Tuning Proximal Policy Optimization Algorithm   
in maze Solving with ML-Agents

Phan Thanh Hung, Mac Duy Dan Truong, Phan Duy Hung

Computer Science Department, FPT University, Hanoi, Vietnam

hungpthe140966@fpt.edu.vn, truongmddhe141711@fpt.edu.vn,

hungpd2@fe.edu.vn

**Abstract:** The proximal Policy Optimization algorithm is the default reinforcement algorithm used in the ML-Agents toolkit. This algorithm can alternate between sampling data through interaction with the environment and optimizing a "surrogate" cost function using stochastic gradient descent. Although when creating a new machine learning model, it is tough to know the optimal model architecture for a given project immediately. In most cases, we can use the default given values from the algorithm's creator or use the machine to perform this exploration and automatically select the optimal model architecture. Hyperparameters define the model architecture; thus, searching for the best model is called hyperparameter tuning. We focus on comparing four hyperparameters: Beta, Epsilon, Lambd, Num\_epoch of PPO algorithm in solving a maze. The results obtained in the training process show the difference in the selection of hyperparameters. The modification of hyperparameters will depend on the maze's complexity and the complexity of the Agent's actions. This paper will help to make appropriate choices at hyperparameters in concrete and practical projects.

**Keywords:** Reinforcement Learning, Proximal Policy Optimization, ML-Agents, maze-solving.

1. Introduction

With the development in computing technology and the inception of new intelligent algorithms, the goal of Artificial Intelligence (AI) has become a step closer to the purpose of mimicking the human brain. In that field, a branch that is becoming more and more important is reinforcement learning (RL) [1]. RL is a promising machine learning method rising in popularity in the video game industry in recent years with many breakthroughs, especially in games, making robots do tasks instead of humans or working in an unknown environment [2].

RL is applied very strongly to games, so the Machine Learning Agents Toolkit (ML-Agent) library [3] was born as a bridge for using RL into Unity[3]. Unity ML-Agent is also an "open-source Unity plugin that enables games and simulations to serve as environments for training intelligent Agents." It also provides implementations (based on PyTorch) of state-of-the-art algorithms to enable game developers and hobbyists to efficiently train intelligent Agents for 2D, 3D, and VR/AR games [4]. RL has been used to train artificial intelligence for play games; such as Dota2 with OpenAI Five in 2017, Chess and Go in 2018, and StarCraft in 2019 [5];

However, advances in RL algorithms and development platforms such as Unity's ML-Agents make it possible to model real-world environments and Agents. Thus creating an ecosystem where AI can be trained using RL algorithms such as Proximal Policy Optimization (PPO) [6], which OpenAI developed, has made a new turning point in reinforcement learning.

Today, more projects use automated software as a substitution for humans. One of them is a puzzling maze that consists of a different branch of passages where the solver aims to reach the destination by finding the most efficient route within the shortest possible time [7]. Artificial Intelligence plays a vital role in defining the best possible way of solving any maze effectively. That is when RL came in handy.

A vital issue in the usability of an RL method is sensitivity to hyperparameters [8]. Learning complex tasks can take hours or days, fine-tuning hyperparameters is tedious. Thus, this research focuses on changing the hyperparameters in configuration for a well-trained PPO algorithm in maze solving with ML-Agent library and Unity. This paper also provides a helpful guideline for tuning hyperparameters when redeployment algorithms on novel environments in the future.

1. Related Works

Several approaches have been proposed for RL with neural network function approximators in recent years. The paper published by John Schulman [6] has proposed improving the current state of affairs by introducing an algorithm that attains data efficiency and reliable performance, called Proximal Policy Optimization (PPO). To optimize policies, the research team alternate between sampling data from the policy and performing several epochs of optimization on the sampled data. They concluded that these methods have the stability and reliability of trust-region strategies but are much simpler to implement. They require only a few lines of code change to a vanilla policy gradient implementation, are applicable in more general settings, and have better overall performance [9]. However, the algorithm was successful on various problems without tuning hyperparameter values, meaning that the results still did not achieve the best possible outcome.

Though previous studies have shown that SAC[10] generally performs better than PPO, hyperparameter tuning can significantly impact the performance of these algorithms. This research by Usguta Vohra[11] has aimed to evaluate the effect of the number of layers and number of nodes in SAC and PPO algorithms in a search-and-retrieve task developed in Unity 3D game engine. The paper compared the SAC and PPO models on four different test conditions that differed in the ratios of targets and distractors. Results revealed that PPO performed better than SAC for all test conditions when the number of layers and units present in the architecture was the lowest. It also implied that similar hyperparameter settings might be used while comparing models developed using DRL algorithms. Increasing the number of nodes while keeping the hidden layers constant proved beneficial for PPO in all the model configurations. They also give new insight into measuring the model’s performance in various testing scenarios to serve as a benchmark for deep reinforcement learning Agents testing.

Other autonomous game-playing Agents have been developed using different rule-based systems to machine learning methods. Many efforts have also been put into creating Agents that behave in a way that resembles as closely as possible the way a human player would play. To estimate the player completion rate of several levels in Lily's Garden by Tactile Games, the research team developed a set of PPO-based reinforcement learning Agents [2]. It evaluated how the number of steps taken by the Agent for completing the levels relates to the behavior of a sample of ∼900,000 players. The results based on ∼60% of the game mechanics – demonstrate that the two-step training scenario leads to the most proficient Agent. In contrast, the Agent attains the most significant correlation to real players' completion rates with the one-step curriculum. The work is only for a limited subset of levels with default values hyperparameter, so the results may not be the best possible outcome [10,11].

All the above studies showed that PPO Is an efficient technique in various problems, but the main focus is testing video games or mimicking how humans play. Although most research still uses the default or just a little tuning, proper hyperparameter initialization and search can improve results.

1. Design Maze and Agent in Unity

This paper aims to build the environment using the game engine Unity. By taking Unity as a simulation platform, the toolkit enables the development of learning environments rich in sensory and physical complexity, provides compelling cognitive challenges, and supports dynamic multi-agent interaction. Unity also helps in interface design quickly; instead of UI code, objects can be dragged, dropped, and arranged scientifically and quickly[13].

The ML-Agents toolkit is built on the OpenAI Gym [14] environment as a wrapper and communicates between Python API and Unity C# Engine to build deep learning models. Although there have been fundamental changes in the way the toolkit works in the latest release, 2.0.0, the core functionality of the ML toolkit remains the same. The latest release of version 2.0.1 at the time of writing this research has several new features such as changing interface, removing some methods previously marked as obsolete, and replacing them with their supported counterpart.

The toolkit allows developers to easily integrate machine learning Agents in their games and provides AI researchers with an easily customizable platform to experiment. The framework defines three brain types: player brain, heuristic brain (i.e., scripted behavior), and learning brain [15]. These brains control Agents in the environment. The ML-Agents toolkit chose to implement a baseline reinforcement learning algorithm based on Proximal-Policy Optimization.

Maze design for training Agent:

* The maze map has 8x8 cells. A cell includes four walls and one floor.
* An Agent with four discrete actions: go up, down, left, and right. Each action is to move into a cell.
* There is a destination for the Agent to complete the maze.
* The Agent has four raycasts (to detect collisions with the maze walls) on four sides around the Agent. The length of the raycast is one cell.
* The Agent has four 3D Ray Perception Sensors [16] - the Agent's observations, arranged according to the other four raycasts.
* The total size of the created observations is: (Observation Stacks) \* (1 + 2 \* Rays Per Direction) \* (Num Detectable Tags + 2) = 1 \* (1 + 2 \* 2) \* (1 + 2) = 15.



**Fig. 1.** Simple 8x8 maze.

The Agent moves in 4 directions (up and down, left, right). When the Agent moves in a specific direction and that side's raycast detects the wall, but the Agent still decides to go in that direction, deduct 1 point.

Agent when entering a cell will award or punished:

* Entering for the first time, Agent gets 3 points, and that background box turns yellow.
* Entering the second time, the Agent deducts 0.5, and the background cell turns orange.
* The Agent deducts 1 point the third time, and the background box turns purple.
* Entering from the fourth time onwards, the Agent has deducted 2 points, and the background is still purple. Purple is the final penalty level when entering.
* When colliding with the end of the maze, Agent will be awarded 100 points and finish solving the maze.

1. Experiments and Results
   1. Implementation

Mazes and Agents use The software Unity Ver.2020 and ML-Agent library to build. All experiments were performed on NVIDIA 1080Ti GPU. The PPO algorithm with the hyperparameters is varied differently, with the max-step during training being 5000000.

There are two conditions to ending an episode:

* The Agent moves in the maze with enough steps of 3000, the episode will end, and the maze will reset.
* The Agent moves to the destination and ends the episode, and the maze will reset.
  1. Config Hyperparameters

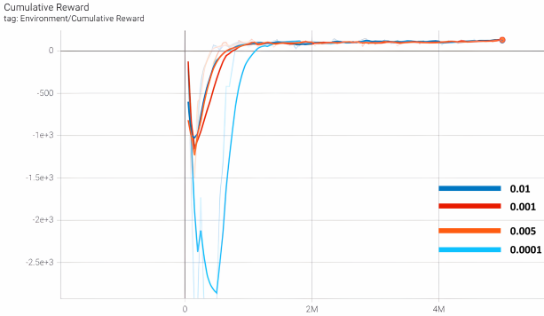
To perform tuning, the hyperparameters are taken to default values and then change each of its values to observe and evaluate the results.

|  |
| --- |
| hyperparameters:  batch\_size: 128  buffer\_size: 2048  learning\_rate: 0.0003  beta: 0.005  epsilon: 0.2  lambd: 0.95  num\_epoch: 3  learning\_rate\_schedule: linear |

**Fig. 5.** Default configuration hyperparameters.

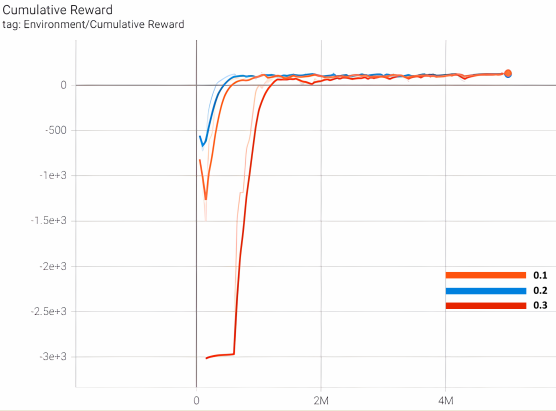
* 1. Results of maze training

Result of Fixed maze 8x8



**Table 1.** Compare the results when changing the hyperparameter Beta.

|  |  |  |
| --- | --- | --- |
| Beta | Reward | Time Cost |
| 0.01 | 122.8 | 49m 45s |
| 0.001 | 134.1 | 43m 10s |
| 0.005 | 143.1 | 43m 53s |
| 0.0001 | 143.2 | 43m 30s |



|  |  |  |
| --- | --- | --- |
| Epsilon | Reward | Time Cost |
| 0.1 | 111 | 43m 12s |
| 0.2 | 143.1 | 43m 53s |
| 0.3 | 132.3 | 43m 37s |

**Table 2.** Compare the results when changing the hyperparameter Epsilon.

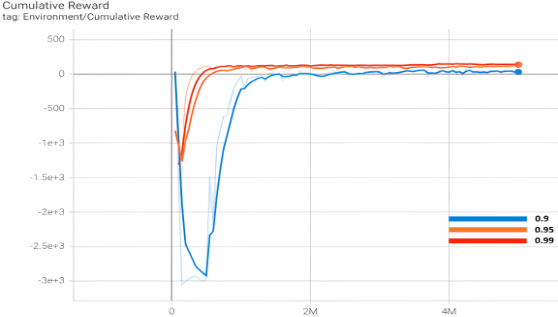
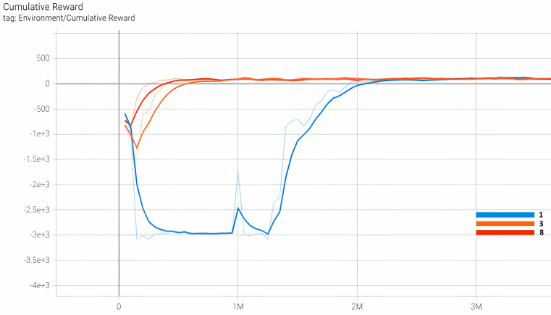


Table 3. Compare the results when changing the hyperparameter Lambd.

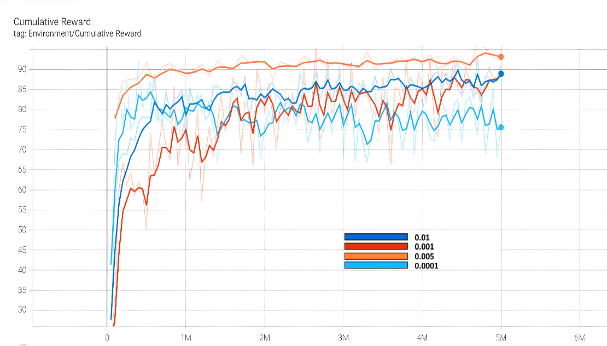
|  |  |  |
| --- | --- | --- |
| Lambd | Reward | Time Cost |
| 0.9 | 34.16 | 49m 5s |
| 0.95 | 130.8 | 43m 53s |
| 0.99 | 140.2 | 43m 4s |



|  |  |  |
| --- | --- | --- |
| Num\_epoch | Reward | Time Cost |
| 1 | 117.6 | 25m 52s |
| 3 | 134.5 | 43m 53s |
| 8 | 125.2 | 1h 17m 0s |

**Table 4.** Compare the results when changing the hyperparameter Num\_epoch.

Results of Random maze 4x4

****

**Table 5.** Compare the results when changing the hyperparameter Beta.

|  |  |  |
| --- | --- | --- |
| Beta | Reward | Time Cost |
| 0.01 | 91.27 | 1h 3m 20s |
| 0.001 | 90.83 | 1h 29m 20s |
| 0.005 | 92.49 | 1h 20m 34s |
| 0.0001 | 76.26 | 2h 6m 18s |

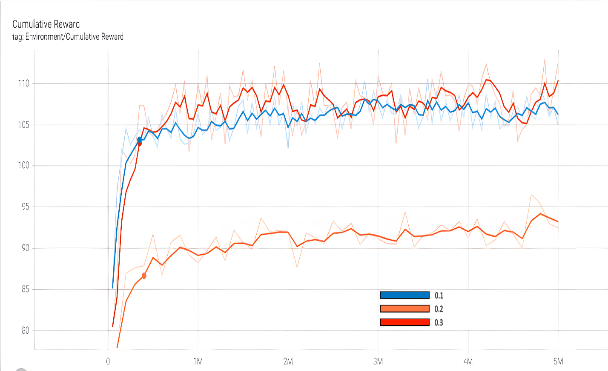
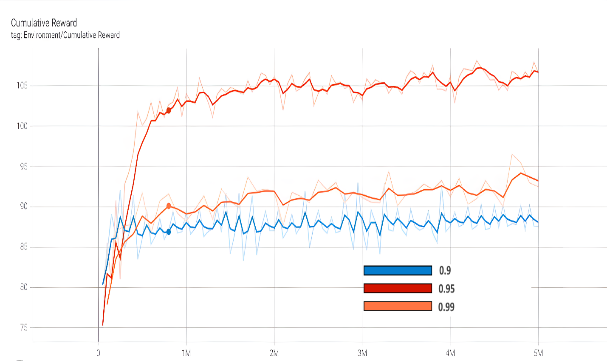


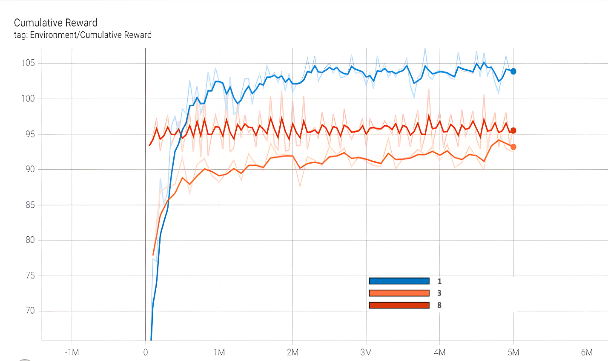
Table 6. Compare the results when changing the hyperparameter Epsilon.

|  |  |  |
| --- | --- | --- |
| Epsilon | Reward | Time Cost |
| 0.1 | 104.9 | 1h 14m 28s |
| 0.2 | 92.49 | 1h 20m 34s |
| 0.3 | 112.7 | 2h 6m 18s |



**Table 7.** Compare the results when changing the hyperparameter Lambd.

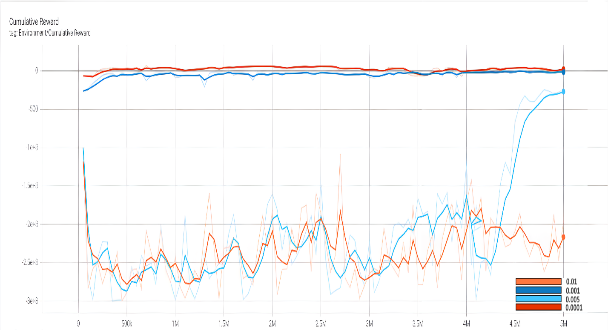
|  |  |  |
| --- | --- | --- |
| Lambd | Reward | Time Cost |
| 0.9 | 87.54 | 1h 47m 8s |
| 0.95 | 92.49 | 1h 20m 34s |
| 0.99 | 106.6 | 1h 15m 16s |



|  |  |  |
| --- | --- | --- |
| Num\_epoch | Reward | Time Cost |
| 1 | 103.6 | 57m 32s |
| 3 | 92.49 | 1h 20m 34s |
| 8 | 95.91 | 2h 15m 26s |

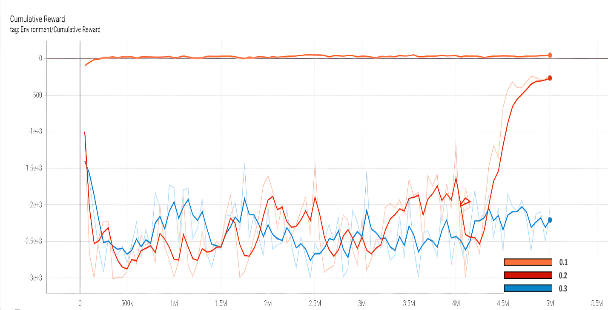
**Table 8.** Compare the results when changing the hyperparameter Num\_epoch.

Results of Random maze 6x6



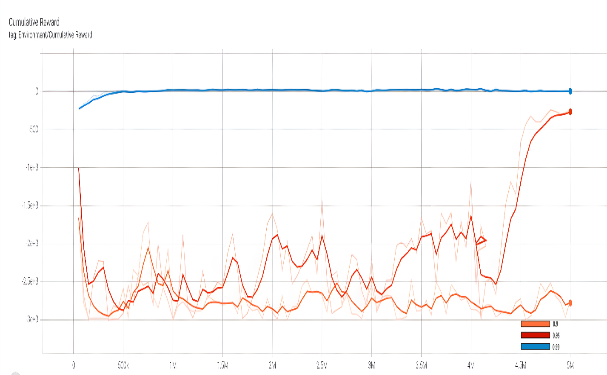
**Table 9.** Compare the results when changing the hyperparameter Beta.

|  |  |  |
| --- | --- | --- |
| Beta | Reward | Time Cost |
| 0.01 | -1949 | 1h 4m 13s |
| 0.001 | -17.02 | 1h 5m 26s |
| 0.005 | -227.4 | 1h 9m 22s |
| 0.0001 | 43.44 | 1h 12m 37s |



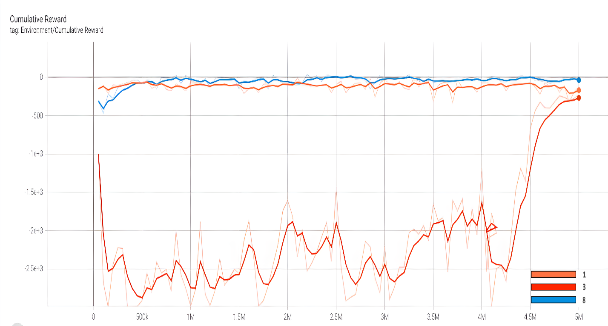
**Table 10.** Compare the results when changing the hyperparameter Epsilon.

|  |  |  |
| --- | --- | --- |
| Epsilon | Reward | Time Cost |
| 0.1 | 42.2 | 1h 6m 50s |
| 0.2 | -268.7 | 1h 9m 22s |
| 0.3 | -2056 | 1h 26m 5s |



**Table 11.** Compare the results when changing the hyperparameter Lambd.

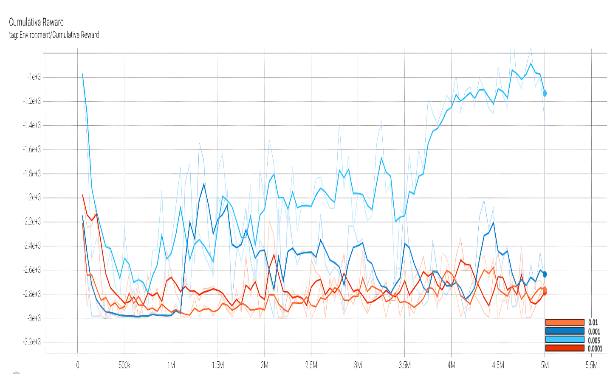
|  |  |  |
| --- | --- | --- |
| Lambd | Reward | Time Cost |
| 0.9 | -2740 | 1h 1m 45s |
| 0.95 | -227.4 | 1h 9m 22s |
| 0.99 | 4.037 | 1h 36m 0s |



|  |  |  |
| --- | --- | --- |
| Num\_epoch | Reward | Time Cost |
| 1 | -121.3 | 56m 52s |
| 3 | -227.4 | 1h 9m 22s |
| 8 | -64.08 | 1h 48m 21s |

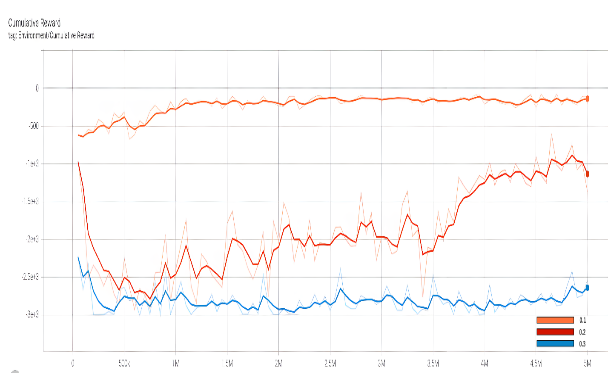
**Table 12.** Compare the results when changing the hyperparameter Num\_epoch.

Results of Random maze 8x8



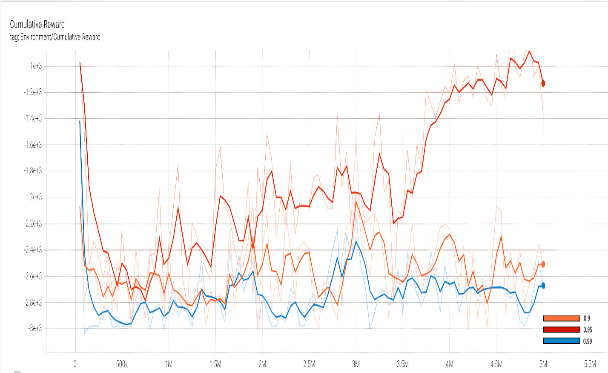
**Table 13.** Compare the results when changing the hyperparameter Beta.

|  |  |  |
| --- | --- | --- |
| Beta | Reward | Time Cost |
| 0.01 | -2785 | 1h 4m 47s |
| 0.001 | -2687 | 1h 35m 10s |
| 0.005 | -1370 | 1h 39m 53s |
| 0.0001 | -2690 | 1h 13m 11s |



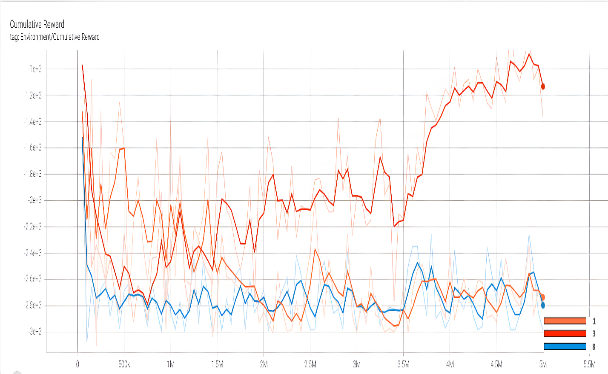
|  |  |  |
| --- | --- | --- |
| Epsilon | Reward | Time Cost |
| 0.1 | -114.1 | 1h 20m 35s |
| 0.2 | -1370 | 1h 39m 53s |
| 0.3 | -2540 | 1h 32m 50s |

**Table 14.** Compare the results when changing the hyperparameter Epsilon.



**Table 15.** Compare the results when changing the hyperparameter Lambd.

|  |  |  |
| --- | --- | --- |
| Lambd | Reward | Time Cost |
| 0.9 | -2510 | 2h 30m 6s |
| 0.95 | -1370 | 1h 39m 53s |
| 0.99 | -2661 | 1h 0m 33s |



|  |  |  |
| --- | --- | --- |
| Num\_epoch | Reward | Time Cost |
| 1 | -2818 | 41m 11s |
| 3 | -1370 | 1h 39m 53s |
| 8 | -2990 | 3h 10m 29s |

**Table 16.** Compare the results when changing the hyperparameter Num\_epoch.

* 1. Hyperparameters Configuration Analysis

**Beta:** Beta corresponds to the strength of the entropy regularization, which makes the policy "more random". This configuration ensures that Agents properly explore the action space during training. Table 1 shows that when the Beta decreases to 0.0001, the Agent explores the maze lesser and keeps moving at a certain distance. Increase the Beta; Agent will take more random action to explore the maze faster. Keep training for a long time; the smallest Beta gets the most reward out of 4 tests.

**Epsilon:** The reward of epsilon with a value of 0.3 in the first steps is greatly subtracted from the other two values ​​, and it takes longer to reach the destination. The value 0.2 has the best training result of the three tests from solving the maze and getting the most points.

**Lambd:** With a Lamda value of 0.9, the training is inferior. The reward is much less than the other two values ​​and the training time is also a bit more. Agent solves the maze about 1 million steps slower. Lambd values ​​from 0.95 - 0.99 give good results, and Agent learns faster.

**Num\_epoch:** Changing this value will make the model train fast or slow and significantly affect the model's performance quality. Num\_epoch has a small value (equal to 1) that makes the training unstable, even taking 2 million steps to solve the maze, much worse than the other two values. Increasing this value makes the Agent learn faster and update more consistently. However, the training time will extend; because of the number of passes made through the buffer before the gradient descent step is applied.

1. Conclusion and Perspectives

This paper gives the tuning for PPO algorithm through hyperparameters Beta, Epsilon, Lambd, and Num\_epoch. These values are changed and RL learning results are evaluated with the maze solving problem. The results show a clear difference between the training process and the hyperparameters. The change is based on different cases according to the complexity of the maze. Therefore, it is necessary to choose reasonable hyperparameters to set the best training results.

This paper also provides a helpful reference for tuning hyperparameters when redeployment PPO algorithm on novel environments in the future.

# References

1. Elgeldawi, E., Sayed, A., Galal, A.R., and Zaki, A.M.: Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis. Informatics, vol. 8, no. 4, pp. 1–21 (2021)
2. Kristensen, J.T., Burelli, P.: Strategies for Using Proximal Policy Optimization in Mobile Puzzle Games. arXiv:2007.01542 (2020)
3. Juliani, A. et al.: Unity: A General Platform for Intelligent Agents. arXiv:1809.02627 (2020)
4. Jonsson, A.: Deep Reinforcement Learning in Medicine. Kidney Dis (Basel). 2019 Feb;5(1):18-22. doi: 10.1159/000492670. Epub 2018 Oct 12. PMID: 30815460; PMCID: PMC6388442.
5. OpenAI et al.: Dota 2 with Large Scale Deep Reinforcement Learning, arXiv:1912.06680 (2019)
6. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O.: Proximal Policy Optimization Algorithms. arXiv:1707.06347 (2017)
7. Sadik, A.M.J., Dhali, M.A., Farid, H.M.A.B., Rashid, T.U., and Syeed, A.: A comprehensive and comparative study of maze-solving techniques by implementing graph theory. In: Proceedings of the Int. Conf. Artif. Intell. Comput. Intell. vol. 1, pp. 52–56 (2010)
8. Hamalainen, P., Babadi, A., Ma, X., and Lehtinen, J.: PPO-CMA: Proximal policy optimization with covariance matrix adaptation. In: Proceedings of the IEEE Int. Work. Mach. Learn. Signal Process. MLSP, doi: 10.1109/MLSP49062.2020.9231618 (2020)
9. Bellemare, M.G., Naddaf, Y., Veness, J., and Bowling, M.: The Arcade Learning Environment: An Evaluation Platform for General Agents. J. Artif. Intell. Res., vol. 47, pp. 253–279, doi: 10.1613/jair.3912 (2012)
10. Kristensen, J.T., Valdivia, A., and Burelli, P.: Estimating Player Completion Rate in Mobile Puzzle Games Using Reinforcement Learning. In: Proceedings of the IEEE Conf. Comput. Intell. Games, pp. 636–639, doi: 10.1109/CoG47356.2020.9231581 (2020)
11. Kim, T., and Lee, J.H.: Effects of Hyper-Parameters for Deep Reinforcement Learning in Robotic Motion Mimicry: A Preliminary Study. In: Proceedings of the 16th Int. Conf. Ubiquitous Robot, pp. 228–235, doi: 10.1109/URAI.2019.8768564 (2019)
12. “Unity - Manual: Creating user interfaces (UI).” https://docs.unity3d.com/Manual/UIToolkits.html (accessed Feb. 01, 2022).
13. Torrado, R.R., Bontrager, P., Togelius, J., Liu, J., and Perez-Liebana, D.: Deep Reinforcement Learning for General Video Game AI. In: Proceedings of the IEEE Conf. Comput. Intell. Games, doi: 10.1109/CIG.2018.8490422 (2018)
14. Johansen, M., Pichlmair, M., and Risi, S.: Video game description language environment for unity machine learning agents. In: Proceedings of the IEEE Conf. Comput. Intell. Games, doi: 10.1109/CIG.2019.8848072 (2019)
15. Jafri, R., Campos, R.L., Ali, S.A., and Arabnia, H.R.: Visual and Infrared Sensor Data-Based Obstacle Detection for the Visually Impaired Using the Google Project Tango Tablet Development Kit and the Unity Engine. IEEE Access, vol. 6, pp. 443–454, doi: 10.1109/ACCESS.2017.2766579 (2017)
16. Zhu, W., and Rosendo, A.: A Functional Clipping Approach for Policy Optimization Algorithms. IEEE Access, vol. 9, pp. 96056–96063, doi: 10.1109/ACCESS.2021.3094566 (2021)