Picker: Reinforcement learning architecture that combines and enhances existing policy for complex problem

Yoon-Chae Na

December 15, 2022

Abstract

Reinforcement learning(RL) is attractive and promising, but it takes lots of learning costs. Moreover, to apply RL to the real world, an agent should do complex tasks that are challenging to learn. Therefore, a hierarchical structure that recycles and combines existing policies can be a solution. We suggest a picker methodology that picks the best policy in existing sub-policies. When targets of sub-policies are independent, the picker makes greedy choices to adapt to its various environments. When the target of sub-policies is similar, the picker makes the rational choice to make synergies. In order to make the better merging of sub-policies, we introduce a picking term trick to do deep exploration.

1 appendix

Reinforcement Learning(RL) might be one of the best AI methodologies. Through self-playing by the agent, it does not need domain data to improve the agent. Through simulation, the agent can get adaptive and less-needed domain knowledge. However, it has one fatal shortcoming: long learning time. Furthermore, as the complexity of the problem increased, the amount of time required increased exponentially. For example, it is simple to teach kicking the ball or walking to the ball to a soccer robot through RL. However, teaching to play soccer using RL is challenging through one reward function from scratch. There still needs to be a champion method for how to teach individual skills (e.g., kicking, walking) and then combining them. Below two questions are the main topic of this research. Could the agent, who has already acquired each skill, learn a new task or improve by picking the existing skills? Does the skill-picker have the intelligence to make the proper choice?

2 Related Work

This section covers the alpha-star(Oriol et al. 2019), which greatly motivated this research [5].

Alpha-star: AI architecture
StarCraft is a strategy simulation game and is famous for its high complexity. In 2019, the google deep-mind team finally made a superhuman-level agent using RL. They merged many AI methods, and this is worth our attention. They divided their model into three parts at the high architecture level. (1) Input agent: gathering current game information, including map image data through MLP, transform, and Resenet. (2) Brain agent: This agent has theta, which could be taught through RL. When it gets the information from the input agent, it calculates the action signal using theta. (3) Output agent: It has a sequential structure to deal with the output signal from the brain agent. For example, the first part of the output agent chooses 'action type', such as an attack, defense, and move. The second part chooses 'when', the third part chooses 'which units', and the fourth part chooses 'target'. This functionally divided and sequentially processed structure help to merge various AI models. One more point to attention in this paper is deep exploration. Because of the high complexity of Starcraft, the exploration in RL interfered with learning. So, the deep-mind team used the pseudo reward strategy-z, which gave rewards for not changing the policy frequently.

3 Picker

Purpose
- Design to reuse the existing policies to save train cost
- Make synergy between existing policies to improve the entire agent

Architecture
The idea behind the picker is straightforward. Existing RL policies are positioned as sub-policies. The picker agent picks one of the sub-policy as its policy instead training from scratch. Therefore, the size of the picker agent action space is the number of sub-policy. For example, when the picker action value is one, the picker's observation value passes to the first sub-policy to get the agent action value. Through the picker's neural network structure(DQN), the picker can learn to get wise selec-

tion. Sub-policies' targets, such as kicking, walking to a ball, and sliding, could be irrelevant. Also, sub-policies targets could be the same, but sub-policies' proprieties could be different. For example, though a target is the same for walking far, each sub-policy prosperity could be using the front leg only, behind leg well, or moving fast.

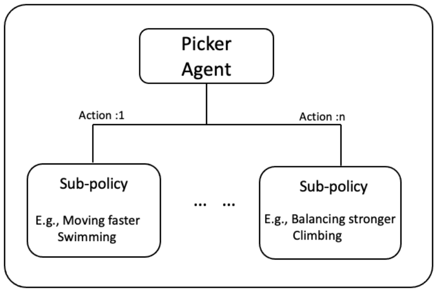


Figure 1: The architecture of picker

4 Methodology

This section is organized as follows. In Sec. 4.1 we show the methodology to make sub-policies to help understand experiments. In Sec. 4.2 we introduce the picker method and deep exploration.

4.1 Making sub-policy

We implement several well-known RL algorithms using the stable baselines library as a sub-policy. We experiment with Proximal Policy Optimization (PPO) (Schulman et al. 2017) [4], Trust Region Policy Optimization (TRPO) (Schulman et al. 2015) [3], and Soft Actor-Critic Algorithms(Thomas et al. 2019)[1]. Each model is trained with two policy networks; one is MlpPolicy given by the library, and the other is a customized network. The detailed hyperparameter setting is in the appendix. We train AntBullet agents in the pybullet library until 100 iterations and save half of each iteration's policy to do picker experiments. The reward of each model is in Figure 2.

4.2 Making Picker and deep exploration

Picker is implemented based on DQN with three action spaces; action 0: picking PPO, action 1: picking TRPO, and action 2: picking SAC. DQN algorithm is customized PyTorch DQN. We attach the pseudocode in Algorithm1. Unfortunately, when the picker chooses a sub-policy for every step, the picker makes the decision not a rational choice but a random one. Furthermore, the picker agent performance worse than naive sub-policies. So, we use a deep exploration method like Oriol et al. did in alpha stars. After picking the sub-policy, the method 'picking term' does not allow the picker to pick again for specific steps; 1, 3, 6, 8, 12, 15. This method is quite useful, and the result will cover in Sec.5.

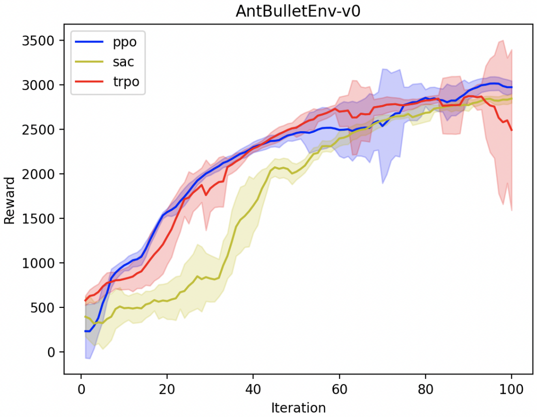


Figure 2: The sub-policies rewards on AntBulletV0

Algorithm 1 Picker agent with deep-exploration

```
Initialize Q(s,a), for all s,a, arbitrarily except that Q(terminal,·) = 0
for each episode : do
    Initialize S
    for each step of episode : do
        if step % Picking term == 0 then
            Take action A, observe R, S'
        end if
        if step % Picking term != 0 then
            observe R, S'
        end if
        Choose A from S using policy derived from Q (e.g., $\epsilon$-greedy)
        Take action A (sub-policy), observe R, S'
        Q(S,A)$\leftarrow$Q(S,A)+$\alpha$[R+$\gamma max_a$Q(s,a')-Q(s,a)]
        S$\leftarrow$ S'
    end for
    until S is terminal
end for
```

5 Empirical Study

This section shows the result of the experiments. Below three questions are the main topics for this section.

- Could the picker architecture merge the sub-policy

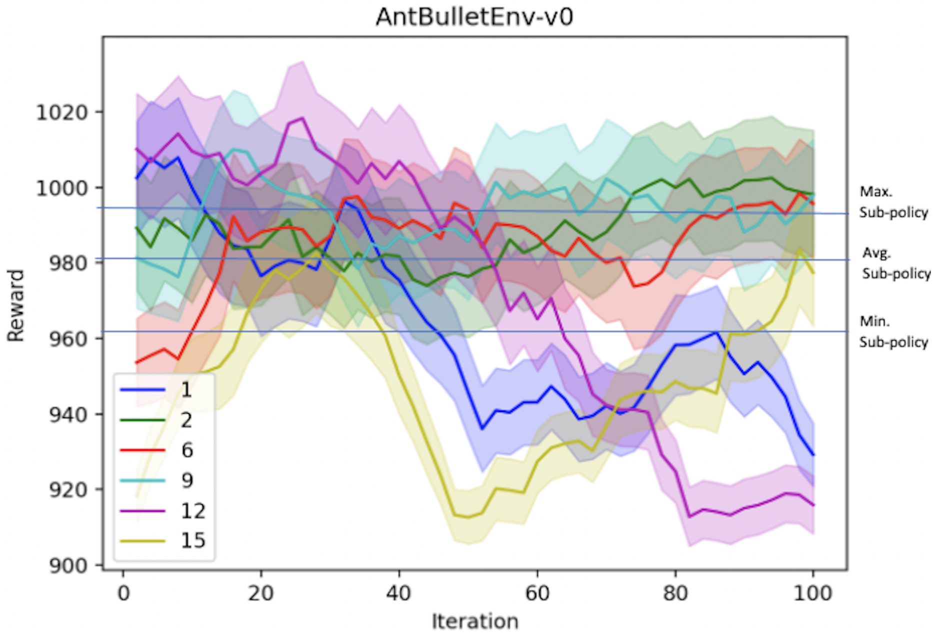


Figure 3: The picker rewards: deep-exploration

properly?

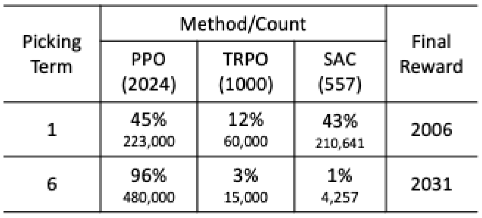
- When sub-policy targets are significantly different, could the picker make a greedy decision?
- When sub-policy targets are similar, could the picker make a rational decision?
- Does deep exploration help the performance of the picker agent?

Picker architecture validation

Picker architecture is valid and able to merge the sub-policies. Before doing experiments, we were concerned that the picker methodology might damage sub-policy qualities or that the pybullet agent would be fallen over in replacing the sub-policy. We experiment with various hyper-parameters, and every experiment's performance of picker methodology is approximately ±20% of sub-policies performance. Let us see the Figure 3. Sub-policy rewards are 963, 983, and 996. The picker agent reward range is from 900 to 1020. In Figure 3, rewards are calculated by ten times the average of evaluations. We use a moving average of 5 steps to draw Figure 3 like Peter et . all did deep reinforcement learning that matters research[2]. These results represent that if we already have a policy, we could use it as a sub-policy with picker architecture to save training time rather than re-training everything from the beginning.

Picker could be greedy : sub-policy differ

The picker agent is valid by seeing the greedy selection when sub-policies differ under certain circumstances. Suppose we implemented the triathlon agent using the picker structure. In the mountain observation space case, the picker should select the climbing sub-policy, and the picker should select the swimming sub-policy in the water observation space. To do this experiment, We set three sub-policies; PPO(2024), TRPO(1000), SAC(557), and count the picker agent selections. If the picker chooses PPO, which has the highest reward, the picker's choice is rational. Let us see the figure 4. When picking term six, agent selection is greedy, and that makes sense. Therefore, we could use the picker architecture to train for the triathlon. That method is more efficient than teaching one agent all three. Furthermore, the picker will find the pattern and automatically pick the most desirable sub-policy without domain knowledge.



Picking Term	Method/Count			Final Reward
	PPO (2024)	TRPO (1000)	SAC (557)	
1	45% 223,000	12% 60,000	43% 210,641	2006
6	96% 480,000	3% 15,000	1% 4,257	2031

Figure 4: The number of picker's selection: sub-policy differ

Picker could be rational : sub-policy similar

The picker agent can make synergy between the similar goal of sub-policies with 60% confidence; detailed normal distribution analysis results are in figure 6. We designed this experiment to check whether sub-policies can achieve one goal and create synergies when the sub-polices have the same target and similar expected rewards. We set three sub-policies; PPO(996), TRPO(963), SAC(983), and count the picker agent selection. When the picking term is one, the picker methodology reward is 962, the same with a mean value of sub-policy(962). However, when the picking term is 3(1024), 6(1026), 9(1016), and 15(1015), the picker methodology is higher than the max value of the sub-policy(996). We can see that the selection ratio variety depends on picking terms in figure 5. The effect of the synergy is weak but present. When Oriol et. all made the alpha-star agent, they blended many tricks and methods. We also need more methods and tricks for an effective hierarchical reinforcement structure. This picker methodology could be a trick to complete the hierarchical RL structure.

Deep exploration effect

This deep exploration method is simple but effective. In figure 4, when we do not use the picking term method(picking term is 1), the picker's selection is not greedy but evenly distributed, contrary to our expectation. Sub-policy SAC reward is 557, but the picker picks it 43%. Figure 5 shows almost the same result. Three sub-policy selection ratio is almost evenly; 39%, 25%, 35%. However, when picking terms are 3 to 9, the picker choice makes a rational choice, and the picker agent's final rewards are higher than the reward of sub-policies. One more interesting point is that figure 3 shows that if the picking term is longer than ten steps, the agent's reward gets worse.

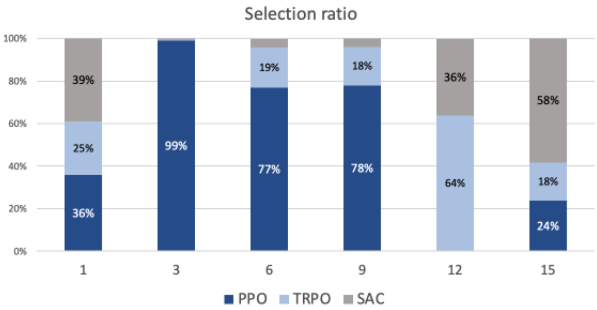
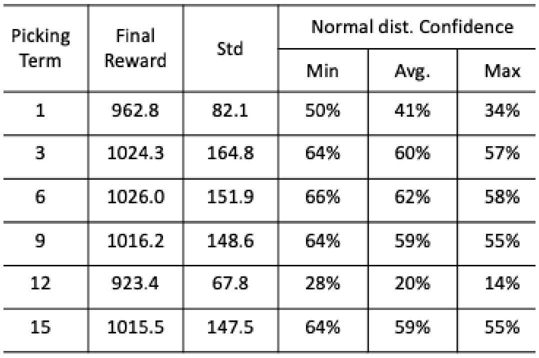


Figure 5: Selection ratio

6 Conclusion

In order to apply RL to the real world, the agent needs to do complex tasks. Hierarchical RL that recycles and develops with sub-policies would be a good solution. We show that sub-policy recycling is possible without specific domain knowledge by using the picker method.



Picking Term	Final Reward	Std	Normal dist. Confidence		
			Min	Avg.	Max
1	962.8	82.1	50%	41%	34%
3	1024.3	164.8	64%	60%	57%
6	1026.0	151.9	66%	62%	58%
9	1016.2	148.6	64%	59%	55%
12	923.4	67.8	28%	20%	14%
15	1015.5	147.5	64%	59%	55%

Figure 6: Normal dist.analysis of picker method

When sub-policies targets are different, the picker can merge sub-policies by greedy picking depending on observation space. When sub-policies targets are similar, the picker could make synergies between existing sub-policies. We show that picking terms as a deep exploration is quite effective, and picking terms should not be too long or too short. However, this experiment is just beginning. We need more variety of experiments to check by combining the agent who cannot use the front leg and who cannot use the behind leg to say the picker agent can make synergy. Also, we need to check various deformed physical environment simulations to say that the picker agent adapts well. We hope to share more information through future works.

References

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[2] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[3] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[5] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster

level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Appendix : Hyper-parameter

Sub-policy hyperparameter

1) PPO
Library : $Stable_baseline3$
Policy network hyper-parameters 1) 'MlpPolicy' (built-in) 2) activation fn=th.nn.LeakyReLU,net arch=[64,64,dict(vf=[64,64], pi=[64,64])]
2) TRPO
Library : $sb3_contrib$
Policy network hyper-parameters
1) 'MlpPolicy' (built-in) 2) activation fn=th.nn.LeakyReLU,net arch=[64,64,dict(vf=[64,64], pi=[64,64])]
3) SAC
Library : $Stable_baseline3$
Policy network hyper-parameters 1) 'MlpPolicy' 2) activation fn=th.nn.LeakyReLU,net arch=dict(pi=[64, 64], qf=[400, 300])

Picker hyperparameter

BATCH SIZE = 128
GAMMA = 0.99
EPS START = 0.9
EPS END = 0.05
EPS DECAY = 1000
TAU = 0.005
LR = 1e-4