RL Project Report

1st Xin Nian
Computer Science
University of the Witwatersrand,
Johannesburg, South Africa
1760829@students.wits.ac.za

1st Ryan Alexander
Computer Science
University of the Witwatersrand,
Johannesburg, South Africa
1827474@students.wits.ac.za

I. Abstract

In this project, we will attempt to solve a burdensome task in MiniHack [1], a sandbox framework for easily designing rich and diverse environments for Reinforcement Learning. The task "MiniHack-Quest-Hard-v0" contains 4 complex subtasks, Navigation; Pick Up; Inventory; Direction. We endeavour to solve the task using two reinforcement learning approaches, Proximal Policy Optimization(PPO) [2] and Option-Critic [3]. Although we did not solve the entire task, our agents showed decent progress in learning.[1]

II. Introduction

There are numerous RL methods and approaches to solve the task we assigned. However, in this project, we will only focus on Proximal Policy Optimization, a policy gradient member and a hierarchical method.

III. Proximal Policy Optimization

PPO was introduced in 2017 as an extension of the policy gradient method of RL. It Attempts to simplify the optimization process while retaining the advantages of TRPO, simultaneously improving the sample complexity. PPO performed comparably or better than other state-of-the-art approaches while being much simpler to implement and tune. Unlike the vanilla PPO that uses an adaptive KL penalty to control the policy change at each iteration. Adapting from OpenAI [2], our variant uses a novel objective function not typically found in other algorithms as shown in equation 1. θ is the policy parameter. $\hat{E}_t$ denotes the empirical expectation over timesteps. r_t is the ratio of the probability under the new and old policies, respectively. $\hat{A}_t$ is the estimated advantages at time t. ϵ is a clipping range hyper parameter.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \quad (1)$$

A. Action Space

MiniHack has a large, structured and context-sensitive action space. In our experiment, we initially considered reducing most of the actions as it is unrelated to the task. We kept eight direction actions N, E, S, W, NE, NW, SE, SW, and six Command actions of KICK, DOWN, OPEN, PICKUP, WILED, ZAP. We soon realize that to use the wand picked up in the room after the maze, we must add 16 more TextCharacters actions to cast a correct spell, bringing our action space to 30 actions. At this point, we decided to neglect the entire task and only focus on solving the randomly generated masked maze in the first section of the task. As we played the game with human input, we realized that we did not need the command action OPEN or KICK to open the doors in the map. We can open a door by using the direction action E. However, door resistance is possible, which will require another attempt. Our final action space consists of only four actions of N, W, E, S.. Our final action space consists of only four actions of N, E, S, W.

B. Observation Space

MiniHack supports several forms of observations, including global or agent-centred viewpoints (or both) of the grid. In order to improve training efficiency, we decided to reduce the observation space to pixel, glyphs_crop and chars. The pixel observation is required to generate a game play of the agent per submission requirement. The glyphs_crop observation is an agent-centred 9*9 matrix version of the full glyphs. From our experiment (3 runs of 1 Million timesteps using the same seed), adapting this observation space increases our agent's belief and training speed. The chars observation space is a 21*79 matrix representing the map's characters. As we played the game with human input, we learned vital information about the environment that will aid us in formulating our custom reward function.

Char	Meaning
32	Unexplored / Unreachable
35	Corridor
43	Closed Door
45	Opened Door
46	Room Floor Tile
47	Wand
60	Stair Up (Goal)
61	Ring
62	Stair Down (Start)
64	Player
72	Monster (minotaur)
91	Armor
125	Lava Pool

[1]code available at: https://github.com/Ryan-Alexander03/RL-Project-2022

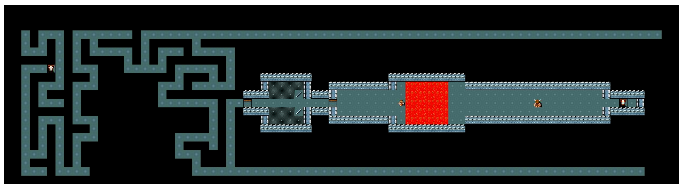

Fig. 1. Pixel observation of MiniHack-Quest-Hard-v0

C. Reward Function

The default reward function of the custom MiniHack Environment is a sparse reward of $+1$ for reaching the staircase down (which terminates the episode). Sparse rewards are frequently employed to avoid human bias when training an RL agent. However, algorithms with sparse rewards are publicly recognized to take enormous time and computation power as the complexity of a problem scales.

We decided to implement some custom reward functions to aid our agent. In addition to the default reward, we award the agent $+1$ when exploring a new corridor block in the maze and increase the penalty reward from -0.01 to -1 for a frozen move. A frozen move is when the agent ends up in the same state as the previous one after a direct action. One thing to notice here, as we spoke before that agent might get a frozen move attempting to open a door, so we added a condition that: if it is a frozen move and the agent is at $(11, 27)$ or $(11, 37)$ (the block before the doors) and the action was E. We add a counter reward of +1.5 by encouraging the agent to take the correct action.

As we can observe in the figure 1, two long corridors are extended from the top to the bottom of the maze. We want to discourage our agents from going into it, as most human agents will do, by giving a reward of -1 when the agent is in either of the long corridors ($x = 3$ or $x = 19$ and $y >= 29$). However, a reward of $+1$ when leaving the long corridor (current y position is less than previous y position).

As we progressed, we added another reward of $+10$ when the agent explored the location of the first door (exit of the maze) and a reward of $+10$ when successfully opening either door. We added a reward of -1 for returning to the maze after a door has opened, and this negative reward doubles when the second door has opened.

At last, for the agent to survive and not accidentally walk into the lava pool, we will award $+2$ for each step it survives in the game after the first door has opened. We give an additional reward of $+2$ for surviving after the second door has opened.

D. Implementation

For implementing the PPO algorithm, we first coded a version utilising TensorFlow to solve a classic control environment, Cart Pole, from the Open AI gym [4]. We modified the code to solve a Box2D environment problem, Lunar Lander. To show the feasibility of our implementation, we compared our method with Stable Baselines3 [5], a set of reliable implementations of reinforcement learning algorithms in PyTorch. We used the same hyper parameter learning rate; n-steps; batch size; discount factor; clip range; n-epochs. We observed that Stable Baselines3 was faster in converges, and compiling speed was much quicker in all five seeds we tested at the cost of higher GPU Video Random Access Memory(VRAM) usage.

E. Parameter Tuning

Initially, we try to test the effect of tuned parameters on a more straightforward task like ”MiniHack-MazeWalk-15x15-v0”. Nonetheless, we observed that the effect of the tuned parameter behaved differently in the more straightforward task and the burdensome one we were trying to solve. So, to evaluate the tuned parameter, we have to let the agent run in the simulated environment for about three million timesteps. Each time on average, takes ± 24 hours.

In PPO, the “batch size” corresponds to how many experiences are used for each gradient descent update. The higher the batch size, the more VRAM the algorithm requires. To take full advantage of the GPU processing, the number of batch sizes should be a power of 2. We compared the result from 2(3GB) to 16(4GB) to 32(6GB) and settled on 64(7.5GB), limited by the GPU we are using, which only has 8GB of VRAM

The “n step” corresponds to the number of steps to run for each environment per update. In our training, we took the value of 512. In our experiment, 128 and 256 showed inconsistent results during the training, and 1024 affected the convergence speed.

The last tuned parameter was “n epochs”, which is the number of passes through the experience buffer during gradient descent. The larger the batch size, the larger it is acceptable to make this. Decreasing this will ensure more stable updates at the cost of slower learning. Since our “batch size” was set to 64, we increased this value from the default of 10 to 16.

F. Result

As we can observe from figure 2. It is indeed learning how to solve the task. Nevertheless, it seems to be taking longer than we anticipated. At over 12 million timesteps (96 Hours), it still has not figured out how to solve the maze (average over $+120$), sitting at a average reward of -380. We believe it might be caused by pixel observation, which adds disproportionate learning parameters. Another reason why we suspect the pixel observation is that, the saved model is a zip file of $1.2GB$. Majority being the policy and policy optimizer.

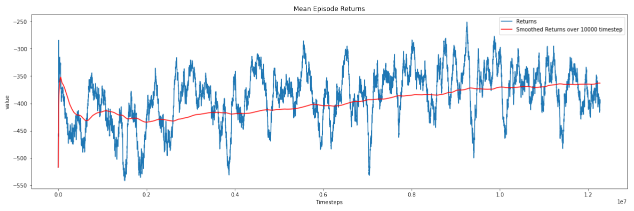

Fig. 2. Training returns over time

IV. Option-Critic Architecture

The Option-Critic Architecture [3] was introduced in 2016. It builds upon the options framework [6]. Options provide a framework for temporal abstraction in RL, options can be considered as macro-actions consisting of a policy, initiation set and termination condition. The Option-Critic architecture is used to learn the internal policies and the termination conditions of options as well as the policy over options. As an aside, For our implementation we decided to learn individual skills and use these as the options in our network. Th reduces the learning required by our Option-Critic model. I hopes that the injected knowledge will speed up learning.

The Option-Critic executes by allowing the agent to pick an option from the policy over options, then follows the internal option policy until termination. The agent is again allowed to pick an option and this cycle repeats. As the model executes, the expected option-value is calculated. Then the gradient of the discounted expected return is calculated. This creates what the authors called the Intra-Option Policy Gradient Theorem and the Termination Gradient Theorem. Thus after an option has terminated if suboptimal with respect to the expected value, the agent can select a better option using their policy over options. This method can be described as a stochastic gradient descent algorithm for learning options

A. Implementation

The MiniHack-QuestHard-v0 environment is a complex learning environment which can be broken up into different sections/sub-tasks such as; exploring a maze, fighting a monster and crossing a lava river. Each of these sub-tasks requires multiple actions to achieve success. As such, we took some of these sub-goals as options for our Option-Critic Network. We trained the agent using the IMPALA framework [7] on the following environments; MiniHack-MazeWalk-9x9-v0, MiniHack-LockedDoor-v0, and MiniHack-LavaCross-v0. These are not all the sub-tasks required to solve the entire MiniHack-QuestHard-v0, but they were chosen based on the relevance to the beginning sections of MiniHack-QuestHard-v0 where the agent is required to solve a maze, move between 2 doors then cross lava using any object allowing levitation or freezing.

We did not restrict the action space while trying in hopes the agent will only learn the necessary action during skill acquisition.

MiniHack-MazeWalk-9x9-v0 (MazeWalk-small) was used to teach the agent basic navigation (eight direction actions N, E, S, W, NE, NW, SE, SW), how to explore and deal with walls as well as memory to be able to actually escape the maze. trained for 4e6 steps. We also trained the agent on the larger mazes but as can be seen in figure 3 the results were not good enough for this to be used for skills.

MiniHack-LockedDoor-v0 was used to teach the agent how to interact with doors in the environment using actions such as OPEN and KICK to open and move past them. Trained for 2e6 steps

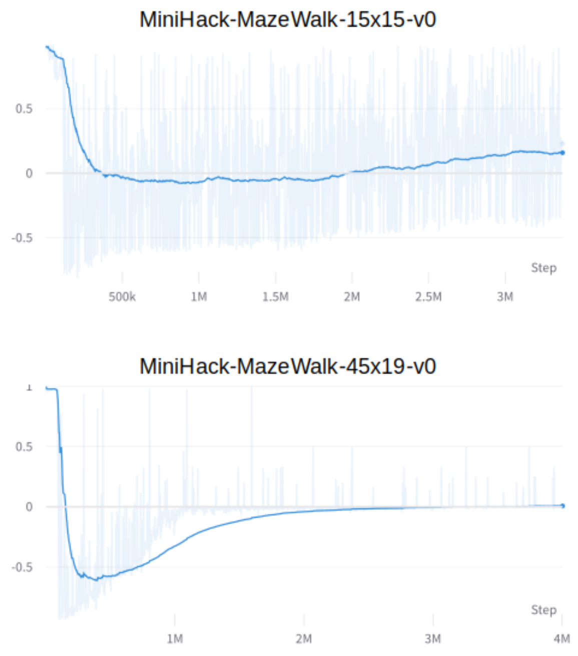

Fig. 3. Average reward per episode for the larger MazeWalk environments

MiniHack-LavaCross-v0 teaches the agent to PICKUP objects such as a potion of levitation or a wand of cold and use it to cross the lava river. Trained for 7e5 steps as it converged much faster than other skills as seen in figure 6

The skills training was done using the open-source TorchBeast agent [8]. The skills transfer and Option-critic network are done using the implementation from SkillHack [9].

The initial skills transfer test was to see if the MazeWalk-small skill could be used to more effectively train on the MiniHack-MazeWalk-Mapped-45x19-v0 environment. It can be seen that this did not help in training. The basic skill was not enough to help achieve a successful agent on the much larger maze. This is most likely due to the fact this maze is exponentially harder than the 9x9 version. A possible method to solving such a large maze would require a larger set of simpler skills to help increase the overall number of skills.

We then used our acquired skills to train on the MiniHack-Quest-Hard-v0 for roughly 6e6 steps.

The hyperparameters for training can be seen in table I

B. Results

As can be seen in figures5, 6, 7 the agent is able to learn the skills which we initially set out to learn. The problem arises when transferring these skills onto the main task of solving MiniHack-QuestHard-v0. This simple set of skills is not enough to complete the task or even speed up learning. It can be seen in the figure 4 that the agent is learning but slowly. Due to time constraints we were unable to train for longer, but it can be seen that the returns were on an upward trajectory.We suspect that the difficulty involved in solving a maze at that

Name	Value
Training settings	
num_actors	256
batch_sieze	32
unroll_length	80
Model Settings	
hidden dim	256
embedding dim	64
glyph type	all_cat
equalize input dim	false
layers	5
crop model	cnn
crop dim	9
use index select	True
max learner queue size	1024
Loss Settings	
entropy cost	0.001
baseline cost	0.5
discounting	0.999
reward clipping	none
normalize reward	True
Optimizer Settings	
learning rate	0.0002
grad norm clipping	40
Intrinsic Reward Settings	
int.twoheaded	True
int.input	full
int.intrinsic weight	0.1
int.discounting	0.99
int.baseline cost	0.5
int.episodic	True
int.reward clipping	none
int.normalize reward	True

TABLE I
HYPERPARAMETERS FOR THE ALGORITHM

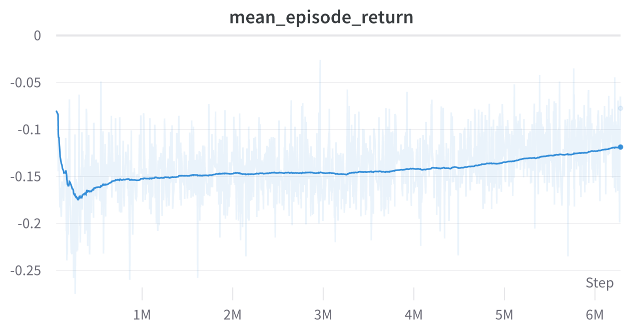

Fig. 4. Average reward per episode for MiniHack-Quest-Hard-v0

scale that keeps changing is too complex and requires learning a more robust set of skills to build an understanding of a general maze environment and how to solve it. So we did not get to test the other trained skills in the target environment.

V. Conclusion

Overall the results were not stellar in regards to actually completing MiniHack-Quest-Hard-v0, but we did show that our approaches resulted in the agents learning. With more training time it is possible that our approaches would much exceed our current results even possibly completing the task.

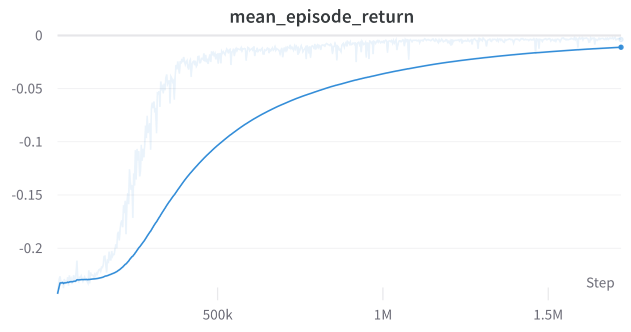

Fig. 5. Average reward per episode for MiniHack-LockedDoor-v0

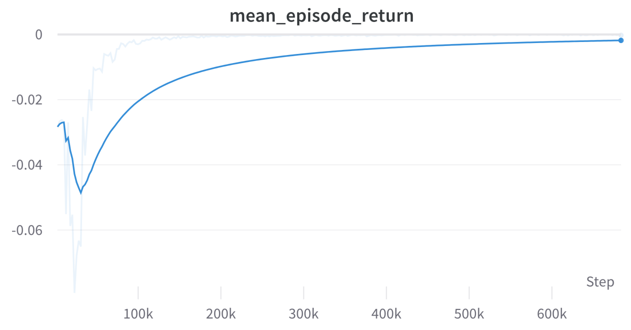

Fig. 6. Average reward per episode for MiniHack-LavaCross-v0

References

[1] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel, "Minihack the planet: A sandbox for open-ended reinforcement learning research," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. [Online]. Available: https://openreview.net/forum?id=skFwlyefkWJ

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

[3] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[5] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, "Stable baselines3," 2019.

[6] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *International conference on machine learning*. PMLR, 2018, pp. 1407–1416.

[8] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, "TorchBeast: A PyTorch Platform for Distributed RL," *arXiv preprint arXiv:1910.03552*, 2019. [Online]. Available: https://github.com/facebookresearch/torchbeast

[9] M. Matthews, M. Samvelyan, J. Parker-Holder, E. Grefenstette, and T. Rocktäschel, "Hierarchical kickstarting for skill transfer in reinforcement learning," *arXiv preprint arXiv:2207.11584*, 2022.

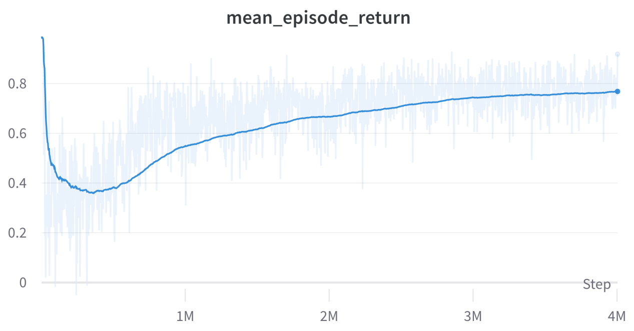

Fig. 7. Average reward per episode for MiniHack-MazeWalk-9x9-v0