Comparing Normalization Methods for Portfolio Optimization with Reinforcement Learning

Omitted due to double blind review process.

Omitted due to double blind review process.

Abstract. Recently, reinforcement learning has achieved remarkable results in various domains, including robotics, games, natural language processing, and finance. In the financial domain, this approach has been applied to tasks such as portfolio optimization, where an agent continuously adjusts the allocation of assets within a financial portfolio to maximize profit. Numerous studies have introduced new simulation environments, neural network architectures, and training algorithms for this purpose. Among these, a domain-specific policy gradient algorithm has gained significant attention in the research community for being lightweight, fast, and for outperforming other approaches. However, recent studies have shown that this algorithm can yield inconsistent results and underperform, especially when the portfolio does not consist of cryptocurrencies. One possible explanation for this issue is that the commonly used state normalization method may cause the agent to lose critical information about the true value of the assets being traded. This paper explores this hypothesis by evaluating two of the most widely used normalization methods across three different markets (IBOVESPA, NYSE, and cryptocurrencies) and comparing them with the standard practice of normalizing data before training. The results indicate that, in this specific domain, the state normalization can indeed degrade the agent's performance.

Keywords: Reinforcement learning · Portfolio optimization · Quantitative finance · Normalization Methods.

1 Introduction

Portfolio optimization is a task in which an agent constantly defines the percentage of each asset in a financial portfolio considering market trends that can be identified through quantitative analyses, political and financial news, public opinion, among others [5]. With the advent of algorithmic trading, computational agents have been increasingly utilized to perform such task, and, thanks to the emergence of Machine Learning techniques, several supervised approaches have been applied to forecast the future value of assets in the portfolio and assist the decision-making [7]. Those approaches, however, are very sensitive to real-world features of the financial market, such as brokerage fees [3], and, thus, a technique that maximizes the future return of the portfolio considering these characteristics is more appropriate.

In this context, the Reinforcement Learning (RL) technique is a good fit: the agent repeatedly interacts with the environment and learns the optimal behavior by trial and error, being guided by a reward function, which is more positive when the agent's actions produce better results [19]. If the reward function is related to the profit made by the agent after it performs its actions, the RL formulation conveniently adheres to the calculation of the future return of the financial portfolio and it can be optimized with the usage of several well-known training algorithms [4].

This idea was applied in [8], which introduced a framework to optimize financial portfolios composed of cryptocurrencies and created a realistic formulation that simulated the effects of the market considering transaction costs. By using a convolutional neural network called *Ensemble of Identical Independent Evaluators* (EIIE) to process the time series of the price of the cryptocurrencies and developing a domain-specific policy gradient algorithm, the framework outperformed several other classical portfolio strategies.

The approach utilized in [8] had a huge impact in the research field and was the foundation of several other works. [14] changed the Deep Learning architecture and applied a multi-scale convolution inspired by Inception networks to create the *Ensemble of Identical Independent Inception* (EI^3) architecture, which outperformed EIIE. [18] went further, and developed an agent that made use of graph neural networks to model the relationship between assets in the portfolio, introducing *DeepPocket*, which generates more profit than the convolutional solutions. [21], on the other hand, applied Transformer architectures to the framework in order to account for the non-Euclidean characteristics of the temporal series of the assets and also achieved great results.

The large adoption of the framework introduced in [8] is a result of the fact that the domain-specific training algorithm converges faster and leads to better policies than other well-known algorithms such as *Deep Deterministic Policy Gradient* (DDPG) and *Proximal Policy Optimization* (PPO) [10]. However, recent studies point that the framework is not as reliable as it appears in stock markets: its performance is considerably worse in comparison to the cryptocurrency market [9]. One of the reasons that can be generating this disparity is the state regularization method utilized, whose effectiveness was not investigated by the scientific community in this specific use-case.

To mitigate this issue, this paper compares the performance of three normalization methods that can be used in portfolio optimization agents trained with reinforcement learning. Two of them are state normalizations commonly used in the field: the normalization by the last closing price and the normalization by the last value. The other technique consists of normalizing the time series of assets independently before the training process. This work aims to answer the following research questions:

1. Using data normalization instead of state normalization improves the performance of the agent in stock markets?
2. What is the effect of applying data normalization in cryptocurrency portfolios such as the one introduced in [8]?

This paper is organized as follows. Section 2 presents the mathematical formulation of the portfolio optimization task. Section 3 explains reinforcement learning and how it applies to the problem at hand. Section 4 highlights the role of normalization and outlines a few applicable techniques. Sections 5 and 6 describe the experiments conducted to test the techniques and their results. Finally, Section 7 concludes the document.

2 Mathematical Formulation of Portfolio Optimization

In the portfolio optimization task, at each time step t and for a portfolio of n assets, the agent is responsible for defining the *portfolio vector* or *weights vector* $\boldsymbol{W_t} \in \mathbb{R}^{n+1}$ whose elements are the rate (or weight) of each asset in the portfolio composition. Considering $w_{t,i}$ the i-th element of the vector, it is adopted that $w_{t,0}$ refers to the uninvested money while the other elements are related to the assets of the portfolio. Therefore, the following conditions must be met:

$$0 \leq w_{t,i} \leq 1, \qquad \sum_{i=0}^{n} w_{t,i} = 1. \tag{1}$$

At each time step t, there is also a price vector $\boldsymbol{P_t} \in \mathbb{R}^{n+1}$ whose elements are the prices of every component in the portfolio. Like in $\boldsymbol{W_t}$, it is considered that $p_{t,0}$ refers to the remaining cash of the portfolio and, thus, $p_{t,0} = 1$ and $p_{t,i}$, in which $i \in \{1, 2, ..., n\}$, are calculated with respect to $p_{t,0}$.

During every simulation step, the price of the assets in the portfolio change, modifying the portfolio vector from $\boldsymbol{W_t}$ to $\boldsymbol{W_t^f}$ through the following equation:

$$\boldsymbol{W_t^f} = \frac{(\boldsymbol{P_t} \oslash \boldsymbol{P_{t-1}}) \odot \boldsymbol{W_t}}{(\boldsymbol{P_t} \oslash \boldsymbol{P_{t-1}}) \cdot \boldsymbol{W_t}}, \tag{2}$$

in which $\oslash$ denotes the element-wise division, $\odot$ is the element-wise multiplication and $\cdot$ represents the dot product of vectors.

In the beginning of step t, the value of the portfolio is defined as V_t, but, just like the portfolio vector, it changes to V_t^f due to the variation of the prices of the assets. Then, in the next time step, a new portfolio vector $\boldsymbol{W_{t+1}}$ is set by the agent continuing the rebalancing cycle. This process is calculated as

$$V_t^f = V_t\Big(\boldsymbol{W_t} \cdot (\boldsymbol{P_t} \oslash \boldsymbol{P_{t-1}})\Big), \qquad V_{t+1} = \mu_{t+1} V_t^f, \tag{3}$$

in which $\mu_{t+1} \in [0, 1]$ is a factor that reduces the portfolio value, simulating the effects of the transaction costs from the rebalancing ($\boldsymbol{W_t^f} \rightarrow \boldsymbol{W_{t+1}}$). Details about the calculation of μ_{t+1} can be found in [8].

The simulation, then, continues, following the same equations presented before until the simulation period is finished. In the initial conditions, no money is invested in the assets, so that $\boldsymbol{W_0} = [1, 0, 0, ..., 0]$ and the agent aims to discover the sequence of portfolio vectors $[\boldsymbol{W_1}, \boldsymbol{W_2}, ..., \boldsymbol{W_T}]$ that maximizes the final value of the portfolio V_T^f, in which T is the final time step of the simulation being run.

3 Reinforcement Learning

The Reinforcement Learning is a technique in which an agent interacts with an environment that provides, at each time step t, observations O_t with data to be used in the decision-making. The agent, then, defines a state S_t, which summarizes its current situation and is utilized by the agent to choose an action A_t to be performed in the environment. After that, the environment transitions to another state and returns a new observation O_{t+1} and a reward R_{t+1} which is a numerical value that is bigger when the action performed leads to a favorable outcome. The process continues until a terminal state is achieved (or indefinitely, if such a state does not exist) and the agent's objective is to maximize the expected return of the rewards. Therefore, the agent learns continuously interacting with the environment by trial and error.

In order to transform a state S_t into an action A_t, the agent utilizes a policy of actions, which is a function $\pi : S \rightarrow A$ that maps all the possible states to all the possible actions. Due to the complexity of real-world problems, in which states and actions can usually be represented by multidimensional vectors of continuous values, neural networks [6] are commonly used as the policy of the agent since it can approximate close values without the need to divide the state space or the action space in small intervals of defined size [19]. The policy that provides the agent with the maximum expected return (and thus, the policy that the agent is trying to learn) is called the *optimum policy*.

3.1 Applying to Portfolio Optimization

In order to apply the RL approach to the portfolio optimization task, it is necessary to define the states, actions, rewards and the policy that the agent will use.

State: In this work, the agent utilizes the time series of closing, high and low prices of the assets in the portfolio since those are the most important features [20]. Therefore, the state is composed of a three-dimensional matrix of shape (f, n, t), in which $f = 3$ is the number of features, n is the number of assets in the portfolio and t is time window considered.

Action: The action of the agent is the portfolio vector introduced in Section 2.

Reward: The reward at time step t is the logarithmic rate of return. It can be calculated with the following equation:

$$r_t = ln\Big(V_t^f / V_{t-1}^f\Big). \tag{4}$$

Policy: For all the experiments of this work, the EIIE architecture introduced in [8] is utilized, since it is fast and is a common baseline in the research field. This architecture make use of convolutional filters in order to independently process the time series of the assets using the same model. An interesting feature of the EIIE policy is that, in order to be able to infer the effects of the transaction costs in the decision-making and improve the performance, it also considers the last performed action, so that $A_t = \pi(S_t, A_{t-1})$.

3.2 Policy Gradient for Portfolio Optimization

Several RL training algorithms can be utilized to train a portfolio optimization agent, but, according to [10], the domain-specific policy gradient (PG) algorithm introduced in [8] achieves better performance in comparison to other general algorithms.

In the PG algorithm, the agent interacts with the environment and saves its experiences in a replay buffer so that it contains one experience for each time step t in the simulation episode (an episode can be defined as the set of simulation steps needed to cover the training period defined by the input historical data). When the buffer is completely filled, the agent starts to sample sequential batches of data: being Δt the batch size chosen, a sequential batch is composed of experiences from the period $[T, T + \Delta t]$, in which T is the first time step of the batch.

The batches of data are, then, utilized to update the parameters θ of the agent's policy π in order to maximize the objective function given by:

$$\sum_{t=T_1}^{T_2} \frac{ln(\mu_t(\boldsymbol{W_t} \cdot (\boldsymbol{P_t} \oslash \boldsymbol{P_{t-1}}))}{T_2 - T_1 + 1}, \tag{5}$$

in which T_1 is the initial time step of the batch used and T_2 is the last one. Note that this objective function is directly obtained by Equation 3, so that the agents policy is optimized to maximize the profit in the batch of experiences.

Since the price variation of the assets in the training data does not change, the PG algorithm directly updates the experiences of the replay buffer after the optimization without the need to constantly interact with the environment, considerably accelerating the training process.

Finally, it is important to highlight that the agent is able to perform *online learning*, so that new experiences can be added to the replay buffer and used by the agent to adapt itself to unknwon data. This is one of the biggest advantages in comparison to supervised approaches [3], since the agent can be continuously trained while still on production.

4 Normalization Methods and its Advantages

Normalizing the inputs of a deep neural network is an essential step to improve its performance [16]: using normalized data prevents that numerical inputs with different scales degrade the learning process. For portfolio optimization, the normalization methods utilized in the research can be classified in two categories.

4.1 State Normalization

This normalization technique is based on the process introduced in [12] and normalizes the data by dividing it by a value during the creation of the state. Let $\boldsymbol{S}$ be the current agent's state and let $\boldsymbol{X_t}$ be the closing prices, $\boldsymbol{X_t}^{(hi)}$

the high prices and $\boldsymbol{X_t}^{(lo)}$ the low prices vector of every asset in $\boldsymbol{S}$, in which $t \in \{1, 2, \ldots, T\}$ is a time step of the time window of size T. In this work, two types of state normalization are being investigated:

By last closing price: In this approach, all the prices are divided by the last closing price. This was the technique utilized in the article that introduced the formulation utilized in this paper [8] and its application can be seen in the equation below.

$$\boldsymbol{X_t} = \boldsymbol{X_t} \oslash \boldsymbol{X_T}, \quad \boldsymbol{X_t}^{(hi)} = \boldsymbol{X_t}^{(hi)} \oslash \boldsymbol{X_T}, \quad \boldsymbol{X_t}^{(lo)} = \boldsymbol{X_t}^{(lo)} \oslash \boldsymbol{X_T}. \quad (6)$$

By last price: This technique was utilized in [14] and [15] and divides all the prices by its last value. The equation below demonstrates how to apply it:

$$\boldsymbol{X_t} = \boldsymbol{X_t} \oslash \boldsymbol{X_T}, \quad \boldsymbol{X_t}^{(hi)} = \boldsymbol{X_t}^{(hi)} \oslash \boldsymbol{X_T}^{(hi)}, \quad \boldsymbol{X_t}^{(lo)} = \boldsymbol{X_t}^{(lo)} \oslash \boldsymbol{X_T}^{(lo)}. \quad (7)$$

Several variations of this type of normalization have been applied in several other articles [17,18], but the core is the same: the state space represents the rate of change of the input time series in the time window, which is adherent to the objective function of Equation 5. However, this approach makes the agent lose information about the true value of the assets so that it only learns trading strategies that act when the prices are falling or booming. Therefore, this might prevent the agent to, for example, identify that an asset is undervalued and buy it (or sell it, if it is overvalued). Figure 1 contains a simple example of how information can be lost using this type of normalization.

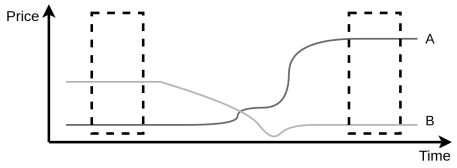

Fig. 1. A demonstration of how the agent may lose information when using state normalization. Since the state focuses on the rate of variation of the assets, stocks A and B, in both time windows (dotted rectangles), are represented by the same time series due to their stability. Thus, the agent cannot detect that stock B is undervalued.

4.2 Data Normalization

This method, on the other hand, consists of normalizing the time series of the prices of assets independently before using it in the training process. Several time series normalization techniques can be used [16] but, since the prices of assets

need to be positive, simply dividing the whole time series by its maximum absolute value is an effective approach and, thus, the focus of this work. Therefore, for a temporal series $y(t)$, the transformed series can be calculated as

$$y'(t) = \frac{y(t)}{max(\{y(t) : t \in \mathcal{T}\})}, \tag{8}$$

in which $\mathcal{T}$ is the set of possible values for t.

Unlike state normalization, this method provides the agent with information about the current asset value, which can be compared with the price history to guide actions: if the price is close to 0, the asset may be undervalued, and buying it might be an interesting investment.

However, since normalization is first fit to the training data, asset prices can exceed 1 during testing, especially in volatile booming markets. As the agent was trained with values between 0 and 1, this may lead to unreliable behaviors and underperformance. This issue can be mitigated with online learning, a feature of the PG algorithm introduced in Section 3.2, making this normalization still suitable for this work's use case.

5 Experiments

In order to verify the performance of the agent using different markets and different normalization methods, three portfolios composed of high-volume assets have been constructed:

1. A portfolio of stocks of the American market (NYSE) composed of 11 assets: BAC, F, RF, WFC, GE, PFE, C, T, MRO, X, JPM. This is considered a very stable market.
2. A portfolio composed of 10 Brazilian stocks (IBOVESPA): VALE3, PETR4, ITUB4, BBDC4, BBAS3, RENT3, LREN3, PRIO3, WEGE3, ABEV3. In comparison to NYSE, this market is considerably more volatile.
3. A cryptocurrency portfolio composed of 9 coins: ADA, BNB, BTC, BTG, DOGE, ETH, LINK, TRX, XRP. Between the three markets, this is the one with the biggest volatility.

The historical data is obtained in two different sources. For NYSE and IBOVESPA, the price of the stocks is taken from *Yahoo Finance website*[1]. The cryptocurrencies' historical prices, on the other hand, were obtained from a dataset available in *Kaggle*[2]. In both cases, the data is composed of daily closing, high and low prices of the components of the portfolio.

[1] Yahoo Finance can be accessed in `https://finance.yahoo.com/`.

[2] The dataset can be found in `https://www.kaggle.com/datasets/svaningelgem/crypto-currencies-daily-prices`.

5.1 Procedure

For each normalization method introduced in Section 4, 50 training and testing processes are run. In the NYSE and IBOVESPA portfolios, the agent is trained between 2011/11/11 and 2019/12/31 and tested during 2020 (January 1st to December 31st). This test period was selected because it posed a challenge to the market, as the COVID-19 pandemic shook stock prices worldwide. Thus, we are testing a worst-case scenario for the agent.

Due to the fact that cryptocurrencies are new investments, in the third portfolio, more recent periods were chosen in order to have enough training data: the training period is from 2018/01/01 and 2022/12/31 and the testing one is from 2023/01/01 to 2023/12/31.

For all the experiments, the same hyper-parameters were utilized both during the training and the test process. Table 1 summarizes the hyper-parameters used.

Table 1. Hyper-parameters utilized in the experiment.

Hyperparameter	**Value**	**Description**
Learning rate	0.00005	The step size used in the AdamW optimizer.
Batch size	200	Size of the sequence of experiences sampled.
Sample bias	0.002	Probability of the sampling geometric distribution.
Steps	300000	Number of training steps used to train the agent.
Online Steps	30	Training steps run after every rebalancing during tests.
Time window	50	Size of the time window considered in the state.
Commission rate	0.0025	Commission fee rate applied.
Initial value	100000	Initial value of the portfolio in BRL (for IBOVESPA) and USD (for NYSE and cryptocurrencies).

5.2 Performance Metrics

To quantify the performance of the agent, three metrics are utilized in this work:

Final Accumulative Portfolio Value (FAPV): This metric calculates the ratio between the final portfolio value and the initial portfolio value:

$$FAPV = V_T^f / V_0, \tag{9}$$

in which V_T^f is the final value of the portfolio and V_0 is the initial one.

Maximum Drawdown (MDD) [11]: The maximum drawdown is a method to quantify the risk of the portfolio. It represents the biggest loss of the rebalancing during a period of time. The bigger the MDD, the more risky is the trading strategy applied. It can be calculated as:

$$MDD = \max\left(\max_{t<\tau} \frac{V_t^f - V_\tau^f}{V_t^f}\right). \tag{10}$$

Sharpe Ratio (SR) [13]: Finally, the Sharpe ratio is a metric that quantifies the return of the portfolio divided by its standard deviation. The bigger the standard deviation of the returns, the more volatile is the portfolio strategy and, thus, more risky. Therefore, the SR evaluates the amount of profit generated considering the risk of the rebalancing. This metric is given by the following equation:

$$SR = \frac{\mathbb{E}_t[\rho_t - \rho_F]}{\sqrt{var_t[\rho_t - \rho_F]}}, \tag{11}$$

in which $\rho_t = V_t^f / V_{t-1}^f$ is the return ratio of the portfolio in step t and ρ_F is the return ratio of the risk-free investment (in this work, $\rho_F = 0$).

Good portfolio strategies produce high FAPV and SR, and MDD close to 0.

6 Results

The experiments[3] described in Section 5 were conducted using *RLPortfolio* [2], an open source library for training portfolio optimization agents using reinforcement learning. The results are discussed below.

6.1 In the American Market

Table 2 demonstrates the aggregated results of the 50 runs conducted. It can be noted that using data normalization produces the best outcome in terms of FAPV and SR, showing that the agent is benefiting from the information about the true value of the assets. With respect to the MDD metric, the normalization by the last price achieves the best performance but, considering its margin of error, it is very similar to the data normalization. All these results indicate that to normalize the data before training is more effective in this market.

Table 2. Mean results of the NYSE portfolio.

Normalization Method	FAPV	MDD	SR
Last Closing Price	0.76 ± 0.06	0.60 ± 0.04	-0.005 ± 0.006
Last Price	1.05 ± 0.10	**0.52 ± 0.04**	0.018 ± 0.010
Data Normalization	**1.19 ± 0.08**	0.55 ± 0.02	**0.033 ± 0.006**

It is interesting to highlight the poor performance of the normalization by the last closing price, which can be seen in details in Figure 2. The majority of the runs using this technique produces suboptimal policies that make the agent lose money ($FAPV < 1$). The other state normalization strategy, on the other hand,

[3] The code utilized can be found in `https://anonymous.4open.science/r/norm_portfolio-3DD2`.

generates a more heterogeneous distribution, which makes it the most unreliable: it can produce agents that have considerable losses but also ones that multiply the portfolio value by 2.

Finally, Figure 2 also shows that the data normalization method typically produces two policies: one where the agent preserves its value and another where it multiplies it by 1.5. The distribution also has lower variance, indicating this normalization is more reliable.

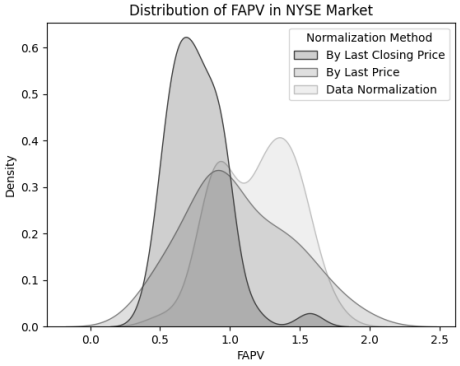

Fig. 2. Approximate distribution of the results in the NYSE market.

6.2 In the Brazilian Market

In the Brazilian market, it is observed a pattern similar to the one noticed in the American market but, since the Brazilian market is more volatile, the differences in the metrics are more noticeable when comparing. As it can be seen in Table 3, the data normalization method achieves the higher performance in both FAPV and SR metrics, indicating that it is the most suitable for this market. The main downside of this method, however, can be observed in the MDD metric, which reveals that it leads to more risky agents.

Table 3. Aggregated results of the IBOVESPA portfolio.

Normalization Method	**FAPV**	**MDD**	**SR**
Last Closing Price	1.48 ± 0.16	0.57 ± 0.02	0.051 ± 0.010
Last Price	1.26 ± 0.16	**0.55 ± 0.02**	0.040 ± 0.008
Data Normalization	**1.79 ± 0.10**	0.7 ± 0.04	**0.069 ± 0.006**

Figure 3 demonstrates the approximate distribution of the FAPV metric along the runs and it is noticeable that, just like in the American market, the data

normalization is more consistent. But it can also be noticed that the behavior of both state normalizations have switched: this time, the normalization by last closing price has the bigger variance instead of the one by last price. This fact evidences that, when using a state normalization method, it is important to choose it wisely and test several possibilities because, depending on the market, one method behaves considerably different than the other.

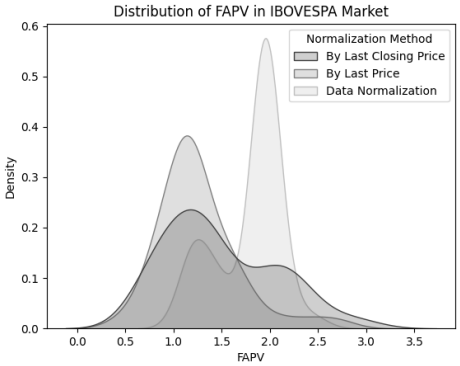

Fig. 3. Approximate distribution of the results in the IBOVESPA market.

6.3 In the Cryptocurrency Market

Curiously, the cryptocurrency market, the one utilized in the paper that introduced the formulation adopted in this work [8], is the one which benefited the most by the data normalization method. As it can be seen in Table 4, the performance of the agent when using this method largely outperforms the other ones in all considered metrics.

Table 4. Aggregated results of the cryptocurrency portfolio.

Normalization Method	**FAPV**	**MDD**	**SR**
Last Closing Price	0.78 ± 0.10	0.54 ± 0.04	-0.025 ± 0.012
Last Price	0.70 ± 0.10	0.56 ± 0.04	-0.034 ± 0.012
Data Normalization	**1.78 ± 0.04**	**0.44 ± 0.02**	**0.059 ± 0.002**

It can also be noticed that the state normalization technique is poorly effective, since the majority of the test runs yields losses in the portfolio value and, thus, produces an agent that loses money. As demonstrated in Figure 4, the two state normalization methods generate similar distributions, meaning that the choice between them is considerably irrelevant.

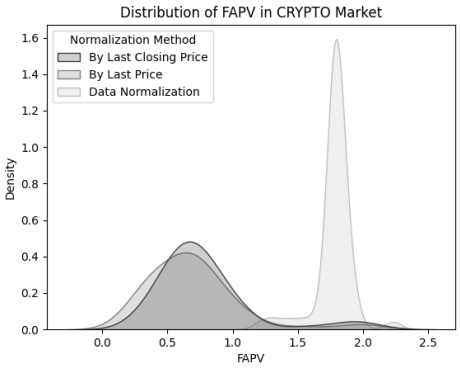

Fig. 4. Approximate distribution of the results in the cryptocurrencies market.

It is interesting to discuss the possible issues that made the cryptocurrency portfolio underperform in this experiments while the portfolio used in [8] achieved great results. The first issue is the fact that the portfolios utilized are not the same: since the original article does not provide the list of assets in their portfolio, this paper simply constructed a portfolio with high volume coins in order to adhere to the formulation presented in 2. Additionally, this article makes use of daily historical prices, which can be easily found in several databases on-line, while [8] generates a portfolio that rebalances its assets every 30 minutes. Finally, since the input data of this work has smaller granularity, more data points are necessary and, thus, a bigger time period is used to train and test the data. The time series utilized are closer to the present time, in which the cryptocurrency market is a little more stable than in [8]: in the latter, the agent is trained and tested in periods of the years of 2017 and 2018, when several cryptocurrencies were surging and booming while, in this work, the test period is the year of 2023.

The possible influence of these changes in the input data underlines that the approach introduced in [8] is not as general as it seems and that optimizations in the hyperparameters, normalization methods and even changes in the objective function might be necessary when applied to different markets.

6.4 Analysis of the Results

As it can be seen in Sections 6.1, 6.2 and 6.3, the positive effects of the data normalization are bigger the bigger the volatility of the target market. From Figure 2 to Figure 4, it is noticeable that the overlap of the distributions of the FAPV gets smaller, showing that the volatility of the market actually degrades the performance of the agent under the state normalization.

Additionally, training logs show that state normalization is more prone to overfitting. In the cryptocurrency market, for instance, most runs yield agents whose performance deteriorates over training, as shown in Figure 5 (generated

with *Tensorboard* [1]), which illustrates the test performance during training in the worst-case scenario (i.e., the run with the most overfitting). A similar behavior occurs with data normalization, but the agent converges to a much better local minimum, as seen in Figure 3.

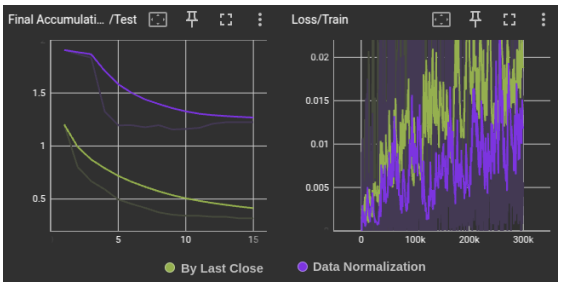

Fig. 5. Example of logs of runs in which overfit occurs. In the left, the graph shows the FAPV over 15 performance tests conducted in the test period while the agent was being trained. In the right, the loss function in the training process is plotted for every step. The data is smoothed to outline trends.

The bigger tendency of overfitting in the state normalization can be justified by the fact that it simplifies the state space more than the data normalization. While the latter keeps information about the true value of the assets, the other one reduces the observations of the agent to price variations so that different situations of the market can be considered the same, as explained in Section 4. Therefore, different moments of the market in the testing period generate states that are approximately equal to states in the training period, so that the agent's deterministic policy generates suboptimal actions. The more the training process proceeds, the more the agent specializes in the training period, generating worse actions when testing.

Finally, it is important to highlight that, even though the data normalization is more reliable statistically, all the normalization methods eventually produced good agents. Therefore, it is possible to run several training processes, choose the best agent considering its performance in a validation period and, then, test it. However, the more unreliable the normalization method, the more difficult it is to find a good agent, the reason why the data normalization technique is favored. Table 5 contains the maximum FAPV achieved by the experiments described in Section 5 and it can be noticed that there is no clear pattern to identify which method yields the best result across multiple markets.

7 Conclusion

This article empirically demonstrates that, in portfolio optimization, normalizing the input data rather than the RL agent's state improves performance in both

Table 5. Max FAPV for each market and normalization method.

Normalization Method	NYSE	IBOVESPA	CRYPTO
Last Closing Price	1.57	**2.94**	2.06
Last Price	**1.96**	2.71	2.06
Data Normalization	1.74	2.47	**2.23**

stock and cryptocurrency markets. The experiments show that more volatile markets benefit more from data normalization, as it avoids the information loss inherent to state normalization.

Although data normalization consistently outperforms state normalization, the latter can still produce effective agents. However, its high variability often slows the training process policies since it leads to worse suboptimal policies more frequently. Moreover, the best state normalization method may vary by market, so it should be treated as a hyperparameter and tuned accordingly.

Finally, the strong performance of data normalization suggests that online learning can compensate for the agent's lack of exposure to input ranges beyond the training data. Future work could explore this trade-off, examining how online learning rates and update steps affect agents using data normalization.

Acknowledgments

Omitted due to the double blind review process. Omitted due to the double blind review process. Omitted due to the double blind review process. Omitted due to the double blind review process. Omitted due to the double blind review process.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation. pp. 265–283. OSDI'16, USENIX Association, USA (Nov 2016)
2. Costa, C.d.S.B., Costa, A.H.R.: RLPortfolio: Reinforcement Learning for Financial Portfolio Optimization. In: Paes, A., Verri, F.A.N. (eds.) Intelligent Systems. pp. 412–426. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-79035-5_29
3. Deng, Y., Bao, F., Kong, Y., Ren, Z., Dai, Q.: Deep Direct Reinforcement Learning for Financial Signal Representation and Trading. IEEE Transactions on Neural Networks and Learning Systems **28**(3), 653–664 (Mar 2017). https://doi.org/10.1109/TNNLS.2016.2522401
4. Felizardo, L.K., Paiva, F.C.L., Costa, A.H.R., Del-Moral-Hernandez, E.: Reinforcement Learning Applied to Trading Systems: A Survey (Nov 2022). https://doi.org/10.48550/arXiv.2212.06064

5. Gunjan, A., Bhattacharyya, S.: A brief review of portfolio optimization techniques. Artificial Intelligence Review (Sep 2022). https://doi.org/10.1007/s10462-022-10273-7
6. Haykin, S.S.: Neural Networks: A Comprehensive Foundation. Prentice Hall (1999)
7. Henrique, B.M., Sobreiro, V.A., Kimura, H.: Literature review: Machine learning techniques applied to financial market prediction. Expert Systems with Applications **124**, 226–251 (Jun 2019). https://doi.org/10.1016/j.eswa.2019.01.012
8. Jiang, Z., Xu, D., Liang, J.: A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem (Jul 2017). https://doi.org/10.48550/arXiv.1706.10059
9. Li, J.: A Deep Reinforcement Learning Framework For Financial Portfolio Management (Sep 2024). https://doi.org/10.48550/arXiv.2409.08426
10. Liang, Z., Chen, H., Zhu, J., Jiang, K., Li, Y.: Adversarial Deep Reinforcement Learning in Portfolio Management (Nov 2018). https://doi.org/10.48550/arXiv.1808.09940
11. Magdon-Ismail, M., Atiya, A.F., Pratap, A., Abu-Mostafa, Y.S.: On the maximum drawdown of a Brownian motion. Journal of Applied Probability **41**(1), 147–161 (Mar 2004). https://doi.org/10.1239/jap/1077134674
12. Ross, S., Mineiro, P., Langford, J.: Normalized Online Learning (May 2013). https://doi.org/10.48550/arXiv.1305.6646
13. Sharpe, W.F.: The Sharpe Ratio. The Journal of Portfolio Management **21**(1), 49–58 (Oct 1994). https://doi.org/10.3905/jpm.1994.409501
14. Shi, S., Li, J., Li, G., Pan, P.: A Multi-Scale Temporal Feature Aggregation Convolutional Neural Network for Portfolio Management. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management. pp. 1613–1622. ACM, Beijing China (Nov 2019). https://doi.org/10.1145/3357384.3357961
15. Shi, S., Li, J., Li, G., Pan, P., Chen, Q., Sun, Q.: GPM: A graph convolutional network based reinforcement learning framework for portfolio management. Neurocomputing **498**, 14–27 (Aug 2022). https://doi.org/10.1016/j.neucom.2022.04.105
16. Singh, D., Singh, B.: Investigating the impact of data normalization on classification performance. Applied Soft Computing **97**, 105524 (Dec 2020). https://doi.org/10.1016/j.asoc.2019.105524
17. Soleymani, F., Paquet, E.: Financial portfolio optimization with online deep reinforcement learning and restricted stacked autoencoder—DeepBreath. Expert Systems with Applications **156**, 113456 (Oct 2020). https://doi.org/10.1016/j.eswa.2020.113456
18. Soleymani, F., Paquet, E.: Deep graph convolutional reinforcement learning for financial portfolio management – DeepPocket. Expert Systems with Applications **182**, 115127 (Nov 2021). https://doi.org/10.1016/j.eswa.2021.115127
19. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA (2018)
20. Weng, L., Sun, X., Xia, M., Liu, J., Xu, Y.: Portfolio trading system of digital currencies: A deep reinforcement learning with multidimensional attention gating mechanism. Neurocomputing **402**, 171–182 (Aug 2020). https://doi.org/10.1016/j.neucom.2020.04.004
21. Xu, K., Zhang, Y., Ye, D., Zhao, P., Tan, M.: Relation-Aware Transformer for Portfolio Policy Learning. In: Twenty-Ninth International Joint Conference on Artificial Intelligence. vol. 5, pp. 4647–4653 (Jul 2020). https://doi.org/10.24963/ijcai.2020/641