Synthetic Data Augmentation for Deep Reinforcement Learning in Financial Trading

Chunli Liu
chunli.liu@kcl.ac.uk
King's College London
London, United Kingdom

Carmine Ventre
carmine.ventre@kcl.ac.uk
King's College London
London, United Kingdom

Maria Polukarov
maria.polukarov@kcl.ac.uk
King's College London
London, United Kingdom

ABSTRACT

Despite the eye-catching advances in the area, deploying Deep Reinforcement Learning (DRL) in financial markets remains a challenging task. Model-based techniques often fall short due to epistemic uncertainty, whereas model-free approaches require large amount of data that is often unavailable.

Motivated by the recent research on the generation of realistic synthetic financial data, we explore the possibility of using augmented synthetic datasets for training DRL agents without direct access to the real financial data. With our novel approach, termed synthetic data augmented reinforcement learning for trading (SDARL4T), we test whether the performance of DRL for financial trading can be enhanced, by attending to both profitability and generalization abilities. We show that DRL agents trained with SDARL4T make a profit which is comparable, and often much larger, than that obtained by the agents trained on real data, while guaranteeing similar robustness. These results support the adoption of our framework in real-world uses of DRL for trading.

CCS CONCEPTS

• **Computing methodologies → Markov decision processes**; **Neural networks**; **Reinforcement learning**.

KEYWORDS

Deep Reinforcement Learning, Quantitative Finance, Markov Decision Process, Generative Model, Synthetic data, Financial Trading.

ACM Reference Format:
Chunli Liu, Carmine Ventre, and Maria Polukarov. 2022. Synthetic Data Augmentation for Deep Reinforcement Learning in Financial Trading. In *3rd ACM International Conference on AI in Finance (ICAIF '22), November 2–4, 2022, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3533271.3561704

1 INTRODUCTION

Recent research has highlighted the great potential of deploying reinforcement learning (RL) in trading [22]. Indeed, the two areas seem perfectly aligned: the core idea of RL is about finding an optimal (i.e., a reward maximizing) policy through interactions with the environment, and the goal of trading is to develop a successful (i.e., profitable) (algorithmic) strategy. As a result, numerous studies have focused on the design of (deep) RL algorithms for solving financial trading problems and portfolio optimization, e.g., [4, 30].

However, despite the similarity between training an RL agent in a loop by interacting with the environment and improving a trading strategy through repeated interactions with the market, there are many challenges to overcome. One of the biggest challenges is the impossibility to feasibly train RL agents to cope with all the dynamics of real-world financial markets, as live interactions with the real environment may lead to large losses. One typical approach is then to rely on simulated financial markets to train an RL agent in a model-based way. This requires a well-designed simulated environment that would provide adequate observations and feedback to RL agents. Unfortunately, known models suffer from epistemic uncertainty, due to, e.g., calibration issues or simplified dynamics. Model-free is the other main approach, whereby historical financial data is used to train the agent. For this to work to some acceptable level, we usually require a large amount of data to learn the mapping between the RL agent actions and the market states. Data is however a scarce resource in finance for a number of reasons: history is not that long (e.g., for technological companies we only have roughly 25 years of data), costs to acquire large high-frequency datasets are prohibitive, privacy issues limit the extent to which data can be shared and so on. This often leads to the common data inefficiency problem in RL.

Against this background, we propose a novel method termed synthetic data augmented reinforcement learning for trading (SDARL4T) that addresses the above challenges in training RL agents for financial trading by using augmented synthetic data. Our work builds upon the emerging research on the generation of realistic synthetic data in finance. We see this as a natural way to address the data scarcity problem in model-free RL, and are inspired by the successful applications of data augmentation methods in computer vision, which demonstrate enhanced generalization abilities. We conduct an extensive empirical analysis that assesses the feasibility and performance of using augmented synthetic data for training RL agents for financial trading: Specifically, we compare the performances of two popular Deep RL (DRL) agents on both Equity and Cryptocurrency Markets. The main contributions of our work are:

- We show the feasibility of training profitable DRL trading agents solely on augmented synthetic data, without the need to directly access market data;
- The SDARL4T agents can make comparable or even higher profits than the agents trained on real data, in terms of higher annual return, Sharpe ratio and Sortino ratio;
- The SDARL4T agents show great stability and robustness;

- The SDARL4T agents show trading behaviors (i.e., being more passive or aggressive) different from those of the corresponding agents trained on real data;
- We show that the extent to which the quality of synthetic data matters depends on the DRL algorithm adopted (with PPO agents being more sensitive than A2C SDARL4T agents).

The remainder of the paper is organized as follows. In Section 2, we overview the relevant literature. In Section 3, we formalize the financial trading problem in our experimental study, and introduce the settings and the methodologies used in our trading environment. In Section 4, we introduce the DRL agents deployed in our experiments. Section 5 then provides a detailed description of both the real and the augmented synthetic datasets, used in our study. In Section 6, we present our experimental results and evaluate the feasibility and performance of our proposed approach. We conclude and discuss future work in Section 7.

2 RELATED WORK

As the need to generate realistic market data in finance has grown, much work has been recently done on how to use deep learning generative models to effectively learn the distributions and properties of the real financial market data. While there is no common standardized evaluation criteria or benchmarks for assessing the quality of the generated synthetic data, many evaluations have been conducted by measuring the distribution similarity between the real data and the synthetic data [1].

Several models for generating realistic synthetic financial datasets that use so called Generative Adversarial Networks (GANs) have been proposed, such as Recurrent Conditional GAN (RCGAN) [9], C-RNN-GAN [20] WaveGAN [8] Conditional Generative Adversarial Networks (cGANs) [14], QuantGAN [28] and StockGAN [16]. Of particular relevance to our study is the framework named Time-series Generative Adversarial Networks, also known as TimeGAN [31]. It has shown great performance in generating realistic synthetic data and, importantly for our purposes, TimeGAN can not only generate the time-series sequence data, but also the time-series cross-sectional data that preserves both the high temporal correlations and the spatial relations in the original dataset. This is important for financial trading scenarios, where we often would like to access additional trading information, such as open, high, low, close and volume data (constituting a temporal cross-sectional dataset), rather than just a single time-series price curve dataset. Therefore, we argue that TimeGAN is the most suitable GAN-based generative model for augmenting our training dataset with realistic synthetic data.

We focus on DRL following the recent advances in deep learning. Deep architectures have superior approximation ability and thus guarantee better convergence. Advanced DRL algorithms can be categorized into three types: value-based algorithms [5, 6, 11], policy-based algorithms [7, 12, 21] and actor-critic based algorithms [2, 15, 29, 32]. Developing deep reinforcement learning in the financial trading domain has also been a popular topic in the recent research on financial asset trading [5–7, 11], portfolio optimization [12, 21, 30, 32], deep hedging [4] and market making [26, 27].

In this work, we deploy the "train on synthetic, test on real (TSTR)" evaluation method [13], to assess the usefulness of synthetic data for trading. Naturally, in order to provide fair comparative analysis of the performances achieved by different DRL agents, in our study we train the agents on both the real dataset and the augmented synthetic dataset. For clarity, in the paper, by augmented synthetic dataset we mean the large synthetic dataset with real data excluded.

3 TRADING MODEL

In this section, we formalize the single asset trading problem as a Markov Decision Process (MDP), and specify our model including assumptions and objectives. Furthermore, we discuss the setup of financial trading environment for deployment of trading agents in our experiment in the OpenAI Gym-like financial trading environment called "FinRL" [17, 18].

Single Asset Trading Problem as MDP. Due to the inherent stochastic nature of dynamic financial markets, asset prices are ever-changing and difficult to predict [30]. Given the similarities between MDPs and the process of making trading decisions in finance, we could formalize our single asset trading problem as an MDP in which an agent interacts with the environment in discrete time. The agent aims at directly taking a sequence of trading decisions without explicit price predictions. Specifically, the agent will receive observations from the environment for each timestep t, denoted as a state s. Given the state s, the agent will choose an action a, based on which the agent will receive a reward r_{t+1} at the next timestep $t+1$ and transition to the next state s'. Thus, through these repetitive interactions, a trajectory $\tau = [s_0, a_0, r_1, s_1, a_1, \ldots]$ is produced. At each timestep t, we let the expected cumulative rewards G_t with the discounting factor γ of the agent, be the objective function for the RL agent to maximize [32], i.e.:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k.$$

Single Asset Trading Environment Setup. We now specify the key components of the trading environment for deploying RL agents in our experiment. We adopt the financial trading environment in FinRL, an open-source library developed for research on DRL in trading. In FinRL, it is easy to use and deploy the "stable-baselines" RL agents for financial trading problems [10]. Thus, we utilize this library to customize our cryptocurrency and stock trading environments for the realistic trading scenario, while adding three technical indicators (see below) for better performance. Following are the detailed setups of our single asset trading environment.

State Space $s = [\mathbf{P}, \mathbf{V}, \mathbf{SMA}, \mathbf{RSI}, \mathbf{OBV}, \mathbf{M}, \mathbf{Position}]$**.** The state space in MDP is formalized as the observation space for our RL agent, and is denoted as a vector consisting of multiple features of the financial market, that provide information for the agent to make decisions on trading. Thus, we include not only the asset trading information $\boldsymbol{P}$ and $\boldsymbol{V}$ (i.e., Open, High, Low, Close, Adjusted Close prices and Volume), the current cash holding $\boldsymbol{M}$ and the position of the agent for trading $\boldsymbol{Position}$ as trading signals in our observation space, but also the following technical indicators:

(1) Simple Moving Average (**SMA**): in our trading model, SMA is calculated by taking the arithmetic average of closing prices based on the chosen rolling window size.
(2) Relative Strength Index (**RSI**): RSI is a momentum oscillator that is often used in the technical analysis for evaluating whether an asset has been overbought or oversold, by measuring the magnitude of price movements. In practice, the RSI is oscillating between 0 and 100. An asset would be considered as overbought when the RSI reaches 70, while it is assumed to be oversold when the index is below 30. It could also be used for identifying the general price trend of an asset.
(3) On-Balance Volume (**OBV**): OBV is calculated based on the changes of the asset in volume, for predicting the price movements in the future. The main idea behind this technical indicator is that if the volume of the asset has been significantly changing without incurring any sudden price movements, the price would still have jumps or falls eventually at a later time.

The size of the rolling window in our experiment is set to 12 days.

Action Space a. The action space is given by the vector consisting of buying, holding, or selling actions with a specific amount of asset. Referring to the action space defined in the FinRL environment, we use the default setting of integer amounts, with the maximum amount of the trading asset that can be bought or sold daily at 100 and limits on the initial cash at 1000000 U.S. dollars [17, 18]. Thus, trading agents make decisions from this action space with the money currently available, and short selling is not allowed. As our main objective in this paper is to justify whether GAN-based synthetic data generation method can be used for training DRL agents in daily single asset trading, we restrict to this relatively simple while realistic setup. Hence, our trading agents are expected to be rational in their decision-making and approaching to the behaviors of real human-traders in the market.

Policy $\pi(s)$. This indicates the agent's trading strategy at state s.

State transition. At each timestep, either a buying, a holding, or a selling action a associated with a specific amount of the asset, is taken as the trade decision of the DRL agent. Specifically, the trade decision is made based on the current state of the agent (i.e., restricted by the current cash holding, $\boldsymbol{M}$, and the current position of the agent for trading, $\boldsymbol{Position}$). If the trade decision is to buy or sell a specific amount of asset at timestep t, then the current available money is used for purchasing the asset at the price at the next timestep $t+1$, and also the holding assets are re-balanced. Thus, the state s transitions into the next state s', according to the historic information of the asset price and the trade decision.

Trading position. We would like our trained agent to be able to make a sequence of trade decisions. Ideally, the agent should be buying and be long when the stock price is relatively low as the trend of the price is increasing while being short when the price is likely to go down in the future. Therefore, a well-trained agent would gain profits from the price movements of the given asset by sticking with the long or short positions.

Objectives of the Trading Model. The goal of solving a single asset trading problem is to make a sequence of trade decisions that would maximize the account value at the end of the trading period. The RL task is solved by adopting the optimal strategy which maximizes the accumulated reward specified by the reward function. In our trading model, we adopt the reward function used in FinRL [17, 18], calculated as the change value of the portfolio (a.k.a. the account value) – defined as the current value of holding asset plus the cash holding – for each day. We call this reward function the accumulated differences of account value between each day.

Assumptions of the Trading Model. To reduce the complexity of our trading model while preserving the practicability of financial trading in real-world settings, we make the following assumptions.

- Market impact: as is common to much work in the area, we assume immediate execution in our trading model – that is, the specific shares of the asset can be traded fully at the closing price, without having short-term or long-term impacts on the market.
- Transaction costs: in practice, a certain amount of money is charged as transaction fees when trading in financial markets. This fee may vary depending on multiple factors, including the cryptocurrency exchange, the cryptocurrency you are trading with, the amount of trading cryptocurrency, etc. To calculate the transaction fee for our trading agents, we use the default setting in FinRL [17, 18] and thus assume the transaction cost to be charged as 0.1% of the total amount of money that is traded, for both selling and buying activities. For holding, there would be no fees incurred by the trading agents. We apply the same charge ratio to the equity market for trading stock.

4 OUR DRL AGENTS

Here, we introduce the DRL agents utilized in our trading model. As discussed earlier, they have a discrete action space, indicating whether to buy, hold, or sell a specific integer amount of asset. Therefore, and due to the compatibility across different libraries within the FinRL trading environment, we adopt the following two common DRL algorithms, which have been widely used in discrete action space settings for reinforcement learning in various domains.

Advantage Actor Critic (A2C). A2C is a classic actor-critic based algorithm in RL, which tackles the policy gradient (PG) problem by using synchronous updates of PG during the training process [19]. The A2C model consists of two parts, namely the actor network and the critic network. The actor is responsible for generating the policy as the output of its neural network, while the critic is designated to measure the performance of the action generated by the policy chosen by the actor at that state. Furthermore, the critic network introduces the so-called advantage function, used to reduce the variance of the policy network and increase the stability of the RL model. Thus, the policy network $\pi_\theta(a_t \mid s_t)$ in the A2C algorithm is updated by maximizing the following objective function [3]:

$$\nabla J_\theta(\theta) = \mathbb{E}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A(s_t, a_t)\right]$$

with $A(s_t, a_t)$ being the advantage function, defined as:

$$A(s_t, a_t) = r(s_t, a_t, s_{t+1}) + \gamma V(s_{t+1}) - V(s_t),$$

where $r(s_t, a_t, s_{t+1})$ is the reward at time t, $V(s_t)$ is the state value function in the critic neural network and γ is the discount factor. To implement the A2C algorithm in our single asset trading problem [17, 18, 23], we adopt the following actor-critic structure: a Multi Layer Perceptron (MLP) with 1 layer of 64 units as the shared global network, an MLP with 3 layers of each 128 units for the critic network, and an MLP with 2 layers of each 64 units for the actor network. For the hyperparameters, we set the learning rate as 0.0007 and the discounting factor for calculating the reward as 0.9999. We use Adam optimizer to train all neural networks.

Proximal Policy Optimization (PPO). The main idea of PPO is to control the deviation from the old policy when updating it according to the objective function in each iteration [10, 25]. The objective function of this variant can be written as follows [3, 24]:

$$L(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{A}\left(s_t, a_t\right), \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}\left(s_t, a_t\right)\right)\right]$$

where $r_t(\theta)$ denotes the ratio between the probabilities of the new policy and the previous policy, parameterized by the policy parameter θ, $\hat{A}\left(s_t, a_t\right)$ is defined as the estimation of the advantage function at timestep t, ϵ is a hyperparameter for the clip function which is often set to 0.1 or 0.2 in practice, and $\mathbb{E}_t$ can be computed as the estimated expectation value of the above terms at timestep t. Thus, the PPO algorithm ensures that the policy update is constrained within a range for each iteration in training, by introducing the clip function and taking the minimum value. Therefore, the updated policy is guaranteed to be close to the old policy, thus increasing the stability of the model.

To implement the PPO algorithm in our experiment [17, 18, 23, 25], we use an MLP with 1 layer of 64 units as the shared global network, an MLP with 3 layers of each 128 units for the critic, and an MLP with 2 layers of each 64 units for the actor. For hyperparameter settings, we use 0.00025 as the learning rate, and ϵ is assigned 0.2 as the clipping range value. The discounting factor is set as 0.9999, and Adam optimizer is used to train all neural networks.

5 DATASETS

In this section, we provide a detailed description of the two real-world financial datasets that we use in our experiments. As the nature and characteristics of trading assets may differ from one to another, we test and analyze our method in different financial markets. Specifically, for this work, we select Google stock from the traditional Equity Market and Bitcoin from the most volatile Cryptocurrency Market. We then introduce the synthetic data generation approach for data augmentation utilized in our work.

Real Datasets. We select 16 years (from 2005-06-01 to 2021-05-30) Google historical daily stock data for our experiment downloaded from Yahoo Finance, which includes Open, High, Low, Close, Adjusted close and Volume data. The whole dataset is split into the training dataset, consisting of the first 15 years of data, and the testing dataset, containing the most recent year.

Since the history of Bitcoin is much shorter compared to the Google stock, we select in total 6 years of Bitcoin historical daily data, ranging from 2015-06-01 to 2021-05-30. The split uses the first 5 years as training data and the last year as testing dataset.

Synthetic Datasets. Unlike the traditional data augmentation approaches in the area of computer vision, where real data is added to the synthetic one, in our experiments, we use only synthetic data to train the DRL agents. We name our method the synthetic data augmented reinforcement learning for trading (SDARL4T). Not only this provides a novel approach to the challenging task of training the data augmented DRL trading agents exclusively on synthetic datasets, it can also serve as the means to evaluate the quality of synthetic data generative models.

We adopt the aforementioned framework of Time-series Generative Adversarial Networks known as TimeGAN, to augment our training datasets by generating daily synthetic Google stock and Bitcoin data. As discussed earlier, we choose TimeGAN since it can be utilized for the synthetic data generation not only of time-series sequence data, but also of cross-sectional data. In order to generate realistic financial data, TimeGAN applies both supervised and unsupervised learning. Following the structure and original hyperparameter setting (unless specified) of TimeGAN in [31], we train our neural networks in the following three steps:

(1) Training phase 1: We use the autoencoder (AE) that takes a real time-series cross-sectional training dataset as input and outputs the reconstructed time-series data. The embedder and the recovery network in the AE are each built with Recurrent Neural Networks (RNN) with 3 hidden layers that each contains 24 Gated recurrent units (GRUs). We use the Mean Squared Error (MSE) to measure the reconstruction loss due to embedding and the recovery loss in the AE neural network, optimized by Adam optimizer and trained for 10000 epochs. Specifically, for generating the Google stock synthetic data, the number of time-series sequences for the input is set to 252, as the total number of trading days in a year. Since the trading period for Bitcoin is the whole year (i.e., 365 trading days) and due to computational limitations, we use 91 as the input of the number of time-series sequences in TimeGAN, which is equivalent to the total number of trading days in a quarter. Through this training phase, we obtain representation of the input in the latent space.
(2) Training phase 2: The supervised learning is conducted by using a sequence of time-series from the real dataset as input, then using the sequence of the next timestep of data as output, trying to capture the temporal information in the dataset. The supervisor network is designed as an RNN with 2 hidden layers of 24 GRUs, and trained for 10,000 epochs. Again, MSE is used for measuring the loss in the supervised learning process, optimized by Adam.
(3) Training phase 3: Here, the joint training is conducted by using both real and random noise data, combined with the first two training phases. The generator and discriminator in the GAN are both built with an RNN with 3 hidden layers of each 24 GRUs. We train it for 10000 epochs. The unsupervised loss is calculated by the MSE and binary cross entropy, optimized by Adam. In this phase of the process, the time-series synthetic data is generated. Specifically, for Google stock, we use 15 years (which is equal to 3,775 trading days) real data as the training dataset for our TimeGAN, to generate 870,912 days data in total. For Bitcoin, we use 5 years (which is equal to 1,825 trading days) of real data as the training dataset in TimeGAN to generate equivalently 151,424 days of data. These two augmented synthetic datasets are then used to train the DRL agents in our experiment.

6 EXPERIMENTAL RESULTS

In this section, we present and evaluate the results obtained from our experiment, conducted in following steps:

(1) First, we collect and preprocess the historical daily trading data of Google stock and Bitcoin, consisting of Open, High, Low, Close, Adjusted Close price and Volume data.
(2) Second, we train the aforementioned TimeGAN models with the specified hyperparameters and augment the Google stock and Bitcoin datasets using respectively trained synthetic data generators. Moreover, we use the t-Distributed Stochastic Neighbor Embedding (t-SNE) method [31] to project our 6D data (corresponding to the 6 features in our financial data) into 2 dimensions, to visualize the distributions of our real datasets and synthetic datasets, as we change the hyperparameters in the TimeGAN. This allows to visualize the distribution of our augmented synthetic data, as shown in Figure 1 for Google and Figure 2 for Bitcoin. We notice that the distribution of the synthetic data for Google captures only partially that of the real data, which may indicate that it is less realistic. For Bitcoin, instead, the synthetic data appears to capture the distribution of the real data very well.
(3) Next, we formalize the single asset trading problem as an MDP and specify the constraints and the objective of the trading model. We then deploy the A2C and PPO agents into trading, by interacting with our specified financial trading environment.
(4) Finally, we train our DRL agents on both the real and the augmented synthetic datasets, respectively. By logging and visualizing the training rewards for each agent in the TensorBoard, we get stable rewards after 50,000 (10,000, respectively) timesteps of training for Google (Bitcoin, respectively), thus indicating the convergence of the training process. This makes sense since both the real and the augmented synthetic datasets of Bitcoin are much smaller than that of Google stock. After training our DLR agents for the number of steps that guarantee convergence, we use the testing datasets for 1,000 out-of-sample instances to calculate the statistics and evaluate the agents' performance in these two financial markets.

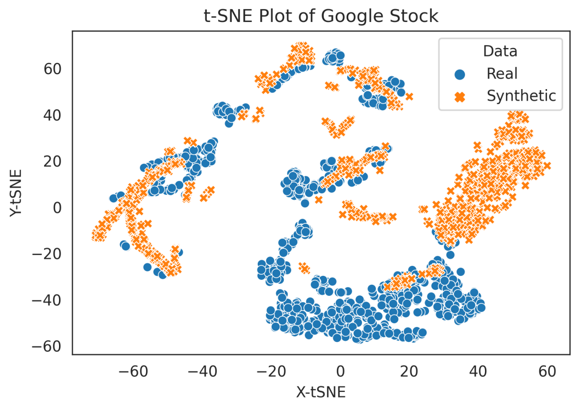

Figure 1: Visualization of the Real and Synthetic data for Google Stock.

As described in Section 3, we aim at training our agents to learn how to trade profitably, while managing risks, in both the traditional Equity Market and the much more volatile Cryptocurrency Market, where the agents start with some initial cash and are allowed to take realistic trading actions (i.e., buying, holding, or selling a specific amount of asset) in our customized FinRL trading environment. To this end, we use the annual return to measure the profitability of trading agents, as well as Sharpe and Sortino ratios to quantify the risks taken by the agents. To demonstrate our trading agents' performances, we plot the annual return, Sharpe ratio, and Sortino ratio during the 1 year out-of-sample testing period of the A2C and PPO agents, trained on both the real and TimeGAN-augmented synthetic datasets. Moreover, to gain more insight into the trading behaviors of the trained DRL agents, we also visualize the actions taken by the agents (taking 1 sample from the 1,000 times, in which we test run each agent). The results of each DRL agent in these two markets are discussed next.

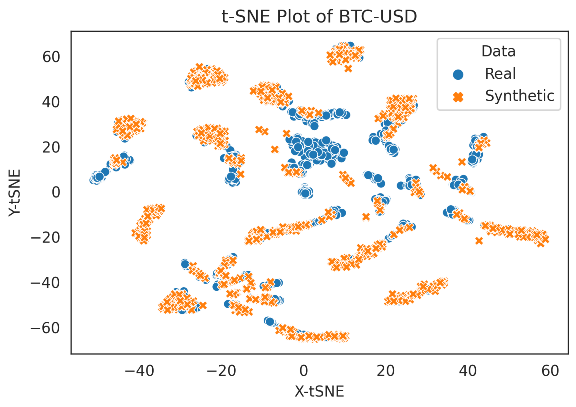

Figure 2: Visualization of the Real and Synthetic data for BTC-USD.

Performances of A2C Agents. For the A2C agents, we observe that the agent trained on the synthetic dataset surpasses the agent trained on the real dataset in both Equity and Cryptocurrency Markets, in terms of both the profitability and the undertaken risks evaluated by the annual return, Sharpe ratio, and Sortino ratio. The violin plots shown in Figure 3 and Figure 4 demonstrate the distributions of the 1,000 out-of-sample test results of the A2C agents in both Equity and Cryptocurrency Markets. It can be seen that the agents trained on the synthetic dataset have much higher annual return, as well as Sharpe and Sortino ratios, than the agents trained on the real dataset. Furthermore, the A2C agents trained on the synthetic data are more stable in the testing, as indicated by the lower variances (i.e., shown as flat shapes in the violin plots) of the aforementioned three metrics. The Descriptive Statistics (DS) results of these two A2C agents are shown in Table 1 and Table 2 (left). To summarize, using augmented synthetic data to train A2C agents leads to much greater profits and more robustness in both Equity and Cryptocurrency Markets.

Trading Strategies of A2C Agents. To better understand the differences between trading strategies selected by the A2C agents trained on the real and synthetic datasets, we visualize their trading activities in Figures 5, 6, 7 and 8. We find that the A2C agents trained

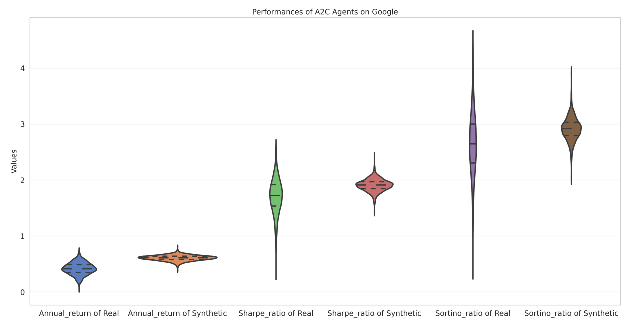

Figure 3: Violin-plot of the Profitability of A2C on Google

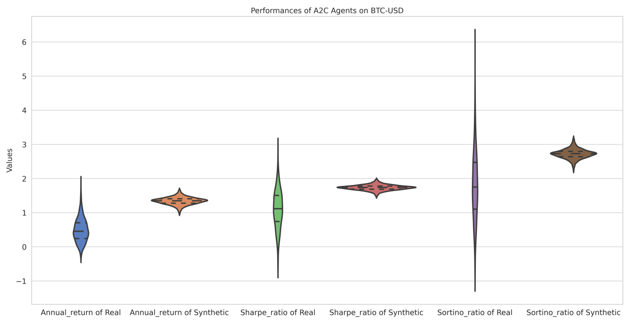

Figure 4: Violin-plot of the Profitability of A2C on Bitcoin

on the real datasets are trading more frequently than the agents trained on the synthetic datasets, for both markets. Moreover, the synthetically trained A2C agent behaves as a passive trader for the Bitcoin trading (shown in Figure 8), as only 5 active trades (buying and selling are considered as active trades here) have been made during the testing period. This agent tends to be very risk-averse, since it only trades when there are limited changes in the market price. Finally, we also find that the agent first sells Bitcoin at a higher price, then it buys it back right after at a lower price for more Bitcoins; the holding period is often short, consistently in all our 1,000 test runs. In summary, the A2C agents trained on the synthetic datasets are less active compared to the agents trained on the real datasets, potentially better predicting the price movements, with consideration of the transaction fee costs, for both markets.

Performances of PPO Agents. For the PPO agents, we observe that all four agents are profitable in trading Google stock and Bitcoin, see Figures 9 and 10. For the Cryptocurrency Market, the PPO agent trained on the synthetic dataset outperforms the agent trained on the real dataset, on average over the 1000 times out-of-sample tests, for both profitability and risk, as evaluated by the annual return, Sharpe ratio, and Sortino ratio (see Figure 10 and Table 2 (right)). The synthetically trained PPO agent for the Equity Market is slightly less profitable compared to the one trained on the real data (see Figure 9 and Table 1) (right)).

Trading Strategies of PPO Agents. We also visualize the trading activities of these four PPO agents during the testing, see Figures

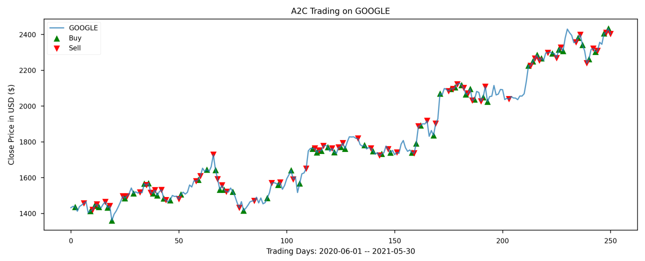

Figure 5: Visualization of A2C Trading Actions on Google close price. The green markers show the agent has made buy decisions on the Google stock, while the red ones indicate its selling actions. By visualizing the actions of the agent, we note that the A2C agent trained on the real dataset can generally capture the price movements by buying low and selling high.

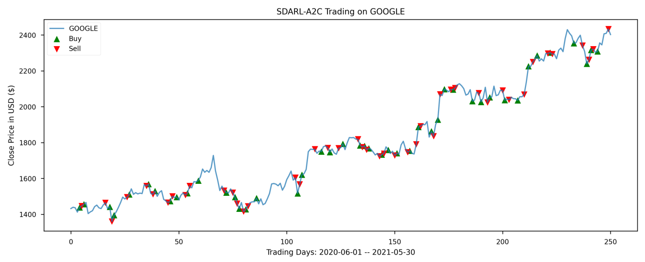

Figure 6: Visualization of SDARL-A2C Trading Actions on Google, showing that the A2C agent trained on the synthetic dataset trades less frequently than the agent trained on the real (cf. Figure 5).

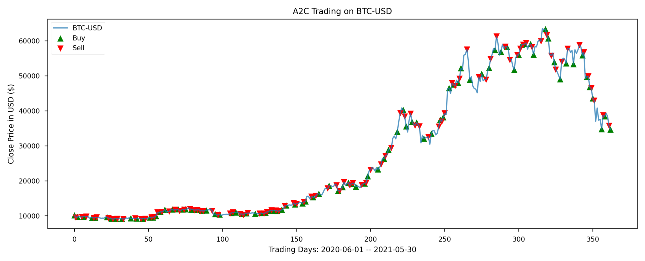

Figure 7: Visualization of A2C Trading Actions on Bitcoin. The A2C agent trained on the real dataset can capture the price movements of Bitcoin in general but trades very frequently.

11, 12, 13 and 14. On the contrary to what we have seen for the A2C agents, we find that the PPO agents trained on synthetic datasets tend to be trading more frequently than the agents trained on real datasets for both markets.

Tables 1 and 2 summarize our experimental results via descriptive statistics of the out-of-sample tests. Specifically, we assess the average performance, stability and the possible extreme behaviors of our trading agents. The A2C agents trained on the GAN-based

Table 1: Statistical Results in Equity Market

Statistics	A2C						PPO					
	Annual Return		Sharpe Ratio		Sortino Ratio		Annual Return		Sharpe Ratio		Sortino Ratio	
	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic
Mean	0.418	**0.608**	1.712	**1.907**	2.648	**2.914**	**0.615**	0.435	**1.944**	1.765	**2.996**	2.756
Std Dev	0.106	**0.048**	0.303	**0.108**	0.557	**0.204**	**0.044**	0.106	**0.095**	0.287	**0.184**	0.535
Minimum	0.053	**0.378**	0.376	**1.418**	0.511	**2.026**	**0.439**	0.104	**1.632**	0.666	**2.444**	0.924
Median	0.416	**0.610**	1.724	**1.911**	2.645	**2.919**	**0.616**	0.443	**1.945**	1.788	**2.994**	2.769
Maximum	0.734	**0.812**	**2.569**	2.440	**4.389**	3.919	**0.738**	0.710	2.307	**2.612**	3.667	**4.745**

Table 2: Statistical Results in Cryptocurrency Market

Statistics	A2C						PPO					
	Annual Return		Sharpe Ratio		Sortino Ratio		Annual Return		Sharpe Ratio		Sortino Ratio	
	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic	Real	Synthetic
Mean	0.489	**1.344**	1.106	**1.735**	1.795	**2.724**	0.303	**0.461**	0.872	**1.086**	1.478	**1.778**
Std Dev	0.342	**0.107**	0.559	**0.077**	1.000	**0.138**	**0.296**	0.332	0.653	**0.573**	1.223	**1.041**
Minimum	-0.286	**0.975**	-0.623	**1.467**	-0.796	**2.245**	-0.406	**−0.343**	-1.358	**−0.936**	-1.633	**−1.216**
Median	0.457	**1.350**	1.119	**1.739**	1.753	**2.730**	0.259	**0.412**	0.847	**1.053**	1.300	**1.672**
Maximum	**1.890**	1.668	**2.902**	1.977	**5.865**	3.177	1.590	**1.850**	**3.285**	2.847	**8.578**	6.319

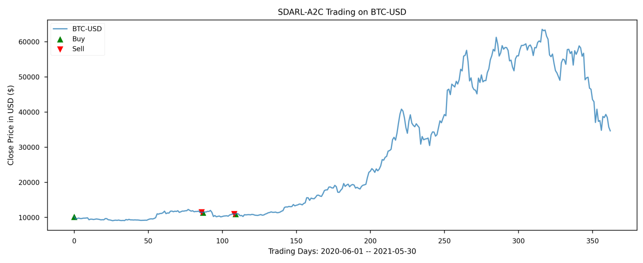

Figure 8: Visualization of SDARL-A2C Trading Actions on Bitcoin, with only 5 active trades (buying and selling are considered as active trades here). Thus, the synthetically trained A2C agent tends to be very risk-averse in trading Bitcoins, since it only trades when there are small price movements of the market price.

augmented synthetic dataset outperform the agents trained on the real datasets in both markets during the 1 year out-of-sample testing period, indicated by the higher mean and median values of the annual return, Sharpe ratio and Sortino ratio. With respect to stability and robustness, we notice that the synthetically trained A2C agents tend to be more robust in our 1000 times testing, indicated by their much lower standard deviations.

The overall performance of synthetically trained PPO agents in the Cryptocurrency Market is better than that of the agent trained on real data, as it achieves higher profits while managing the risks. While the standard deviation of the annual return is slightly higher than that of the agent trained on real data, its robustness is in general rather good, as the values are very small and close to each other. The only less profitable synthetically trained agent in our experiments is the PPO agent for trading Google stock, see Table 1.

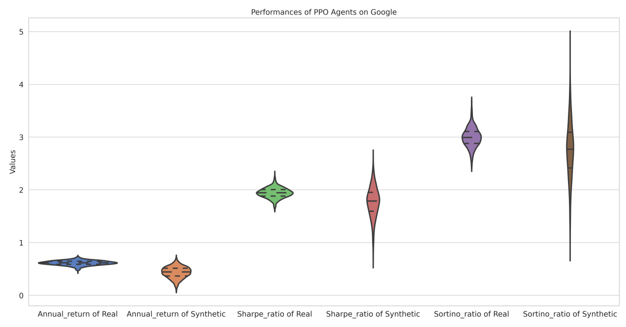

Figure 9: Violin-plot of the Profitability of PPO on Google

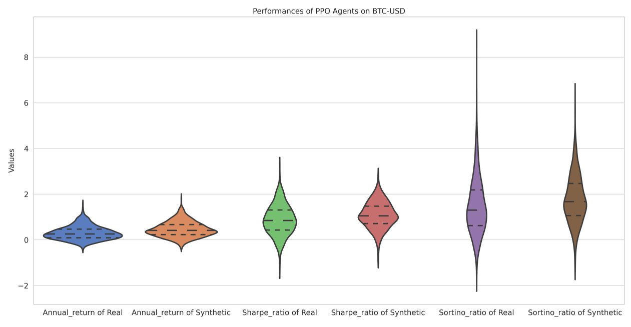

Figure 10: Violin-plot of the Profitability of PPO on Bitcoin

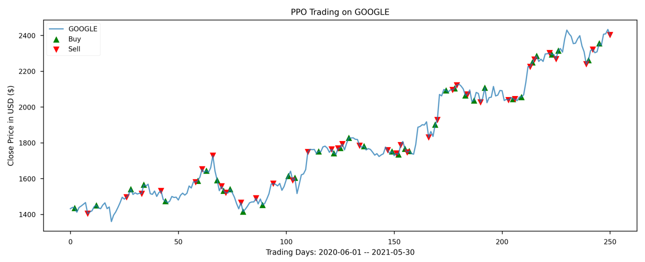

Figure 11: Visualization of PPO Trading Actions on Google. The PPO agent trained on the real data can follow the Google stock price movements by taking sensible actions. We note that the holding time between each active trade tends to be longer than the A2C agents.

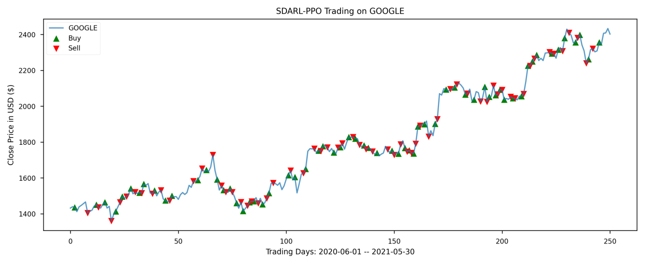

Figure 12: Visualization of SDARL-PPO Trading Actions on Google, showing the agent trained on the synthetic dataset has a similar trading behavior vis-a-vis the PPO agent trained on the real data but tends to trade more frequently on Google stock.

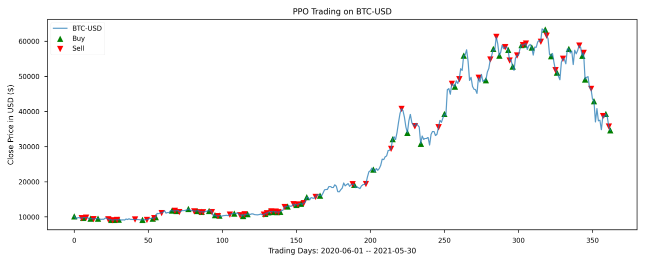

Figure 13: Visualization of PPO Trading Actions on Bitcoin, which shows that the PPO agent trained on the real Bitcoin dataset trades more often when the price movements are small (left-hand side of the plot), while it trades less frequently when there are price jumps (as on the right-hand side of the plot).

In summary, in our experiments, the A2C agents trained on synthetic datasets show great performances in both Equity and Cryptocurrency Markets, in terms of the generalization ability, average performance and stability. The fact that the synthetically-trained PPO agent for the Google stock has worse performances might be due to the quality of the TimeGAN augmented synthetic dataset of Google, which seems to be less realistic especially when

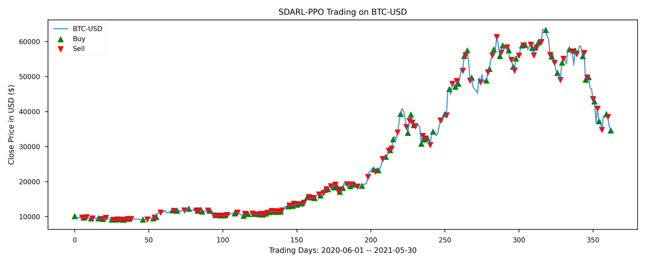

Figure 14: Visualization of SDARL-PPO Trading Actions on Bitcoin. The PPO agent trained on the synthetic dataset shows similar trading behavior as the one trained on real data, while trading more frequently.

compared to Bitcoin's (as shown in the aforementioned t-SNE plots). Interestingly, the A2C agent seems to be less sensitive and more robust to the quality of the synthetic dataset, since the synthetically trained A2C agent on Google still shows better results. Therefore, it would be interesting to examine other DRL agents to see whether a particular approach is more robust and best suited for our method.

Our experiments confirm that DRL agents for daily single asset trading can be trained solely on the GAN-based augmented synthetic datasets, without the need to directly access the market data. Moreover, the better robustness gained by the agents can be linked to the data augmentation, while the quality of synthetic data only matters for PPO agents. Remarkably, A2C agents seem to do well even without super-realistic synthetic data for training.

7 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel approach, namely synthetic data augmented reinforcement learning for trading. We combine the GAN-based synthetic data generation model with classic reinforcement learning algorithms, to investigate the feasibility and evaluate the performance of our approach on a single asset daily trading problem in Equity and Cryptocurrency Markets.

Our experimental results suggest that training DRL agents on augmented synthetic data for financial trading is a promising direction in developing efficient, profitable agents, without direct access to the market data. In fact, our synthetic data augmented reinforcement learning agent can make comparable or even higher profits than the agents trained on the real data, in terms of higher annual return, Sharpe ratio and Sortino ratio.

We also evaluate the stability of the agents trained by our proposed method, which shows better robustness than of those trained on the real data. In conclusion, our work supports the deployment of deep reinforcement learning agents in financial trading and shows the applicability of our approach in practical settings.

To this end, alternative combinations of generative models for synthetic data augmentation with other deep reinforcement learning algorithms for financial trading should be explored, which opens a new line of research on AI in finance.

ACKNOWLEDGMENTS

Carmine Ventre acknowledges funding from the UKRI Trustworthy Autonomous Systems Hub (EP/V00784X/1).

REFERENCES

[1] Samuel Assefa. 2020. Generating synthetic data in finance: opportunities, challenges and pitfalls. *Challenges and Pitfalls (June 23, 2020)* (2020).

[2] Stelios D Bekiros. 2010. Heterogeneous trading strategies with adaptive fuzzy actor–critic reinforcement learning: A behavioral approach. *Journal of Economic Dynamics and Control* 34, 6 (2010), 1153–1170.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym. arXiv:arXiv:1606.01540

[4] Hans Buehler, Lukas Gonon, Josef Teichmann, Ben Wood, Baranidharan Mohan, and Jonathan Kochems. 2019. Deep hedging: hedging derivatives under generic market frictions using reinforcement learning. *Swiss Finance Institute Research Paper* 19-80 (2019).

[5] Lin Chen and Qiang Gao. 2019. Application of deep reinforcement learning on automated stock trading. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 29–33.

[6] Quang-Vinh Dang. 2019. Reinforcement learning in stock trading. In *International Conference on Computer Science, Applied Mathematics and Applications*. Springer, 311–322.

[7] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. 2016. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems* 28, 3 (2016), 653–664.

[8] Chris Donahue, Julian McAuley, and Miller Puckette. 2018. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208* (2018).

[9] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. 2017. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633* (2017).

[10] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines.

[11] Gyeeun Jeong and Ha Young Kim. 2019. Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications* 117 (2019), 125–138.

[12] Zhengyao Jiang and Jinjun Liang. 2017. Cryptocurrency portfolio management with deep reinforcement learning. In *2017 Intelligent Systems Conference (IntelliSys)*. IEEE, 905–913.

[13] James Jordon, Jinsung Yoon, and Mihaela Van Der Schaar. 2018. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International conference on learning representations*.

[14] Adriano Koshiyama, Nick Firoozye, and Philip Treleaven. 2019. Generative adversarial networks for financial trading strategies fine-tuning and combination. *arXiv preprint arXiv:1901.01751* (2019).

[15] Jinke Li, Ruonan Rao, and Jun Shi. 2018. Learning to trade with deep actor critic methods. In *2018 11th International Symposium on Computational Intelligence and Design (ISCID)*, Vol. 2. IEEE, 66–71.

[16] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael Wellman. 2020. Generating realistic stock market order streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 727–734.

[17] Xiao-Yang Liu, Hongyang Yang, Qian Chen, Runjia Zhang, Liuqing Yang, Bowen Xiao, and Christina Dan Wang. 2020. FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. *Deep RL Workshop, NeurIPS 2020* (2020).

[18] Xiao-Yang Liu, Hongyang Yang, Jiechao Gao, and Christina Dan Wang. 2021. FinRL: Deep reinforcement learning framework to automate trading in quantitative finance. *ACM International Conference on AI in Finance (ICAIF)* (2021).

[19] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1928–1937.

[20] Olof Mogren. 2016. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904* (2016).

[21] John Moody and Matthew Saffell. 2001. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks* 12, 4 (2001), 875–889.

[22] Tidor-Vlad Pricope. 2021. Deep Reinforcement Learning in Quantitative Algorithmic Trading: A Review. *arXiv preprint arXiv:2106.00123* (2021).

[23] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. 2021. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research* 22, 268 (2021), 1–8. http://jmlr.org/papers/v22/20-1364.html

[24] John Schulman, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel. 2015. Trust Region Policy Optimization (TRPO). *CoRR abs/1502.05477* (2015).

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[26] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market making via reinforcement learning. *arXiv preprint arXiv:1804.04216* (2018).

[27] Thomas Spooner and Rahul Savani. 2020. Robust market making via adversarial reinforcement learning. *arXiv preprint arXiv:2003.01820* (2020).

[28] Magnus Wiese, Robert Knobloch, Ralf Korn, and Peter Kretschmer. 2020. Quant GANs: deep generation of financial time series. *Quantitative Finance* 20, 9 (2020), 1419–1440.

[29] Zhuoran Xiong, Xiao-Yang Liu, Shan Zhong, Hongyang Yang, and Anwar Walid. 2018. Practical deep reinforcement learning approach for stock trading. *arXiv preprint arXiv:1811.07522* (2018).

[30] Hongyang Yang, Xiao-Yang Liu, Shan Zhong, and Anwar Walid. 2020. Deep reinforcement learning for automated stock trading: An ensemble strategy. *Available at SSRN* (2020).

[31] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. 2019. Time-series Generative Adversarial Networks. *Advances in Neural Information Processing Systems* 32 (2019), 5508–5518.

[32] Zihao Zhang, Stefan Zohren, and Stephen Roberts. 2020. Deep reinforcement learning for trading. *The Journal of Financial Data Science* 2, 2 (2020), 25–40.