An Ablation Study of Actor-Critic Architecture for Community Detection in Dynamic Social Networks

Author 1[1], Author 2[1], and Author 3[1]

Institute 1
author1@institute1, author2@institute1, author3@institute1

Abstract. This article presents an ablation study of the Actor-Critic Architecture for Community Detection (AC2CD) developed upon Deep Reinforcement Learning (DRL) and Graph Attention Networks (GAT). In this work, the ablation study method is based on the explainable artificial intelligence approach, including execution time, memory, and GPU usage to assess the AC2CD performance. The dataset used in the experiments includes real-world data of an email network between members of a European research institution (Email-Eu) with 1,005 nodes, 25,571 edges, and 42 communities available on the Stanford Snap Project. The three hyperparameters used to analyze the architecture execution are the learn_rate, batch_size, and n_games, varying from 10%, 30%, 50%, and 70%. With the achieved experimental results, we aim to find a set of hyperparameters with optimal balance contributing to analysis that might interest the DRL and GAT community.

Keywords: Ablation study · AC2CD · Hyperparameters.

1 Introduction

Undoubtedly, Machine Learning (ML) is no longer far from the reality of the Artificial Intelligence (AI) community and the current society. Especially, Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL), subfields of ML area that refer to learning how to make decisions sequentially while being influenced by the environment become mature in the past years [1]. The RL goal is to map situations to actions that maximize a numerical reward signal indicating how well the agent performs tasks. Agents learn through trial and error, adjusting their actions to achieve the highest possible reward. And DRL integrates deep learning into RL techniques to train an agent.

Given the advancement in research and the diverse applications of ML, specifically RL, in various fields, including scientific and commercial domains, it becomes imperative to understand the impact of selecting specific components and parameters for developing an ML system, given its complexity. Therefore, a compelling approach to address this issue is to conduct an Ablation Study (AS) [2]. Indeed, the technique of AS has gained significant attention in the field of ML in recent years, as presented in [3].

The findings of this study can inform future research on optimizing actor-critic architectures and potentially lead to the development of automated HPs tuning techniques. Furthermore, investigating the impact of learning strategies and other components on the optimization process contributes to the broader understanding of DRL models and their transformation towards based Explainable Artificial Intelligence (XAI) [4].

Therefore, to understand the DRL advancements and the significance of AS in AI, this article presents an AS approach using an actor-critic architecture developed upon DRL and Graph Attention Networks (GAT) called AC2CD [5]. The AS aims to analyze and identify the components that significantly influence the AC2CD algorithm when executing a real-world dataset of an email network between members of a European research institution (Email-Eu) presented in Section 4. We evaluate the AC2CD performance by empirically modifying specific Hyperparameters (HPs) within the algorithm.

This work contribution is to present an AS with the AC2CD architecture, shedding light on the influential factors that contribute to the algorithm's performance. The insights gained from this study contribute to the ongoing efforts to enhance the efficiency and effectiveness of DRL algorithms in real-world applications in the XAI direction.

The rest of the manuscript presents in Section 2 an overview of concepts, definitions, and methods used in this work, in Section 3 related work, in Section 4 experimental method adopted in the AS, in Section 5 experimental results with discussions, and finally, in Section 6 conclusion and directions for future work.

2 Background

The background presents in Section 2.1 an overview of AI concepts focusing on ML, RL, DRL, and actor-critic applied to the community detection problem with the AC2CD architecture. Section 2.2 presents the AS technique.

2.1 AI Overview

According to [1], the AI field is vast, encompassing various domains of knowledge such as engineering, pharmacy, biology, medicine, and many others. Currently, AI has branched out and formed subfields such as ML, culminating in new aspects of understanding such as XAI [6]. Since 1959, [7] defines ML as a field of study that gives computers the ability to learn without being explicitly programmed. ML aims to emulate human intelligence through learning based on the parameters of the environment and context in which the machine is embedded [8]. Traditionally, AI can be divided into three main areas of study: supervised learning, unsupervised learning, and RL.

RL is defined by [1] as a type of ML that operates through rewards given by the model to the learner for each correct learning instance. The objective is to learn mapping situations to actions in order to maximize the accumulated reward. There are two essential characteristics: trial-and-error search refers to

the learner's attempts throughout the algorithm involving trial and error to subsequently receive a reward upon successful execution; the delayed reward refers to the consequences of the agent's learning, which not only determine the immediate reward but also the next state of the environment and future rewards [9]. According to [10], DRL is an RL approach combined with deep learning employed when decisions become too complex for RL alone, and a neural network is used to estimate states instead of mapping all possible solutions. This allows for a more manageable solution space in the decision-making process.

With the growth of ML approaches, various domains of knowledge have benefited, however, systems using AI have become complex and difficult to understand and explain. As a result, a new approach to AI-based systems has emerged with the goal of providing explainability to human users, highlighting the strengths and weaknesses of the algorithm, and conveying an understanding of how it will behave in the future. According to [6], XAI enables greater transparency and interpretability in complex AI systems allowing users' trust and permitting humans to make informed decisions while effectively cooperating with such systems. By providing explanations for algorithmic decisions, XAI bridges the gap between the black-box nature of traditional AI and the human need for comprehensibility. This enhances the usability and ethical considerations of AI applications across various domains.

Actor-Critic are temporal difference (TD) learning methods representing the policy function independent of the value function. The policy function returns a probability distribution over the actions that agents can take based on the provided state or the strategy agents employ to achieve a goal. On the other hand, the value function determines the expected return for an agent starting in a particular state and continually acting under a specific policy [1, 11]. In the actor-critic method, the actor is responsible for deciding which action to take. The critic provides feedback to the actor on the quality of the action and how it can be adjusted to achieve the goal [12]. In short, the actor-critic is a hybrid architecture combining value-based and policy-based methods that help to stabilize the training by reducing the variance. It provides a solution to reducing the RL algorithm variance, training agents faster and better.

Community Detection is one of the fundamental problems in network analysis, belonging to the field of complex network studies. According to [13], the community detection technique is characterized by having a community structure, where the nodes in the network can be grouped into sets such that each set of nodes is densely connected. For [14], community detection is the process of identifying relevant communities in a network that evolves as in a dynamic network. Community detection is key to understanding the structure of complex networks. Community detection techniques are useful for social media algorithms to discover people with similar opinions, similar functions, similar purposes, and common interests vital to scientific inquiry and data analytics. There are classic methods of community detection using spectral clustering and

statistical inference. However, such methods are drop out, as deep learning techniques demonstrate an increased capacity to handle high-dimensional graph data with impressive performance.

Actor-Critic Architecture for Community Detection (AC2CD) is a DRL architecture with an approach based on GAT, which are novel neural network architectures that operate on graph-structured data, leveraging masked self-attention layers [15]. It is employed to find an optimal community structure in a dynamic social network while also serving as a learning component to select actions and improve the value function [5]. Experimental work indicates that AC2CD copes well with dynamic real-world social networks. Nevertheless, the performance of such complex architecture motivates AS approach to enhance performance to evaluate the growing size of dynamic social networks.

2.2 Ablation Study

The first idea of AS comes from speech recognition studies of [16]. Although not a new idea, it is a relatively young AI research theme [4]. AS is defined by [17] as a scientific method that involves highlighting or removing individual or blocks of components from a system to prove and understand which aspects of a system are vital through statistical analysis. Using statistics and analyzing the results obtained from AS, it is possible to gain insights into the relative importance of the parameters of architecture or model. With these insights, it is possible to get improvements in the design, optimization, and interpretability of the system. AS is a valuable tool for discovering the component's influence in ML systems. Through statistical analysis, it is possible to enhance the interpretability of ML approaches.

The use of AS in XAI systems becomes interesting, considering its complementarity to understanding AI systems. The AS aims to understand the importance of parameters and code blocks in an architecture or model [4]. This study enables identifying and quantifying the influence of various components on an algorithm, model, or architecture, leading to a better understanding of the underlying mechanisms. This understanding is crucial for building trust and ensuring transparency in AI systems.

3 Related Work

Related work focusing on AS and DRL using the actor-critic architecture is presented in this section, with publications from 2018 to 2023. Table 1 presents the related work outline. Note this work is the only one that includes GAT as a novel convolution-style neural network architecture that operates on graph-structured data. GAT is one of the most popular types of graph neural networks applied to the community detection problem. But although GAT presents a significant direction for ML research, it has received comparatively low levels of

Table 1. Related work outline.

Reference	AS	DRL	Actor-Critic	GAT
Fan et al. (2023) [18]	✓	✓	✓	
Naqvi & Anggorojati (2022) [19]	✓	✓		
Ye et al. (2022) [20]	✓	✓	✓	
da Silva Filho et al. (2022) [21]	✓	✓		
Hessel et al. (2018) [22]	✓	✓		
The AC2CD AS	✓	✓	✓	✓

attention, motivating this AS to assess its effectiveness through the AC2CD case study.

The authors in [18] present a new approach using DRL and actor-critic for a multi-agent system that analyzes and simulates an environment with multiple intelligent agents across various domains. Additionally, an AS is conducted to assess the effectiveness of the innovative components in the proposed method. The results show that each component of the actor-critic algorithm is indispensable for good interception performance, including success rate, good reward, and interception steps.

In [19], the authors explore the utilization of DRL for congestion control in cellular network settings. Congestion control uses algorithms responsible for regulating the data transmission rate in a network to prevent congestion. The author employs an AS to identify the component that influences the algorithm that uses the policy gradient method. Using the AS, the authors remove or modify parameters to analyze the impact of the changes on the algorithm's performance. In conclusion, a higher reward for the method presented is not always related to better networking performance.

In [20], the authors focus on the popularity of multi-agent DRL demand for large-scale real-world tasks, which are hampered by the low sample efficiency of the models and the high cost to collect data. Thus, AS is used to investigate, validate and understand the contribution of each component in multi-agent actor-critic methods. The authors propose PEDMA, a plugin unit for multi-agent DRL that consists of three techniques: (i) parallel environments to accelerate the data acquisition; (ii) experience augmentation that utilizes the permutation invariance property of the multi-agent system to reduce the cost of acquiring data; and (iii) delayed updated policies to improve the data utilization efficiency of the multi-agent DRL model. Experiments on three multi-agent benchmark tasks show that the multi-agent actor-critic model trained with PEDMA outperforms the baselines and state-of-the-art algorithms.

The authors in [21] discuss the learning-to-optimize method for automatically optimizing algorithms from data instead of using traditional HPs tuning. The focus is on learning global optimization by DRL. The authors advocate that learning to optimize is not a well-explored theme. It provides a direct framework to learn an optimizer able to deal with the exploration-exploitation dilemma

and that the applied techniques improved stability and generalization. Thus, the authors conducted AS to investigate the significance of learning strategies and components concerning this optimization.

In [22], the authors perform an AS to understand the contribution of the components and parameters in the Deep Q-Network (DQN) algorithm that utilizes DRL to address the challenge of learning in complex and high-dimensional environments. In each ablation phase, an algorithm component is removed or changed. Subsequently, the algorithm's performance is analyzed. The authors propose Rainbow to combine improvements in DRL. In experiments, the authors examine six extensions to the DQN algorithm and empirically study their combination. The results show that the combination provides state-of-the-art performance on the Atari 2600 benchmark considering data efficiency and final performance.

4 Experimental Method

Figure 1 presents the experimental method with three steps. The first step includes the dataset and baseline definitions. The dataset used refers to an email network between members of a European research institution (Email-Eu) which is available on the Snap Project website[1]. This dataset is called Email-Eu-Core and is used as input to AC2CD. The second step includes the HPs variations during the AC2CD executions (*tunning*). Finally, in the third step, we use the AS to observe the importance of the selected HPs. The effect analysis focuses on the execution time, GPU, and memory usage.

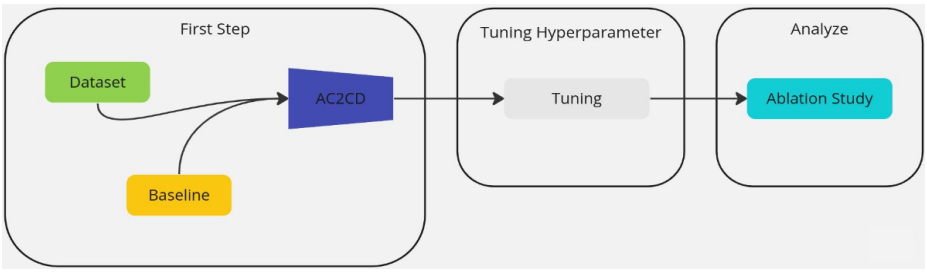

Fig. 1. Experimental method diagram.

The Email-Eu-Core dataset is used in the experiment, which is a directed network representing an email network between members of a European research institution. According to [23], the network is formed by an edge (u, v), where u represents the person who sends at least one email to v. The communities in the dataset represent the departments in the organization. There are $1,005$ nodes,

[1] https://snap.stanford.edu/data/email-Eu-core.html

25,571 edges, and 42 communities, with the longest path being seven, and the average clustering coefficient is 0.3994.

The experiments were performed on a computer composed of a CPU Intel® Xeon Gold 5220R with 48 cores, 187GB of RAM, and two GPU NVIDIA® V100S. The operating system used was Ubuntu and all external libraries were provided by the Conda project.[2]

4.1 Experimental Definitions

The AS setup is based on I, the set of previously defined percentage variations (10%, 30%, 50%, 70%). These percentage variation values were based on their coverage and the comprehensiveness they provided with the conducted tests. We defined three HPs of H for the experiments ($learn\_rate, batch\_size, n\_games$). The *learn_rate* (LR) determines the extent to which an agent learns from each sample in the environment [24]. The *batch_size* (BS) represents the number of samples propagated during the training session [25]. Lastly, *n_games* (NG) defines the number of episodes the agent will process.

Definition 1 (Hyperparameters). *refer to parameters set before the model is trained, rather than being estimated from learning as they define the architecture of the model [26]. They are used to configure an ML model and specify the algorithm to minimize the loss function, for example.*

The HPs and their respective values were chosen according to the suggestion of the author of the AC2CD architecture, considering the previously analyzed influence of each parameter on the developed algorithm. Therefore, the baseline values for the execution of the experimental method regarding each HP of H are 40, 40, and 100 for the LR, BS, and NG, respectively.

Definition 2 (Baseline). *is the reference for a particular ML study [4]. In our case, it refers to the set of executions with the default values of the HPs used to compare the variations of the selected HPs.*

After the definition of HPs values for the baseline, the AC2CD architecture integration with memory, and GPU profiler known as Scalene[3] is done, the execution of the AS started. This experiment's objective is to define a set of HPs values for each HP in H that makes the AC2CD consume less GPU and memory resources while accomplishing this in the fastest way possible. Scalene is used since it outperforms other well-known profilers, such as cProfile, and memory_profile[4]. Additionally, this tool performs profiling at the line level and per function, providing functions information and specific lines of code responsible for the program's execution time, as can be verified in [27].

The execution of baseline values was performed using the percentages from the set I for each hyperparameter (HP) in the set H. Subsequently, each HP was

[2] Conda Project available at `https://docs.conda.io/en/latest/`
[3] https://github.com/plasma-umass/scalene
[4] https://pypi.org/project/memory-profiler/

paired with another HP, with the first parameter fixed and the second parameter tuned. For example, a fixed LR of 10% and a BS varying from 10% to 70% were used, followed by a fixed learning rate of 30% and a tuning BS ranging from 10% to 70%, and so on. Similarly, this process was repeated for the remaining HPs, ensuring that all HPs were tuned in pairs with each other. This process tries to capture the interaction among HPs.

Definition 3 (Manual Tuning of HPs). *is a technique for adjusting the value of an HP. We employed this method to vary four percentages outlined in the I set. We used this approach in conjunction with the conceptualization of AS in section 2.2 to determine the optimal configuration of GPU, memory, and time consumption of the three HPs specified in the set H.*

5 Results and Discussion

In this section, the GPU usage, memory usage, and runtime execution results are presented for the AC2CD architecture after executing it with variations in the percentage of the I set with the Email-Eu-Core dataset. The results of the baseline execution were: 12.092 GiB for the memory GPU usage, 120.615 GiB for main memory consumption, and $1h40m24s$ for runtime execution.

GPU Consumption

Figure 3 presents the GPU usage, considering the relation LR and BS. We observe that GPU usage is higher when only LR value is varied until 50% of variation, along with the BS. Additionally, within this same variation, Figure 2 presents the Scalene interface where we can observe that the code segments related to agent learning and community applications for each node in the dataset are the ones that consume the most GPU in this configuration.

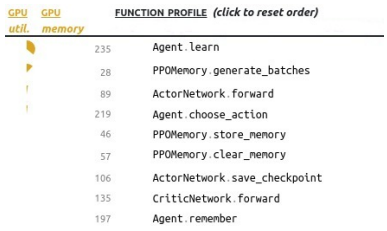

Fig. 2. GPU rank usage in scalene tool for LR fixed in 10%

Regarding the variation of LR along with NG, as we can see in Figure 4, the fixed LR value at 30% and the NG varying in the percentages of I predominantly

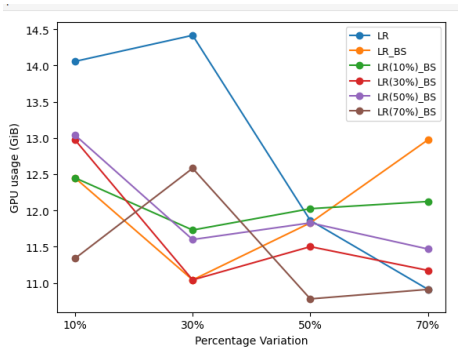

Fig. 3. GPU usage with LR and BS variation

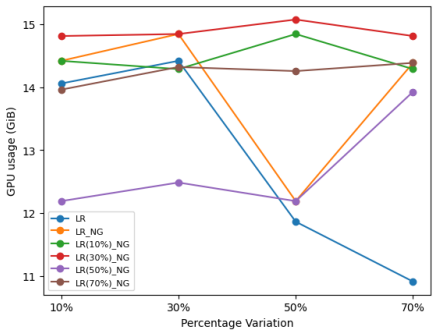

Fig. 4. GPU usage with LR and NG variation

consume more GPU compared to other variations. More once, the agent learning is the code block that consumes the most GPU in all variations.

With the BS fixed along with LR variations as shown in Figure 5, it can be observed that the fixed BS at 10% and varying LR consumes more GPU in most of the variations compared to other fixed BS values with LR variations. Additionally, the GPU consumption remains relatively constant when only BS is varied.

Regarding the variation of BS and NG shown in Figure 6, the GPU consumption was found to be similar for the cases where both BS and NG are varied together and when the BS is fixed at 10% with NG varying. Also here, the agent learning stage is the code block of AC2CD that consumes the most GPU in the context of these variations.

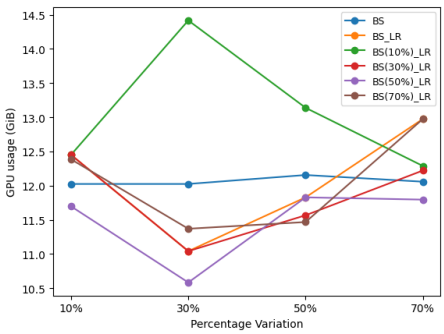

Fig. 5. GPU usage with BS and LR variation

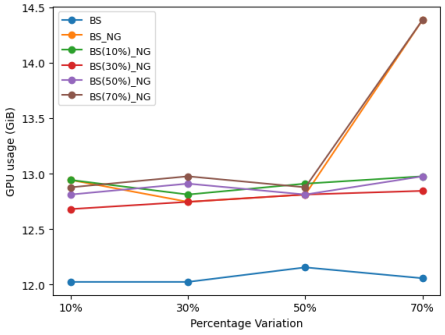

Fig. 6. GPU usage with BS and NG variation

As shown in Figure 7, the NG fixed at 70% with varying LR exhibited more consistent results than the other variations. In Figure 8, the GPU consumption for the NG varying along with BS was similar to the case where NG was fixed at 70% with varying BS.

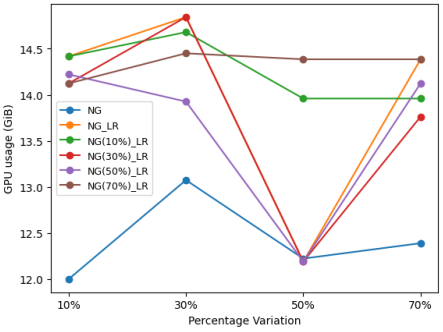

Fig. 7. GPU usage with NG and LR variation

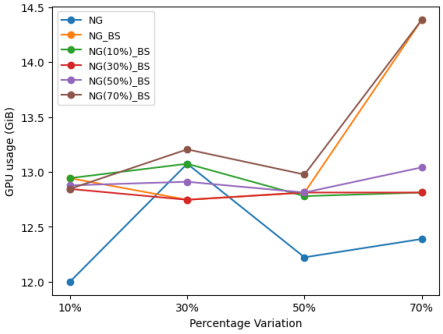

Fig. 8. GPU usage with NG and BS variation

Memory Consumption

As observed in Figures 9,10,11,12,13,14, the memory consumption does not exhibit as much variation with LR, BS and NG variations compared to the varied GPU consumption. It is important to note that when LR, BS, and NG are varied individually, the memory consumption remains constant, as seen in Figures 9,11,13, respectively. We can attribute this behavior to the fact that most memory allocation in the AC2CD is used by tensors, preferentially stored in the GPU.

It can also be observed in figure 9 and 11 that when the BS reaches a variation of 70% with fixed values, the architecture tends to consume more memory. This aspect can be attributed to the characteristic of the BS, where a wider BS leads to faster learning but at the expense of higher memory consumption.

In figures 10 and 13, it is observable that when the NG is fixed at certain percentage values and the LR is varied, there is a convergence in memory consumption. On the other hand, when the LR is fixed at a certain percentage value of I, and the NG is varied, the results show outliers but, in general, the consumption remains relatively constant and lower than the previously mentioned scenario. This can be attributed to the direct influence of the LR on memory usage, similar to the BS.

Runtime Execution

We can observe in Figure 15 that the execution time decreases according to the percentage variations. On the other hand, in Figure 16, the execution time

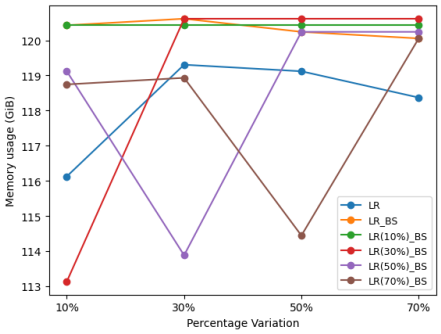

Fig. 9. Memory usage with LR and BS variation

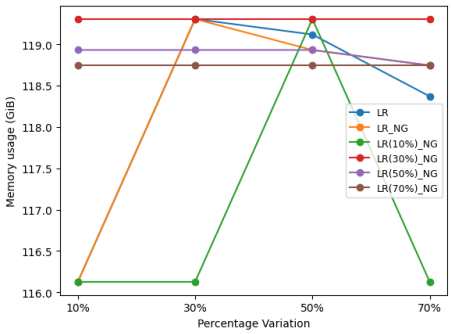

Fig. 10. Memory usage with LR and NG variation

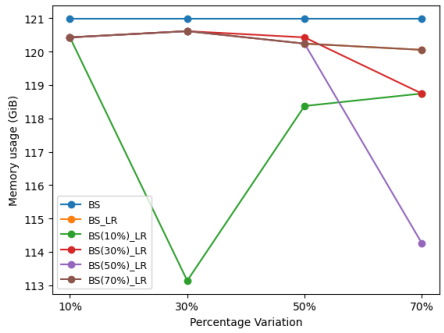

Fig. 11. Memory usage with BS and LR variation

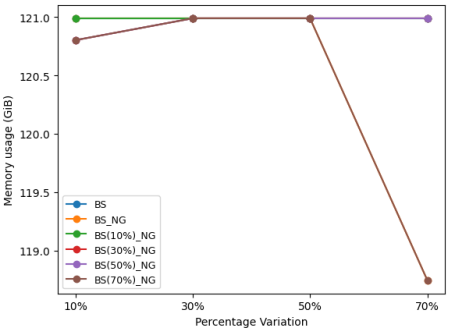

Fig. 12. Memory usage with BS and NG variation

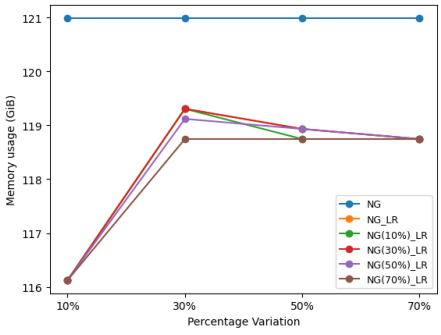

Fig. 13. Memory usage with NG and LR variation

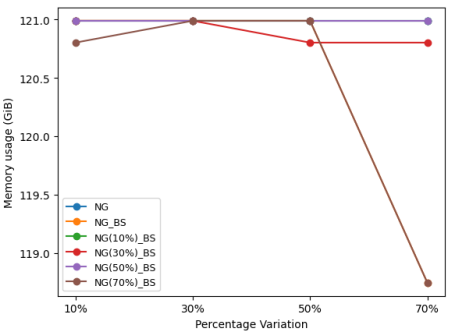

Fig. 14. Memory usage with NG and BS variation

increases, and this happens because the NG parameter determines the number of episodes that the agent will process during the execution of the architecture. In

Figure 16, it is possible to see that the execution time does not increase because when the LR is too big the execution is faster, this fact is a consequence that we do not need too many iterations to converge to good values.

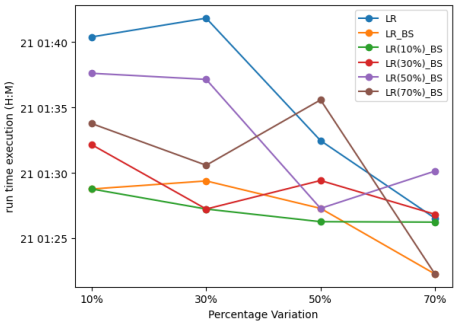

Fig. 15. Runtime execution with LR and BS variation

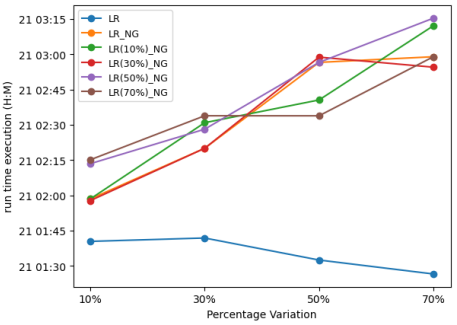

Fig. 16. Runtime execution with LR and NG variation

The execution time remained relatively constant when both BS and LR varied together, as well as when BS varied alone, and the other HPs are kept at the baseline value, as observed in Figure 17. Conversely, the results in Figure 18 demonstrate that when BS was fixed at a certain percentage while NG varied, the execution time was consistently high. These findings suggest that the choice of BS and NG can significantly impact the execution time of the algorithm. Considering the results that consistently showed no decrease in execution time when NG varied together with BS.

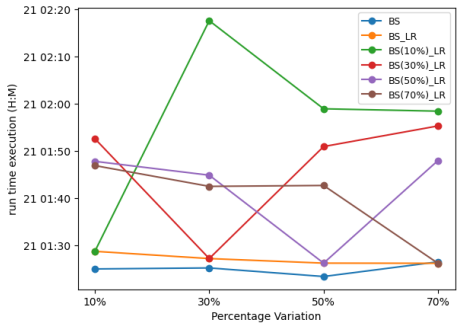

Fig. 17. Runtime execution with BS and LR variation

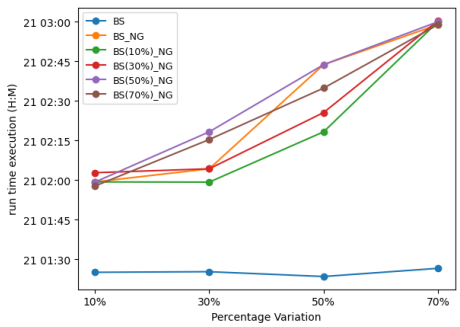

Fig. 18. Runtime execution with BS and NG variation

In Figures 19 and 20, when NG was fixed at 10 and LR and BS were varied, there was a consistent pattern of execution time. This could possibly be

attributed to the NG parameter having a stronger influence on the execution time compared to LR and BS. When NG is fixed and the other parameters are varied, the variation in execution time is not as pronounced, suggesting that the impact on execution time is primarily driven by the variability of NG. These findings suggest that optimizing the NG parameter may have a more significant effect on controlling execution time compared to LR and BS.

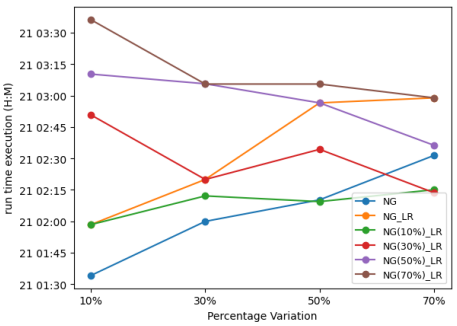

Fig. 19. Runtime execution with NG and LR variation

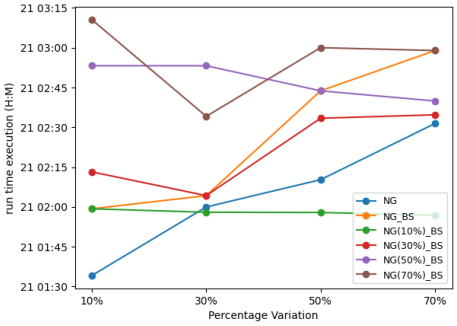

Fig. 20. Runtime execution with NG and BS variation

6 Conclusion

The AS application in AC2CD reveals that it is important to carefully consider the values of LR, BS, and NG in order to optimize GPU and memory consumption as well as runtime execution. Based on the results obtained from our experiments and analyses, it is evident that LR and BS have shown a strong influence on GPU and memory usage, while NG has the most significant impact on execution time. Therefore, the definition of a fixed value of LR, BS, and NG as 70%, 50%, and 10%, respectively of the baseline value, proved to be a good option for the HPs configuration of the architecture in terms of GPU usage, considering the consumption ranged between 12 GiB and 13 GiB. Regarding memory usage, a fixed value of LR, BS, and NG as 10% of the baseline value, for each HP, emerged as a favorable configuration in terms of memory usage. Lastly, in terms of execution time, the combination of a fixed LR and NG at 10% with a fixed BS at 50% proved to be a potential configuration for achieving more efficient execution.

This work contribution is to present an AS with the AC2CD architecture, shedding light on the influential factors that contribute to the algorithm's performance. The insights gained from this study contribute to the ongoing efforts to enhance the efficiency and effectiveness of DRL algorithms in real-world applications in the XAI direction.

In future work, we consider the implementation of AS for the remaining HPs of the AC2CD architecture with other datasets and explore the potential for automatic HP tuning. Additionally, investigating the impact of an AS on the agent's learning stage within the AC2CD architecture presents a promising research area for advancing the study of DRL models with a focus on XAI transformations. This path of research holds the potential for significant advancements in understanding and interpreting the decision-making processes of DRL models.

References

1. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.
2. Allen Newel. A tutorial on speech understanding systems. In Raj Reddy, editor, *Speech Recognition Invited Papers Presented at the 1974 IEEE Symposium*, pages 3–54. Carnegie Mellon University, USA, 1974.
3. Sina Sheikholeslami. Ablation programming for machine learning. Master's thesis, KTH Royal Institute of Technology, School of Electrical Engineering and Computer Science (EECS), SE-100 44 Stockholm, Sweden, 2019.
4. Isha Hameed, Samuel Sharpe, Daniel Barcklow, Justin Au-Yeung, Sahil Verma, Jocelyn Huang, Brian Barr, and C. Bayan Bruss. Based-xai: Breaking ablation studies down for explainable artificial intelligence, 2022.
5. Aurélio Ribeiro Costa and Célia Ghedini Ralha. AC2CD: An actor–critic architecture for community detection in dynamic social networks. *Knowledge-Based Systems*, 261:110202, 2023.
6. David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.
7. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
8. Issam El Naqa and Martin J. Murphy. *What Is Machine Learning?*, pages 3–11. Springer International Publishing, Cham, 2015.
9. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, may 1996.
10. Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.
11. Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.
12. Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.
13. Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
14. Rémy Cazabet, Giulio Rossetti, and Fredéric Amblard. Dynamic community detection, 2017.

15. Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.
16. Dabbala Rajagopal Reddy. *Speech recognition: invited papers presented at the 1974 IEEE symposium*. Elsevier, 1975.
17. Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks, 2019.
18. Dongyu Fan, Haikuo Shen, and Lijing Dong. Switching-aware multi-agent deep reinforcement learning for target interception. *Applied Intelligence*, 53(7):7876–7891, 2023.
19. Haidlir Naqvi and Bayu Anggorojati. Ablation study of deep reinforcement learning congestion control in cellular network settings. In *Proc. of 25th Int. Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 80–85. IEEE, 2022.
20. Zhenhui Ye, Yining Chen, Xiaohong Jiang, Guanghua Song, Bowei Yang, and Sheng Fan. Improving sample efficiency in multi-agent actor-critic methods. *Applied Intelligence*, pages 1–14, 2022.
21. Moésio Wenceslau da Silva Filho, Gabriel A. Barbosa, and Péricles B. C. Miranda. Learning global optimization by deep reinforcement learning. In *Proc. of 11th Brazilian Conference on Intelligent Systems (BRACIS)*, page 417–433, 2022.
22. Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. of AAAI Conference on Artificial Intelligence*, volume 32, 2018.
23. Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proc. of 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, page 817–826, 2009.
24. Joshua Romoff. *Decomposing the Bellman Equation in Reinforcement Learning*. PhD thesis, 2021. Copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Última atualização em - 2023-03-05.
25. Brennan Shacklett, Erik Wijmans, Aleksei Petrenko, Manolis Savva, Dhruv Batra, Vladlen Koltun, and Kayvon Fatahalian. Large batch simulation for deep reinforcement learning. *arXiv preprint arXiv:2103.07013*, 2021.
26. Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.
27. Emery D. Berger, Sam Stern, and Juan Altmayer Pizzorno. Triangulating python performance issues with scalene, 2022.