

Section 3.1 introduces the dataset used in our study, discuss its particularities and the constraints it will impose on our process. We will present the architecture of the **autoencoder** (LSTM-based autoencoder) we use for translating between the initial space and the latent space, and the experiments we have made with it. Then, we explore the distribution of our latent vectors, and identify underrepresented sequences via **clustering algorithms** (OPTICS). Finally, we describe the **adversarial network** (WGAN) used to sample new examples, and the processing required for them to adhere to the inherent constraints of the dataset.

The main stages employed in synthesis are as follows:

1) The initial space of the instances from the preprocessed dataset is translated into a latent (encoded) space, through a LSTM-based autoencder.
2) The latent vectors determined at the previous stage are clustered using the OPTICS algorithm to identify minority clusters, representing underrepresented inatsnces.
3) A WGAN adversarial network architecture is used to sample new training examples which are then processed to adhere to the constraints of the dataset.

A high-level overview of our $DAuGAN$ approach is depicted in Figure 1.

3.1. Dataset

We remind the reader that the purpose of this paper is to identify anomalies in the evolution of a security price and augment its time series using generative methods. To this end we have chosen to conduct our experiments using a dataset describing the evolution of the price for Apple's company stock, denoted by the AAPL symbol, as present on the New York Stock Exchange. The dataset is not public access but can be provided on demand for scientific purposes.

The dataset presents samples at every 15 minutes, covering the company's price evolution starting from 1^{st} of January 1998 until 3^{rd} of December 2021, totalling 289487 time steps. The data used in our experiment features 7 initial columns: $date$, $time$, $open$, $high$, low, $close$, $volume$. These features are often used in the domain of algorithmic trading, and offer indicators on the price's evolution per time step. The $open$ column describes the price at the start of the time step, $high$ and low describe the maximum and minimum reached throughout the time-step, while the $close$ price describes the price at interval's end. The $volume$ column represents the number of trades executed in the given period.

Table 1 presents a sample fragment from the beginning of the time series.

TABLE 1. AN EXCERPT FROM THE BEGINNING OF THE TIME SERIES

date	time	open	high	low	close	volume
1998/02/01	09:30	13.6250	13.7500	13.5000	13.6875	20270
1998/02/01	09:45	13.6875	13.7500	13.5000	13.6250	334000
1998/02/01	10:00	13.6250	13.7500	13.5625	13.7500	299900
1998/02/01	10:15	13.7500	14.0000	13.6250	14.0000	430201
1998/02/01	10:30	13.9375	14.8125	13.7500	14.6250	944200
1998/02/01	10:45	14.6250	14.7500	14.3750	14.4375	218103

Since the time steps are continuous, there are constraints that apply for any timstep t

$$\forall t \geq 1 : close_{t-1} = open_t \tag{1}$$

$$x_t \leq high_t, \forall t, \forall x \in \{low, close, open\} \tag{2}$$

$$x_t \geq close_t, \forall t, \forall x \in \{low, high, open\} \tag{3}$$

However, real world imposes some exceptions to this constraint. First and foremost, the data used comes only from the trading hours action, starting from 09:30 until 16:00, Monday to Friday, when all traders can take part

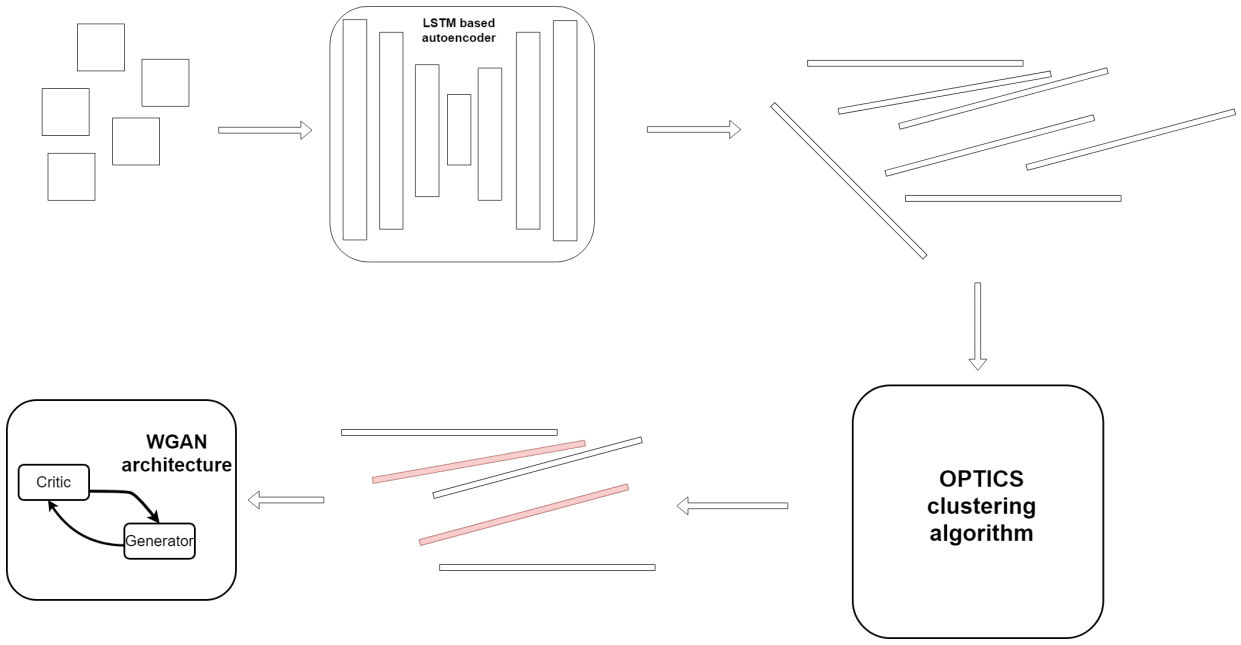

Figure 1. Overview of our $DAuGAN$ approach.

in the market. However, the NYSE, and in general, the exchanges located in the United States, also present a "before-market" and "after-market" period, limited to institutional investors such as pension funds, hedge funds or banks. Furthermore, shares can be traded between any two interested parties, without the exchange as an intermediary.

It is beyond the scope of the paper to identify and enumerate all possible source of discontinuity that could violate Equation (1). We will make the simplifying assumption that the condition holds for any two consecutive steps. We obtain a dataset of 24123 chunks.

We continue by calculating basic statistics for the numerical features of the dataset. The analysis from Table 2 reveals that $volume$ features a very wide, $[10^2, 10^7]$ domain.

TABLE 2. COUNT, MEAN, STANDARD DEVIATION, MINIMUM VALUES FOR ENTIRE DATASET, PLUS UPPER BOUNDS FOR EACH QUARTER OF THE DATASET IN SORTED ORDER.

	open	high	low	close	volume
count	2.894870e+05	2.894870e+05	2.894870e+05	2.894870e+05	2.894870e+05
mean	187.757776	188.038462	187.466230	187.758491	4.580510e+05
std	160.514466	160.689723	160.326415	160.513598	9.206810e+05
min	12.550000	12.950000	11.312500	12.850000	1.000000e+02
25%	78.750000	78.920000	78.500000	78.750000	5.900000e+03
50%	131.040000	131.330000	130.790000	131.040000	1.033630e+05
75%	248.160000	248.605000	247.625000	248.150000	5.635505e+05
max	704.800000	705.070000	704.530000	704.800000	7.514145e+07

We plot the columns $open$ and $volume$, observing that columns $high$, low and $close$ will trend in correlation with $open$, leading to Figure 2. The left side image from Figure 2 presents the histogram of $volume$ column in initial dataset, while the rightmost image depicts the $open$ column evolution in time.

The keen observer will be very interested in the two sudden drops in price illustrated in $open$ column evolution illustrated in the right-side image from Figure 2. They represent a domain specific event called *share splitting*. In a $X : 1$ share split, the price of one share is divided by X while each share pre-split is replaced with X times more. This preserves the value of the investment while lowering the financial bar for buying one share, attracting interest and activity from smaller investors with the better price per action.

3.1.1. Data preprocessing. We discard 'date' and 'time' for training purposes. Considering the exponential distribution of the volume highlighted in Figure 2, we start by applying a logarithm transformation over every feature column, followed by a applying two independent normalizers: one for the $[open, high, low, close]$ columns and a separate one for $volume$. Applying normalization column-wise would violate the constraint of Equation (1), as the features follow different distributions. During autoencoder training we have observed that training on a non-logarithmized but normalized dataset leads to a reproduction collapse, with most values outputted be decoder for $volume$ being zero. Due to an initial domain of $[10^2, 10^7]$, the volume rows with minimal value are expected to be translated to the magnitude of 10^{-5}, which are approximated to 0 by the decoder.

Figure 3 illustrate the data set after normalization. One observes that the transformation of $volume$ feature (left image from Figure 3) is notable, compared to the initial data (left image from Figure 2).

The Pearson correlation heat-map between the features is illustrated in Figure 4. We observe that features have weak correlation, indicating an optimum number of features.

In preparation for the training of the models, we split the time series into chunks of twenty time steps and an overlap of 8 time steps between two chunks formed of the forty

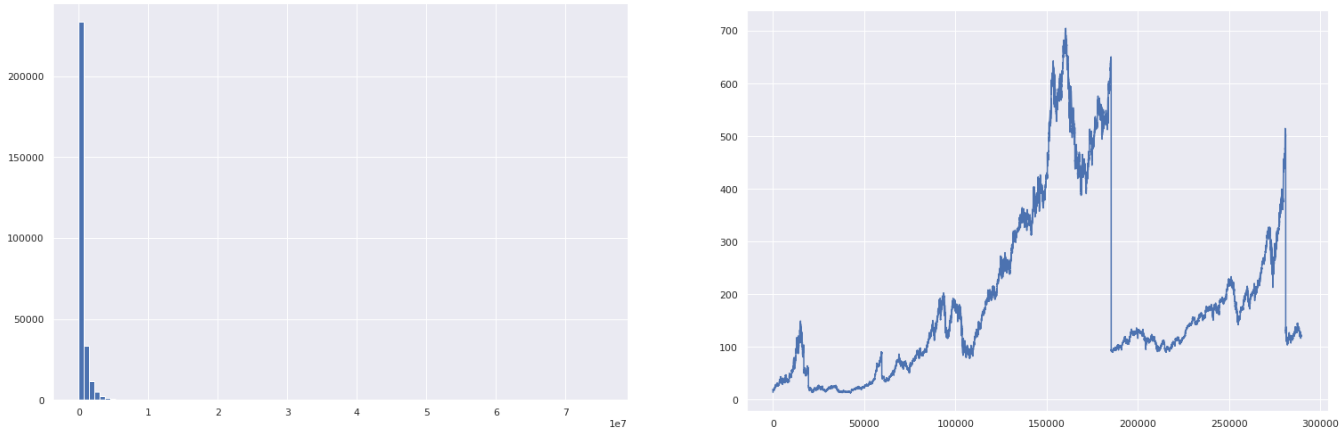

Figure 2. Left: Histogram of $volume$ feature in initial dataset; Right: $open$ feature evolution in time

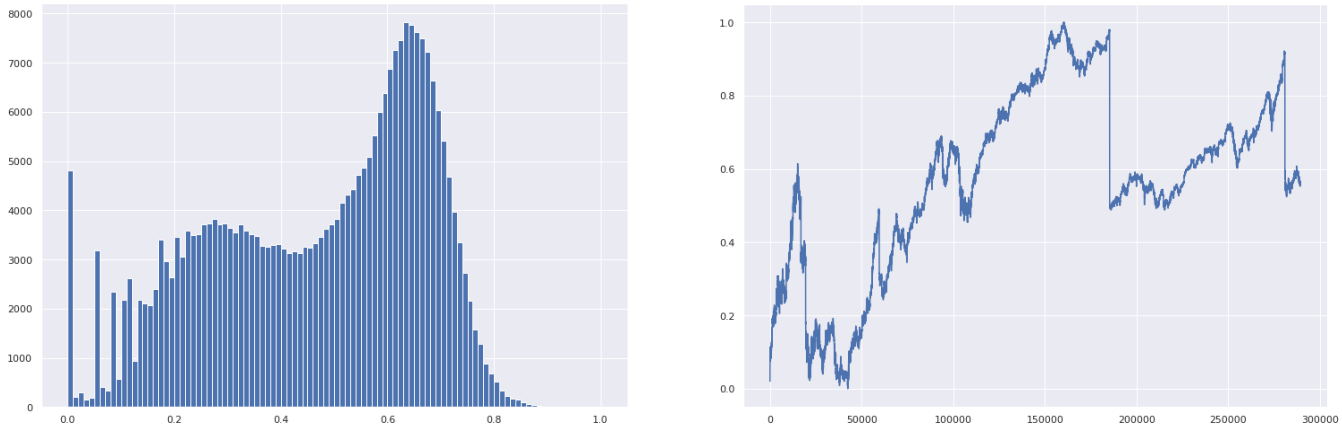

Figure 3. Left: Histogram of $volume$ feature in the normalized dataset; Right: $open$ feature evolution in time in the data set after normalization.

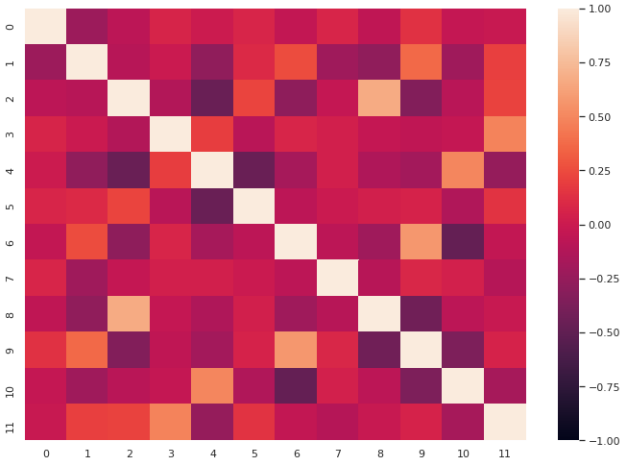

Figure 4. Pearson correlation heat map between features.

consecutive time steps.

3.2. Autoencoder

The preprocessed dataset will be fed into an autoencoder network with the purpose of identifying outliers in a space with reduced dimensionality. Under the intuition that the network should be able to account for temporal dependencies, we propose a neural architecture that makes use of one dimensional convolutions and LSTM units. Our architecture proposes using 1-dimensional convolutional networks at the start of encoder network, followed by stacked LSTM layers. The decoder mirrors the layers of the encoder.

When reproducing the experiments, one should expect training and validation losses in the domain of 10^{-4} after 1000 epochs of training. We have experimented with architectures similar to U-NET [36], with the convolutional layers progressively reducing dimensionality until the latent space is reached, but the performance has proven inferior despite the intuition. Besides the val_{loss} metric, we use Person correlation between dimensions of latent space in order to determine minimum number of non-redundant dimensions.

Our 1-dimensional convolutions use $same$ padding mode, and do not reduce the dimensionality of the data.

Rather, the dimensionality is increased, and then fed into the LSTM layers. Since the three LSTM layers are stacked, the first two actually return the output of the hidden units instead of the normal output with reduced dimensionality, allowing for further manipulation. Following best practices from literature, we intertwine Dropout layers with a rate $\beta = 0.3$ for regularization purposes [37], and use PReLu as activation between all layers [38]. The optimization algorithm involved in training is Stochastic Gradient Descent with a learning rate of $\alpha = 0.0001$ that uses Nesterov momentum [39]. Our choice for the optimizer is motivated by the fact that Adam is not guaranteed to converge [40]. The architecture of the proposed AE is illustrated in Figure 5.

3.2.1. Analysing the Latent Space. Applying the autoencoder on the dataset, we obtain an euclidean latent space on which clustering is possible. We run an OPTICS algorithm [41] to identify minority clusters, sampling the epsilon hyper-parameter linearly from the $[0.05, 0.5]$ range. We use the elbow method [42] applied on Total Variance. Our analysis yields a majority cluster of of $\approx$ 20000 points, the rest being outliers overwhelmingly classified as noise. In order to visualise the clusters we employ principal component analysis [43] and reduce dimensionality to three axis.

3.3. Synthesising New Samples

The generative architecture presented in Figure 6 is able to synthesise credible examples that resemble the minority class. To confirm, from a statistical viewpoint, that the examples resemble the minority class, we applied a multivariate Wilcoxon test [44] between the real minority points and the generated points. We have resorted to providing our own implementation for the multi-variate Wilcoxon test in accordance with literature [45]. A p-value of 0.7275 was obtained, thus being unable to refute the null hypothesis, meaning that there is no significant difference between the real and the generated points, at a significance level $\alpha = 0.05$. Further proof can be observed in Figure 7 by plotting the distribution of the synthesised latent vectors against the minority and majority clusters.

Figure 8 depicts the correlation heat map for the synthesised examples.

We also note that the feature correlation for synthesised latent vectors slightly differs from the minority features' correlation, as shown in Figure 9.

3.3.1. Smoothing the synthesised examples. A limitation of our generative system becomes evident when we transform the synthesised latent vectors back into the original space. While the system is able to model examples similar to the minority space, it has not been programmed to account for the constraints from Formulae (1), (2) and (3).

We approach this issue from the perspective of an optimization problem: find the closest point to the initially synthesised data point that satisfies all constraints simultaneously. To this end, we employ Bayesian search, a method with numerous applications due to its ability to optimize black-box functions [46]. The search attempts different permutations of the domain, using a Bayesian process to choose the next permutation to be tested considering the points attempted in the past.

We use the Bayesian search in tandem with a greedy algorithm. For each generated data point, we iteratively "smooth" the values of the time steps contained. If the time step's values do not respect the constraint imposed, the time step will be assigned a negative value, and a new permutation will be verified. Should the time step be found adequate, we assign the inverse of the distance between the original location of the sampled synthetic point and the current location of the data point. It should be noted that we redefine current location as we alter the time steps.

Formally, we define $d = \|clv - olv\|^2$ (clv denotes the current latent vector and olv represents the original latent vector) and ask the Bayesian search to optimize the black-box function γ for each time step it contains, minimising the distance d at the overall chunk level. After fixing a time step, we set the $open$ price of the time step that follows to the $close$ price of the current one in order to preserve continuity.

$$\gamma(open, high, low, close) = \begin{cases} -1 & if\ \neg(time\ step\ passes) \\ 1/d & otherwise \end{cases}$$

We allow the Bayesian search to sample γ's domain ten times, samples divided in half for an initial mapping of the domain space, and the last five conducted under Bayesian optimization. We constrain the search space of the Bayesian process to $X \pm \sqrt{X}$, where X is the value for any of the features. We select the point from the domain that has achieved the highest score, and integrate it in the time series. We would like to note that γ suffers a small modification for all time steps with index ≥ 1. Since restriction from Formula (1) must hold, for $t \geq 1$ we only search the domain for $\{high, low, close\}$.

Our approach towards integrating a synthesized chunk ζ into the original time series consists in finding a pair of consecutive chunks, defined by index $t < len(time\_series) - 1$, such that we minimize the following objective:

$$\begin{aligned} &\|(close_t, open_{t+1}, \gamma(volume_t : volume_{t+1}) \\ &- (open_\zeta, close_\zeta, \gamma(volume_\zeta)\|^2 \end{aligned} \tag{4}$$

We hypothesise that the objective is able to identify a pair t such that we minimise the difference between $(close_t, open_\zeta), (close_\zeta, open_{t+1})$, preserving the overall naturalness of the time series. We have experimented with including the 'volume' in the optimization algorithm, but decided to leave it out in order to minimise the difference between prices. Furthermore, we hypothesise that the difference in 'volume' relative to its neighbourhood is one of the main factors in leading to the outlier status of a point, and choose to preserve it. Table 3 presents a chunk fragment obtained from our generation process.

Model: "encoder"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 20, 5)]	0
conv1d (Conv1D)	(None, 20, 256)	5376
p_re_lu (PReLU)	(None, 20, 256)	5120
dropout (Dropout)	(None, 20, 256)	0
conv1d_1 (Conv1D)	(None, 20, 256)	262400
p_re_lu_1 (PReLU)	(None, 20, 256)	5120
dropout_1 (Dropout)	(None, 20, 256)	0
conv1d_2 (Conv1D)	(None, 20, 256)	262400
p_re_lu_2 (PReLU)	(None, 20, 256)	5120
dropout_2 (Dropout)	(None, 20, 256)	0
lstm (LSTM)	(None, 20, 256)	525312
p_re_lu_3 (PReLU)	(None, 20, 256)	5120
dropout_3 (Dropout)	(None, 20, 256)	0
lstm_1 (LSTM)	(None, 20, 256)	525312
p_re_lu_4 (PReLU)	(None, 20, 256)	5120
dropout_4 (Dropout)	(None, 20, 256)	0
lstm_2 (LSTM)	(None, 12)	12912

Total params: 1,619,312
Trainable params: 1,619,312
Non-trainable params: 0

Model: "decoder"

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 12)]	0
repeat_vector (RepeatVector)	(None, 20, 12)	0
lstm_3 (LSTM)	(None, 20, 256)	275456
p_re_lu_5 (PReLU)	(None, 20, 256)	5120
dropout_5 (Dropout)	(None, 20, 256)	0
lstm_4 (LSTM)	(None, 20, 256)	525312
p_re_lu_6 (PReLU)	(None, 20, 256)	5120
dropout_6 (Dropout)	(None, 20, 256)	0
conv1d_3 (Conv1D)	(None, 20, 256)	262400
p_re_lu_7 (PReLU)	(None, 20, 256)	5120
dropout_7 (Dropout)	(None, 20, 256)	0
conv1d_4 (Conv1D)	(None, 20, 256)	262400
p_re_lu_8 (PReLU)	(None, 20, 256)	5120
dropout_8 (Dropout)	(None, 20, 256)	0
conv1d_5 (Conv1D)	(None, 20, 256)	262400
p_re_lu_9 (PReLU)	(None, 20, 256)	5120
dropout_9 (Dropout)	(None, 20, 256)	0
time_distributed (TimeDistri	(None, 20, 5)	1285
time_distributed_1 (TimeDist	(None, 20, 5)	0

Total params: 1,614,853
Trainable params: 1,614,853
Non-trainable params: 0

Figure 5. The architecture of our autoencoder.

Model: "critic"

Layer (type)	Output Shape	Param #
input_3 (InputLayer)	[(None, 12, 1)]	0
conv1d_6 (Conv1D)	(None, 6, 16)	80
leaky_re_lu (LeakyReLU)	(None, 6, 16)	0
conv1d_7 (Conv1D)	(None, 3, 16)	1040
leaky_re_lu_1 (LeakyReLU)	(None, 3, 16)	0
conv1d_8 (Conv1D)	(None, 2, 16)	1040
leaky_re_lu_2 (LeakyReLU)	(None, 2, 16)	0
flatten (Flatten)	(None, 32)	0
dense_1 (Dense)	(None, 100)	3300
dense_2 (Dense)	(None, 1)	101

Total params: 5,561
Trainable params: 5,561
Non-trainable params: 0

Model: "generator"

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 15, 1)]	0
dense_3 (Dense)	(None, 15, 20)	40
conv1d_9 (Conv1D)	(None, 8, 16)	1296
leaky_re_lu_3 (LeakyReLU)	(None, 8, 16)	0
conv1d_10 (Conv1D)	(None, 4, 16)	1040
leaky_re_lu_4 (LeakyReLU)	(None, 4, 16)	0
flatten_1 (Flatten)	(None, 64)	0
dense_4 (Dense)	(None, 100)	6500
dense_5 (Dense)	(None, 100)	10100
dense_6 (Dense)	(None, 12)	1212

Total params: 20,188
Trainable params: 20,188
Non-trainable params: 0

Figure 6. The architecture of our WGAN.

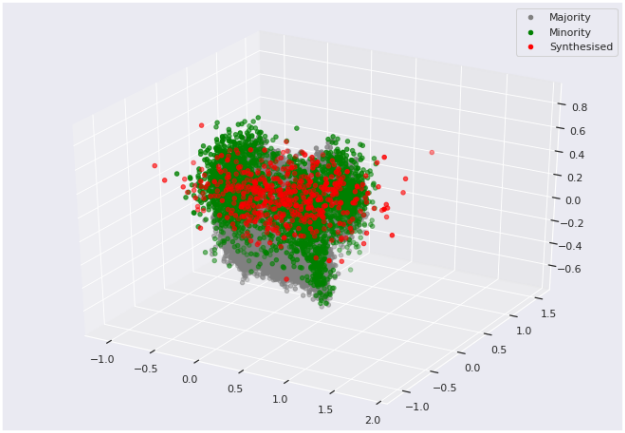

Figure 7. Distribution of latent points before and after synthesising new examples.

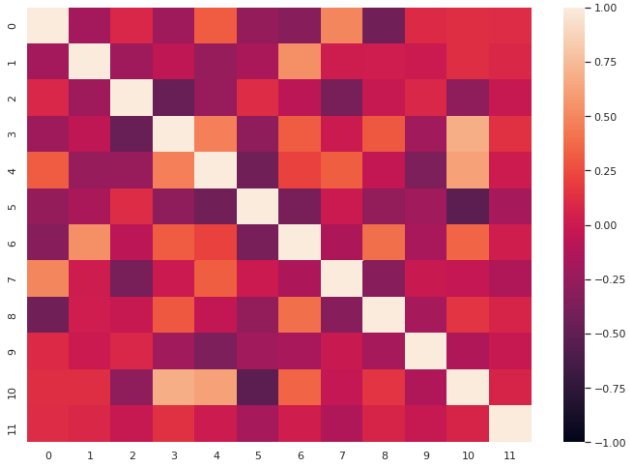

Figure 8. Correlation heat map for fake examples.

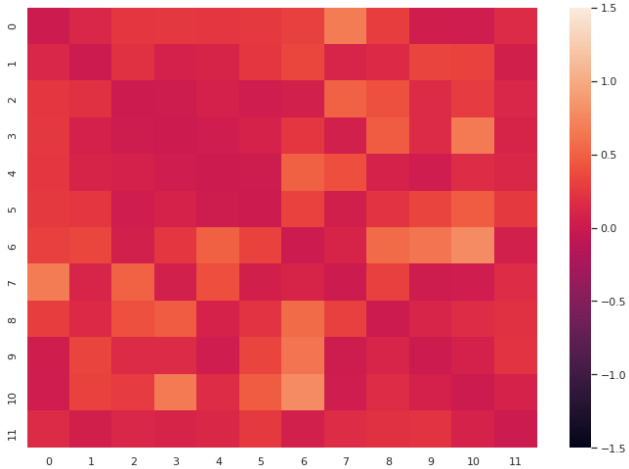

Figure 9. Absolute difference between the minority correlation map and fake examples correlation map.

TABLE 3. CHUNK FRAGMENT OBTAINED FROM OUR GENERATION PROCESS.

	open	high	low	close	volume
0	326.366272	341.629822	314.827942	330.429718	517.638489
1	330.429718	336.826141	310.153687	325.654053	899.540161
2	325.654053	338.448029	311.383545	327.100647	2897.673340
3	327.100647	351.454468	313.967712	351.090942	3598.678955
4	351.090942	340.434845	313.185944	329.017059	4032.858887
5	329.017059	340.508057	313.215607	329.068115	5119.424316
6	329.068115	335.061981	308.940491	324.127441	153.940964
7	324.127441	331.650665	305.705658	320.777008	144.449570
8	320.777008	335.563629	308.537842	324.214722	4437.651367
9	324.214722	340.812073	313.569183	329.397736	3368.686523
10	329.397736	340.041931	312.984558	328.713165	2112.148193
11	328.713165	344.093750	312.746857	319.489136	575.971191
12	319.489136	339.126801	312.318512	327.886292	1199.597290
13	327.886292	344.057251	316.442230	332.473938	9648.517578
14	332.473938	346.419373	318.820251	334.856781	5601.142578
15	334.856781	345.538300	318.073364	334.029663	4347.550781

4. Results and discussion

With the goal of answering research question **RQ2**, we discuss in Section 4.1 the benchmark used to assess the effectiveness of our training data augmentation methodology introduced in Section 3 in improving the performance of reinforcement learning based trading algorithms. The obtained results are then presented in Section 4.2.

4.1. Benchmark methodology and data preparation

In order to assess the impact of our augmentation we devise the following benchmark. We employ the *FinRL* library, which offers benchmark trading algorithms built using the reinforcement learning paradigm, pre-built algorithm training environments as well as standardized procedures of data preparation [47]. Furthermore, we opt for an independent benchmark in order to provide credibility to our results.

A difficulty met in comparing the impact of our augmentation is the uneven amount of data samples present in the original versus augmented time-series. We opt for holding 80% of the original time-series for training purposes, while employing the rest of 20% for validation. In order to obtain

a fair comparison we employ our augmentation procedure only on the training dataset. We acknowledge that this method is imperfect, as the improvement in performance could partially or totally be explained by the algorithm observing more data points in the augmented time series. We have no perfect answer to this issue; nonetheless we propose that training two algorithms until convergence - defined as the point where the return on training the model becomes marginal in terms of performance gain - and observing a positive impact of our augmentation on the unaugmented time series should prove the validness of our augmentation procedure.

We apply a data pre-processing procedure offered by *FinRL* library on both time series. The procedure adds technical indicators, whose purpose is to highlight trends in a security's price evolution, such as relative volatility, magnitude of price shifts, or trends in price shift. The effect of adding indicators as data features includes faster convergence and better performance [48]. The indicators added by *FinRL* are 12-MACD, Bollinger Bands, 30-RelativeStrengthIndex, 30-CommodityChannelIndex, 30-AverageDirectionalIndex, 30-CloseSimpleMovingAverage,60-CloseSimpleMovAvg.

We note that *FinRL* requires the presence of two extra columns in order to calculate the technical indicators: the tic column which represents the security's descriptor, (*FinRL* is able to trade multiple securities at once, hence the requirement for this column) and the $date$ column which represents the date and time of the column. We make the simplifying assumption that the security has been traded continously in intervals of 15 minutes starting from an arbitrary date, discarding the particularities discussed in Chapter 2.

As benchmark we use the provided reinforcement learning algorithms [49] and trading environment. The agent's state is described by the tuple $(shares,\ capital)$, where $shares$ describes the number of owned shares, while $capital$ describes the amount of monetary units available for buying shares. The state of the agent is completed by the time series, as seen until the moment t of time. We set the initial state to $(shares_0, capital_0) = (0, 200000)$. We define the portfolio value with r shares as $capital_t + \sum_{0,r} open_t$.

At every time step, the agent's action space spans the integers $[-min(k, shares_t), k]$. k is a positive integer hyperparameter set in the environment. Negative integers indicate selling that amount of shares, receiving an amount of capital equal to the opening price for each sold share. Positive amounts indicate buying stock units, and has the reverse effect on capital. We would like the special case $k = 0$ denoting the agent's choice to hold its current position.

4.2. Results

We follow the methodology presented above using different reinforcement algorithms and amount of samples. We find a positive correlation between the amount of samples introduced in the original time series and the performance improvement of the algorithm. We define the performance improvement as the difference in portfolio value between training on the original time series and augmented series after applying the trained algorithms over validation dataset.

Table 4 summarizes the performance improvement for our $DAuGAN$ augmentation method. permutations of introduced samples and benchmark algorithm used: Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Advantage Actor Critic (A2C).

TABLE 4. PERFORMANCE IMPROVEMENT FOR OUR AUGMENTATION METHOD, MEASURED IN PERMUTATIONS OF INTRODUCED SAMPLES AND BENCHMARK ALGORITHM USED.

Algorithm	# of introduced samples		
	2000	**4000**	**6000**
DDPG	+2.71%	+3.44%	**+4.25%**
PPO	+3.12%	+3.47%	**+3.82%**
A2C	+1.89%	+2.31%	**+2.42%**

Table 4 reveals a performance improvement gained by the RL algorithms on the enhanced training data. We note that the performance improvement increases as the number of artificially generated samples introduced in the training data grows. One also observes that the performance improvement is not as strong with modern reinforcement learning algorithms such as Advantage Actor Critic.

5. Conclusions and future work

We introduced in this paper a general approach named $DAuGAN$ (***D**ata **Au**gmentation using **GAN**s*), with the goals of identifying poorly represented sections of a time series, studying the synthesis of new data points and their integration into the time series, and assessing the performance improvement on a benchmark machine learning model. We have chosen to investigate the performance of our $DAuGAN$ approach in the domain of *trading* that is nowadays an active topic in the discipline of economics, consisting of the buying and selling of various financial instruments, from stocks, futures, options to cryptocurrencies, with the purpose of turning a profit.

The research questions stated at the beginning of the paper were answered. We introduced a methodology for augmenting training data using adversarial techniques with the goal of improving the performance of reinforcement learning-based trading algorithms. The experiments highlighted a performance gain of a RL agent trained on the enhanced data over a baseline RL agent trained to trade a financial instrument.

We conclude that data synthetisation is a valid training augmentation in the area of algorithmic trading, and hypothesise that the procedure can be adapted for multiple tasks involving time series. For the future we are interested in applying the technique to mission-critical tasks such as screening for rare medical conditions, defect detection or weather nowcasting. Furthermore, we hypothesised that moving time series related issues in the latent space opens up myriads of augmentation possibilities that could be attempted. We focused on weather a universal, reinforcement-learning based augmentation procedure could be devised

over the latent space. The experiments highlighted a performance gain of a RL agent trained on the enhanced data over the agent trained on the data without augmentation.

We would also like to investigate further improvements in the methodology of our current experiment. While finding the autoencoder sufficient in reproducing small chunks of a time series, we would like to reproduce the use of recurrent neural networks at the generation step [10], combining it with the auto-normalization discussed in DoppelGANger paper [34] in order to obtain longer synthesised time series. Large samples could be treated as independent training episodes for the benchmark reinforcement learning algorithm, transforming the dataset from a continuous time series into a shuffled array of episodes, each episode being either real or fully synthesised. Per the findings of the Deep Q-Network (DQN) paper regarding shuffling the episodes [50] we acknowledge that this approach has the potential of being superior to our approach of interleaving synthesised chunks as naturally as possible into the series.

Data Availability

The data used to support the findings of this study are available from the first author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Funding Statement

The research leading to these results has received funding from the NO Grants 2014-2021, under Project contract no. 26/2020.

Acknowledgments

The first author acknowledges the financial support received from Babeş-Bolyai University through the special scholarship for scientific activity for the academic year 2020-2021.

References

[1] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, jul 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," 2011.

[3] V. Sandfort, K. Yan, P. J. Pickhardt, and R. M. Summers, "Data augmentation using generative adversarial networks (CycleGAN) to improve generalizability in CT segmentation tasks," *Scientific Reports*, vol. 9, no. 1, Nov. 2019. [Online]. Available: https://doi.org/10.1038/s41598-019-52737-x

[4] M. Mustafa, B. Deborah, B. Wahid, L. Zarija, A.-R. Rami, and M. K. Jan, "CosmoGAN: creating high-fidelity weak lensing convergence maps using Generative Adversarial Networks." *Comput. Astrophys.*, vol. 6, p. Article number: 1, 2019.

[5] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, 2019, pp. 4396–4405.

[6] M. Pasini, "MelGAN-VC: Voice Conversion and Audio Style Transfer on arbitrarily long samples using Spectrograms," 2019.

[7] M. Chauvet, "Stock market fluctuations and the business cycle," *SSRN Electronic Journal*, 2001. [Online]. Available: https://doi.org/10.2139/ssrn.283793

[8] S. Edwards, J. G. Biscarri, and F. P. de Gracia, "Stock market cycles, financial liberalization and volatility," Tech. Rep., Jul. 2003. [Online]. Available: https://doi.org/10.3386/w9817

[9] N. Tavakoli, S. Siami-Namini, M. A. Khanghah, F. M. Soltani, and A. S. Namin, "Clustering time series data through autoencoder-based deep learning models," *CoRR*, vol. abs/2004.07296, 2020. [Online]. Available: https://arxiv.org/abs/2004.07296

[10] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[11] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-Attention Generative Adversarial Networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 7354–7363.

[12] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *CoRR*, vol. abs/1701.00160, 2017. [Online]. Available: http://arxiv.org/abs/1701.00160

[13] G. B. Goh, N. O. Hodas, and A. Vishnu, "Deep learning for computational chemistry," *Journal of Computational Chemistry*, vol. 38, no. 16, pp. 1291–1307, Mar. 2017. [Online]. Available: https://doi.org/10.1002/jcc.24764

[14] J. Kolen, *A field guide to dynamical recurrent networks*. New York: IEEE Press, 2001.

[15] N. Kodali, J. D. Abernethy, J. Hays, and Z. Kira, "How to train your DRAGAN," *CoRR*, vol. abs/1705.07215, 2017. [Online]. Available: http://arxiv.org/abs/1705.07215

[16] J. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *CoRR*, vol. abs/1703.10593, 2017. [Online]. Available: http://arxiv.org/abs/1703.10593

[17] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2242–2251.

[18] M. Amodio and S. Krishnaswamy, "Travelgan: Image-to-image translation by transformation vector learning," *CoRR*, vol. abs/1902.09631, 2019. [Online]. Available: http://arxiv.org/abs/1902.09631

[19] ——, "TraVeLGAN: Image-To-Image Translation by Transformation Vector Learning," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8975–8984, 2019.

[20] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 214–223.

[21] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, vol. abs/2003.05991, 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[22] K. Pearson, "LIII. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, Nov. 1901. [Online]. Available: https://doi.org/10.1080/14786440109462720

[23] G. E. Hinton, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006. [Online]. Available: https://doi.org/10.1126/science.1127647

[24] I. Goodfellow, *Deep learning*. Cambridge, Massachusetts: The MIT Press, 2016.

[25] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, pp. 969–978, 2009, special Section on Graphical Models and Information Retrieval. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0888613X08001813

[26] C. Hansen, C. Hansen, J. G. Simonsen, S. Alstrup, and C. Lioma, "Unsupervised neural generative semantic hashing," *CoRR*, vol. abs/1906.00671, 2019. [Online]. Available: http://arxiv.org/abs/1906.00671

[27] C. Yu, J. Liu, and S. Nemati, "Reinforcement learning in healthcare: A survey," *CoRR*, vol. abs/1908.08796, 2019. [Online]. Available: http://arxiv.org/abs/1908.08796

[28] L. Wang, W. Zhang, X. He, and H. Zha, "Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2447–2456.

[29] Y. Li, Y. Wen, D. Tao, and K. Guan, "Transforming cooling optimization for green data center via deep reinforcement learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 5, pp. 2002–2013, 2020.

[30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[31] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," in *Proceedings of ICAIF '20: ACM International Conference on AI in Finance*. New York, NY, USA: Association for Computing Machinery, 2020.

[32] T. Théate and D. Ernst, "An application of deep reinforcement learning to algorithmic trading," *Expert Systems with Applications*, vol. 173, p. 114632, 2021.

[33] D. Mehta, "State-of-the-Art Reinforcement Learning Algorithms," *International Journal of Engineering Research and TEchnology*, vol. 8, pp. 717–722, 2020.

[34] Z. Lin, A. Jain, C. Wang, G. C. Fanti, and V. Sekar, "Generating high-fidelity, synthetic time series datasets with doppelganger," *CoRR*, vol. abs/1909.13403, 2019. [Online]. Available: http://arxiv.org/abs/1909.13403

[35] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[36] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015. [Online]. Available: http://arxiv.org/abs/1502.01852

[39] A. Botev, G. Lever, and D. Barber, "Nesterov's accelerated gradient and momentum as approximations to regularised update descent," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, May 2017. [Online]. Available: https://doi.org/10.1109/ijcnn.2017.7966082

[40] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=ryQu7f-RZ

[41] M. Ankerst, M. M. Breunig, H. peter Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure." ACM Press, 1999, pp. 49–60.

[42] M. A. Syakur, B. K. Khotimah, E. M. S. Rochman, and B. D. Satoto, "Integration k-means clustering method and elbow method for identification of the best customer profile cluster," *IOP Conference Series: Materials Science and Engineering*, vol. 336, p. 012017, Apr. 2018. [Online]. Available: https://doi.org/10.1088/1757-899x/336/1/012017

[43] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, Apr. 2016. [Online]. Available: https://doi.org/10.1098/rsta.2015.0202

[44] H. Oja and R. H. Randles, "Multivariate nonparametric tests," *Statistical Science*, vol. 19, no. 4, Nov. 2004. [Online]. Available: https://doi.org/10.1214/088342304000000558

[45] C. fan Sheu and S. O'Curry, "Implementation of nonparametric multivariate statistics with s," *Behavior Research Methods, Instruments, & Computers*, vol. 28, no. 2, pp. 315–318, Jun. 1996. [Online]. Available: https://doi.org/10.3758/bf03204789

[46] R. Turner, D. Eriksson, M. McCourt, J. Kiili, E. Laaksonen, Z. Xu, and I. Guyon, "Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020," *CoRR*, vol. abs/2104.10201, 2021. [Online]. Available: https://arxiv.org/abs/2104.10201

[47] X.-Y. Liu, H. Yang, Q. Chen, R. Zhang, L. Yang, B. Xiao, and C. Wang, "FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance," *SSRN Electronic Journal*, 2020. [Online]. Available: https://doi.org/10.2139/ssrn.3737859

[48] F. B. Oriani and G. P. Coelho, "Evaluating the impact of technical indicators on stock forecasting," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, Dec. 2016. [Online]. Available: https://doi.org/10.1109/ssci.2016.7850017

[49] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1509.02971

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602