Concurrent Computing for Accelerating Financial Machine Learning Model Training

Thiago S. Araújo[1], Cristiano A. Künas[1], Dennis G. Balreiras[1],
Arthur F. Lorenzon[1], Philippe O. A. Navaux[1], Paulo S. G. de Mattos Neto[2],
[1]Informatics Institute, Federal University of Rio Grande do Sul (UFRGS) — Porto Alegre, Brazil
[2]Informatics Center, Federal University of Pernambuco (UFPE) — Recife, Brazil

{tsaraujo, cakunas, dgbalreira, aflorenzon, navaux}@inf.ufrgs.br, psgmn@cin.ufpe.br

***Abstract*—The financial market generates a large volume of data daily, allowing the increasing use of machine learning algorithms in building predictive models for the stock market. In this environment, time is a crucial factor since stock prices change daily, so the training time of models is a critical factor. This paper proposes a method to optimize the overall training time of 5 reinforcement learning algorithms that predict the weights of each stock in a stock portfolio. Experiments were conducted by varying the number of algorithms executed simultaneously. In addition, the computational characteristics of each algorithm were analyzed concerning the use of memory and processing. From the proposed combination of running the algorithms concurrently, it was possible to reduce the total training time by 33% compared to running the algorithms sequentially. Moreover, this execution led to a commendable 15% decrease in energy consumption.**

***Index Terms*—Reinforcement learning; Performance and energy consumption; Stock market**

I. Introduction

The financial market is essential for trading stocks, currencies, and options. It is essential for the economy, as it facilitates companies and governments in acquiring capital through shares and bond issuance, which, in turn, funds investments that foster economic growth. In this scenario, due to the vast amount of data generated daily within the financial market, high-performance computing (HPC) becomes essential to process information swiftly. Consequently, the abundance of data has paved the way for employing machine learning algorithms in this domain to predict market movements, price fluctuations, and stock prices [1].

Training machine learning algorithms with extensive data sets is inherently time-consuming, often taking even days to complete. Although these machine-learning approaches are trained using historical data, they do not interact with market dynamics. Consequently, they can struggle to adapt to extreme and unpredictable scenarios, such as the drastic fall in the stock market during the onset of the COVID-19 pandemic. As a result of this limitation and the challenge of adapting models, the financial sector has begun to explore innovative solutions.

One of these promising approaches is Reinforcement Learning (RL), which has received increasing attention for its potential to revolutionize the optimization of financial portfolios. RL makes models more dynamic and adaptive, allowing them to learn from their own experiences and adjust their strategies in response to changing market conditions [2]. This allows algorithms to make better decisions in line with market movement, even in the face of unprecedented events [3].

In this scenario, exploring the simultaneous execution of training algorithms is an interesting alternative. Instead of running the training process of a single algorithm, concurrent computing makes it possible to run the training process of several algorithms [4]. By adopting concurrent computing, the computing power of several cores can be harnessed simultaneously, increasing training efficiency.

However, executing many reinforcement algorithms concurrently over the same data does not guarantee the best overall performance. Hardware and software aspects may prevent linear improvements when increasing the number of concurrent algorithms. Hardware-related issues include CPU bottlenecks when the algorithms are CPU intensive or competition for shared resources, such as caches, buses, or interconnects. This may lead to contention and reduce the potential for linear improvements. Furthermore, on the software side, load imbalance, communication overhead, and resource allocation may limit linear improvements.

Given the above scenario, this paper explores ways of running reinforcement learning algorithms concurrently to reduce training time when considering stock market data. By considering five well-known learning algorithms in the literature (PPO, A2C, DDPG, SAC, and TD3), we investigate different scenarios of combinations of these algorithms running on data from eight stocks on a modern multi-core architecture with 40 cores. Through a large set of experiments, we show that:

- Each learning algorithm behaves differently regarding CPU and memory usage. These characteristics affect the performance of each algorithm when running them simultaneously.
- A conservative scenario, with only two reinforcement learning algorithms running simultaneously, offers better overall performance than a more aggressive scenario.
- The best combination of simultaneous execution of learning algorithms found through an exhaustive search can improve training time by 33% while reducing energy consumption by 15% compared to the standard way of executing reinforcement learning algorithms.

The remainder of this paper is organized as follows. In Section II, we describe the related work. In Section III, the methodology used is listed. Then, we discuss the performance and energy results in Section IV. Finally, we draw the conclusions and future works in Section V and the acknowledgment in Section VI.

II. Related Work

This section is divided into subsections according to the proximity between the works. At the end of this section, we discuss the contributions of this work.

A. Machine learning applied to the stock market

Several strategies using machine learning have been applied. Least-square support vector machine (LS-SVM) was used to predict daily stock prices, and the Particle Swarm Optimization (PSO) algorithm was used to optimize (LS-SVM) [5]. Regression and Long short-term memory (LSTM) networks were used to predict the stock prices, opening, closing, maximum, minimum prices, and trading volume utilized [6]. Machine learning classifiers were applied to perform prediction in the stock market, using stock data, social networks, and news, where the random forest obtained the best result [7]. A random forest and an artificial neural network were used to predict the next day's closing stock price [8]. The application of reinforcement learning has been growing in the context of the stock market. The survey in [9] shows the recent advances of reinforcement learning in the financial market where two reinforcement learning approaches, Gated Deep Q-learning (GDQN) and Gated Deterministic Policy Gradient (GDPG), were applied to stock trading. A trading ensemble strategy was created using three algorithms: Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Deterministic Policy Gradient (DDPG). This strategy utilized the best features of each algorithm, achieving better performance than the individual algorithms and two baselines [10].

B. Concurrent computing applied to reinforcement learning

A method using concurrent federated reinforcement learning was proposed for the problem of resource allocation in edge computing. The addition of concurrency in the decision-making approach brought benefits on the global scale of resource allocation. The results showed improved speed and resource utilization [11]. Cloud computing requires automated provisioning and de-provisioning based on demand. Algorithms are used to define instances according to the requests received. Defining the number of requests that are processed in parallel is a challenging task. The application of reinforcement learning to find the best configuration was investigated, and the results show an increase in performance when using this algorithm [12]. A study on using reinforcement learning techniques applied to dynamic task scheduling was conducted, and a comparison of these techniques was made [13].

C. Our contributions

Unlike the works discussed in the previous subsections, this work employs concurrent computing to optimize the training phase of reinforcement learning algorithms applied to the stock market. Therefore, the main contributions of the state-of-the-art are as follows: a performance analysis of reinforcement learning algorithms applied to the stock market and a performance and energy evaluation of distinct scenarios where concurrent computing could be used to improve the training of these algorithms.

III. Methodology

We have selected five reinforcement learning algorithms:

- **Proximal Policy Optimization (PPO)** performs multiple updates in the policy without significant changes, which helps to stabilize the learning process [14].
- **Advantage Actor-Critic (A2C)** has two networks, the actor to learn policy and the critic to estimate state-value function.
- **Deep Deterministic Policy Gradient (DDPG)** can handle continuous action spaces using a deterministic policy. Also, the experience replay and target networks help to stabilize the learning process and improve sample efficiency [15].
- **Soft Actor-Critic (SAC)** maximizes the expected cumulative reward while also explicitly maximizing the entropy of the policy, leading to better exploration in high-dimensional continuous action spaces [16].
- **Twin Delayed Deep Deterministic Policy Gradient (TD3)** uses two critic networks to reduce overestimation bias and the delayed updating of the actor-network to improve stability. TD3 addresses the problem of overestimation in the Q-value estimates by using the minimum of the target critic networks.

The chosen algorithms have different characteristics regarding the optimization process, such as their approach to policy optimization, handling of value functions, exploration strategies, and use of replay buffers. Thus, by applying each of them, it is possible to identify which characteristics are better for optimizing the weights of a stock portfolio. The algorithms also have distinct CPU and memory usage behaviors: the on-policy algorithms (PPO, A2C) have lower memory usage and moderate CPU usage. On the other hand, the off-policy algorithms (DDPG, SAC, and TD3) are CPU intensive and use more memory than on-policy algorithms to store the replay buffer.

A. Execution Environment and Input Data

The experiments were performed on a multicore architecture with *2x* Intel Xeon E5-2650 v3 Haswell, with 20 physical cores (40 with HyperThreading), and 128GB of RAM. We used the Linux Ubuntu Operating System with kernel v.5.6.0. Each application was interpreted with Python 3.7.1 and executed with the DVFS governor set to *performance* as it is the standard choice in HPC servers. To get the execution time, we used the *Times* package, and the energy consumption was extracted using the *CodeCarbon* package.

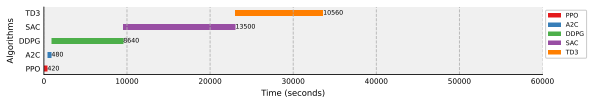
To train the five algorithms, we took as input the adjusted closing price data of eight stocks obtained from the financial library[1] from 2010 to 2017. We divided the data into two sets, 80% for train and 20% for test [2]. The following stocks were used: Apple, General Electric, JPMorgan, Microsoft, Vodafone, Nike, NVIDIA, and 3M Company.

B. Set of experiments

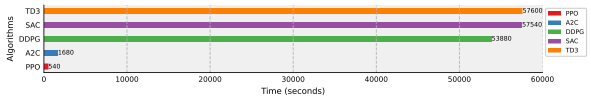
We have organized the set of experiments in five different scenarios:

- **Scenario I**: All learning algorithms are executed in sequential order, one after another. In this scenario, every algorithm can use all available hardware resources without sharing them.

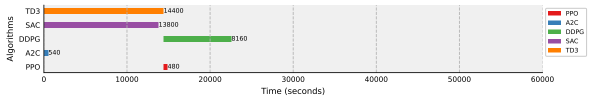
[1]https://github.com/yahoo-finance/yahoo-finance



(a) Execution behavior of the training time in Scenario I.



(b) Execution behavior of the training time in Scenario V.



(c) Best combination of algorithms found by an exhaustive search.

Fig. 1. Timeline comparison of three different scenarios.

- **Scenario II**: Two learning algorithms are deployed to execute concurrently at once. With that, one can observe the two more suitable algorithms for sharing hardware resources.
- **Scenario III**: The same as Scenario II, but considering three learning algorithms.
- **Scenario IV**: The same as Scenario II, but considering four learning algorithms.
- **Scenario V**: All learning algorithms are concurrently executed at once.

To manage the execution of Scenarios II, III, IV, and V, we rely on the Processing package. It is a package for the Python Language that supports the spawning of multiple processes through the threading model from the standard library. Thus, when multiple learning algorithms are deployed for execution, they will share the available hardware resources.

IV. Experimental Evaluation

This Section discusses the performance and energy results of running each scenario described in Section III-D. For that, we start by evaluating the behavior of each learning algorithm in Section IV-A. Then, we discuss in Section IV-B the results.

A. *Performance of each Reinforcement Learning Algorithm*

Let us start by discussing the results for Scenario I, where all the algorithms are executed one after another. Hence, Figure 1(a) depicts the execution time for each algorithm, given in seconds. As observed, PPO is the algorithm with the shortest execution time, while SAC took more time to execute among the five algorithms. Different reasons explain the behavior of each learning algorithm, as discussed next.

Algorithm Complexity: Each algorithm has its own underlying optimization strategies, network architectures, and update procedures. For example, TD3 involves twin Q-networks for better stability, target policy smoothing, and delayed updates, which make it computationally more expensive when compared to the other learning algorithms. **Sample Efficiency:** Algorithms that are more sample-efficient might require fewer interactions with the environment to achieve good performance. PPO is known for being relatively sample-efficient. **Update Frequency:** Algorithms may differ in how often they update their policy or value functions. A2C updates the policy and value functions synchronously, which can lead to more frequent updates than asynchronous methods. TD3, on the other hand, performs multiple updates for each interaction with the environment due to delayed Q-network updates, contributing to the higher execution time. **Exploration vs. Exploitation:** Exploration strategies impact the number of environmental interactions required to learn an effective policy. TD3 and DDPG are algorithms that require more exploration using target policy smoothing, leading to slower initial convergence compared to PPO and A2C, which have a more deterministic exploration strategy. **Exploration Noise:** The algorithms that use exploration noise, like DDPG and SAC, usually need additional environmental interactions to learn an effective policy, contributing to the execution time. **Target Networks and Delayed Updates:** Learning algorithms like DDPG and TD3 employ target networks and delayed updates, leading to a slower convergence than other methods.

For the evaluation performed in the following sections, we consider the results of this scenario as the **baseline** since this is the standard way reinforcement learning algorithms are executed in multicore architectures. The total time to train and generate the prediction model is the sum of each algorithm, resulting in 33600s (about 9.3 hours).

TABLE I
Performance results for each combination of algorithms

Combination of Algorithms	Sequential	Concurrent	Speedup
PPO, A2C	900s	420s	**2.14**
PPO, DDPG	9060s	8160s	1.15
PPO, SAC	13920s	10980s	1.27
PPO, TD3	10980s	9120s	1.20
A2C, DDPG	9120s	8100s	1.12
A2C, SAC	13980s	9780s	1.43
A2C, TD3	11040s	8820s	1.25
DDPG, SAC	22140s	14040s	1.58
DDPG, TD3	19200s	12720s	1.51
SAC, TD3	24060s	13860	1.73
PPO, A2C, DDPG	9540s	8160s	1.16
PPO, A2C, SAC	14400s	9900s	1.45
PPO, A2C, TD3	11460s	9780s	1.17
PPO, DDPG, SAC	22560s	15600s	1.44
PPO, DDPG, TD3	19620s	14580s	1.34
A2C, DDPG, SAC	22620s	16200s	1.39
A2C, DDPG, TD3	19680s	12900s	1.52
A2C, SAC, TD3	24540s	14400s	**1.70**
DDPG, SAC, TD3	32700s	63360s	0.51
PPO, A2C, DDPG, SAC	23040s	15060s	1.52
PPO, A2C, SAC, TD3	24960s	15120s	**1.65**
PPO, A2C, DDPG, TD3	20100s	13380s	1.50
PPO, DDPG, SAC, TD3	33120s	67920s	0.48
A2C, DDPG, SAC, TD3	33180s	65700s	0.50

B. *Optimizing the Execution of Reinforcement Learning Algorithms through Concurrent Execution*

This Section discusses the benefits of employing concurrent execution of learning algorithms to reduce the total training time. For that, Table I depict the execution time of all combinations of learning algorithms. The *sequential* column represents the time to execute the algorithms one after another, while the *concurrent* column is the time to execute the algorithms concurrently. Also, the *speedup* column highlights the performance improvements of the concurrent over the sequential execution. Hence, the higher this value, the better. We discuss each scenario separately next, along with the best solution found through an exhaustive search that tries all possible combinations of reinforcement learning algorithms.

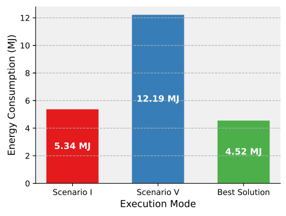


Fig. 2. Energy consumption of each Scenario shown in Figure 1.

1) Scenario II: The results for this scenario are shown in Table I. The first observation is that simultaneous execution brings performance benefits in all combinations of learning algorithms. In the best result, simultaneous execution of the PPO and A2C algorithms optimizes performance in *2,14*. These two algorithms consume fewer resources than the others, so sharing resources greatly improves performance. However, there are situations in which simultaneous performance is similar to sequential execution. We observed this situation for the combination of A2C and SAC, with a speed increase of only 1.12. This behavior occurs because the two algorithms are more memory-intensive.

2) Scenario III: When three algorithms are executed concurrently, the competition for shared resources increases, leading to more penalties for the total execution time of the concurrent execution than Scenario II, as shown in Table I. In this scenario, exploiting the execution of A2C, SAC, and TD3 algorithms concurrently can deliver the best performance improvements over the sequential execution (1.70 of speedup). These three algorithms present complementary CPU and memory usage characteristics, improving execution time. For example, A2C is an asynchronous algorithm that each agent operates independently and synchronously, making it suitable for concurrent execution. On the other hand, when three memory-intensive algorithms are deployed to execute concurrently (DDPG, SAC, and TD3).

3) Scenario IV: Table I shows the execution time when four algorithms are run simultaneously. As can be seen, the training time has been optimized in some cases. In the most significant situation, the speedup is 1.65. On the other hand, when the algorithms have similar CPU and memory requirements, it is not possible to improve performance by exploiting simultaneous computing (PPO, DDPG, SAC, TD3).

4) Scenario V: Different from all previous scenarios, the execution time of all training runs increases by a factor of 1.71 compared to sequential execution, as shown in Figure 1. This behavior occurs because each algorithm creates 40 threads, totaling 200 threads, and excess threads only add overhead in thread management and competition for shared resources. For this reason, this scenario had the worst result.

5) Best Solution: Figure 1 shows the execution behavior of Scenario I (the standard way learning algorithms are executed), Scenario V (the worst results), and the best combination found through the exhaustive search. Figure 2 has the energy consumption of these three behaviors. As observed, the total execution time is reduced by 33% when the best possible combination of the five algorithms is applied. By being able to optimize the use of shared resources, also reduced the total energy consumption by 15% compared to the baseline (Scenario I) and 63% compared to Scenario V.

V. Conclusions

We have investigated distinct scenarios of concurrent computing to optimize the performance and energy consumption of financial machine learning model training. These algorithms play an important role nowadays as they predict the weight of stocks within a stock portfolio. Through extensive experiments, we have shown that selecting the ideal combination of algorithms to execute concurrently leads to significant performance and energy improvements over the common practice adopted by software developers and end-users: 33% reductions in the execution time and 15% energy savings. In future work, we intend to exploit heterogeneous architectures to execute even more algorithms simultaneously and improve the training step of such models.

VI. Acknowledgment

This study was partially supported by CNPq/MCTI/FNDCT - Universal 18/2021 under grants 406182/2021-3 and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

[1] B. M. Henrique, V. A. Sobreiro, and H. Kimura, "Literature review: Machine learning techniques applied to financial market prediction," *Expert Systems with Applications*, vol. 124, pp. 226–251, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741741930017X

[2] R. Durall, "Asset allocation: From markowitz to deep reinforcement learning," 2022.

[3] T. G. Fischer, "Reinforcement learning in financial markets - a survey," FAU Discussion Papers in Economics, Nürnberg, FAU Discussion Papers in Economics 12/2018, 2018. [Online]. Available: http://hdl.handle.net/10419/183139

[4] O. Y. Al-Jarrah, P. D. Yoo, S. Muhaidat, G. K. Karagiannidis, and K. Taha, "Efficient machine learning for big data: A review," *Big Data Research*, vol. 2, no. 3, pp. 87–93, 2015.

[5] O. Hegazy, O. S. Soliman, and M. A. Salam, "A machine learning model for stock market prediction," *arXiv preprint arXiv:1402.7351*, 2014.

[6] I. Parmar, N. Agarwal, S. Saxena, R. Arora, S. Gupta, H. Dhiman, and L. Chouhan, "Stock market prediction using machine learning," in *2018 first international conference on secure cyber computing and communication (ICSCCC)*. IEEE, 2018, pp. 574–576.

[7] W. Khan, M. A. Ghazanfar, M. A. Azam, A. Karami, K. H. Alyoubi, and A. S. Alfakeeh, "Stock market prediction using machine learning classifiers and social media, news," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–24, 2020.

[8] M. Vijh, D. Chandola, V. A. Tikkiwal, and A. Kumar, "Stock closing price prediction using machine learning techniques," *Procedia computer science*, vol. 167, pp. 599–606, 2020.

[9] B. Hambly, R. Xu, and H. Yang, "Recent advances in reinforcement learning in finance," *Mathematical Finance*, vol. 33, no. 3, pp. 437–503, 2023.

[10] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid, "Deep reinforcement learning for automated stock trading: An ensemble strategy," in *Proceedings of the first ACM international conference on AI in finance*, 2020, pp. 1–8.

[11] Z. Tianqing, W. Zhou, D. Ye, Z. Cheng, and J. Li, "Resource allocation in iot edge computing via concurrent federated reinforcement learning," *IEEE Internet of Things Journal*, vol. 9, no. 2, pp. 1414–1426, 2021.

[12] L. Schuler, S. Jamil, and N. Kühl, "Ai-based resource allocation: Reinforcement learning for adaptive auto-scaling in serverless environments," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 804–811.

[13] C. Shyalika, T. Silva, and A. Karunananda, "Reinforcement learning in dynamic task scheduling: A review," *SN Computer Science*, vol. 1, pp. 1–17, 2020.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[15] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[16] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.