人工知能学会研究会資料
SIG-AGI-008-07

Sub-policy pruning in Meta Learning Shared Hierarchies

Qing HONG [1, 2*] Yusuke TANIMURA[2,1] Hidemoto NAKADA[2,1]

[1] University of Tsukuba
[2] National Institute of Advanced Industrial Science and Technology

Abstract: MLSH (Meta Learning Shared Hierarchies) is a meta learning method that divide a policy into multiple sub-policies and a master-policy that picks one of the sub-policies to be actually used. By training sub-policies in advance, master policy can rapidly adjust to given environments. However, MLSH is not suitable for complicated environments. In complicated environments, a number of sub-policies are required, and it is very difficult to train them properly. We propose a method to effectively prune excessive sub-policies to give better chance the other sub-policies to be trained. We demonstrate that we can prune 40 % of sub-policies while preserving the performance.

1 Introduction

Meta Learning Shared Hierarchies [1] is a new architecture which tries to challenge a wide variety of tasks by using prior knowledge and primitive policies to adjust new task quickly. By setting the agents to solve distribution of related work and sharing information between different tasks. it successfully reaches the good result by training new policy ceaselessly.

Each distribution tasks have specific optimal policies. As the training goes on, the algorithm's body will become large, it means a large number of the primitives will decrease the learning speed. While the environment is complicated, we also need a huge number of sub-policies to cover the unseen tasks.

Therefore, our challenge is finding a proper number of primitives to keep the architecture stay an efficient state. We proposed a method called Multi-layered Hierarchical Architecture to prune the idle primitives in a huge number of policies. Each primitive will be trained iteratively and the body will keep grows. in the subsequent works, the new policy also needs cost extra time to keep these policies. accordingly, we try to find an efficient way to reduce the number of them and keep the agent stay the same performance.

2 MLSH

2.1 Architecture

Meta Learning Shared Hierarchies is an end-to-end meta-learning approach that uses prior knowledge. Similar to the Option framework [2], they pre-train the primitives to allow master policy to quickly learn on new tasks.

Figure 1 shows the overview of the MLSH. First, MLSH creates a set of primitives to lead an optimization problem called sub-policies. These sub-policies will share information with each other and could handle the task solely. In order to enable an agent to learn on tasks sampled from a specific task distribution, one of the sub-policies is activated to maximize the reward. That is to say, we create a stochastic policy called master policy whose action is to choose one from the sub-policies. Each master policy will start with randomly-initialized parameter and fine-tuned to the task, then run a warmup period to optimize such random parameter. At last, the algorithm updates sub-policies and master policy in the joint section, then passes the sub-policies to the next master-policy. By learning new master policy ceaselessly we could reach the purpose: quickly reach high reward on unseen tasks.

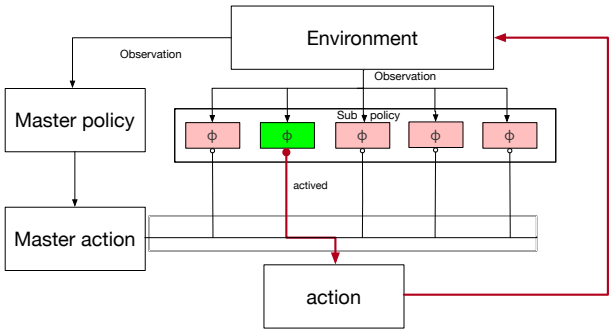

Figure 1: The architecture of MLSH [1]. θ represents the master policy, ϕ represents sub-polices

2.2 Algorithm

Figure 2 shows the overview of the MLSH algorithm. Master policy starts from a randomly initialized state, the algorithm will iteratively perform a warmup update and a joint update in order to update master-policy θ and sub-policies ϕ. the master-policies is fine-tuned to the task and repeat these periods until getting convergence.

*Address: University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki , 305-8577
E-mail: kousei19920804@gmail.com

Meta Learning Shared Hierarchies

initialize ϕ
repeat
  Initialize θ
  Sample task M $\sim P_M$
  for w=0,1,...W (warmup period)do
    Collect D timesteps of experience
    Update ϕ to maximize expected return from 1/N timescale
  end for
  for u=0,1,....U (joint update period)do
    Collect D timesteps of experience
    Update θ to maximize expected return from 1/N timescale
    Update ϕ to maximize expected return from full timescale
  end for
until convergence

Figure 2: The algorithm of MLSH.

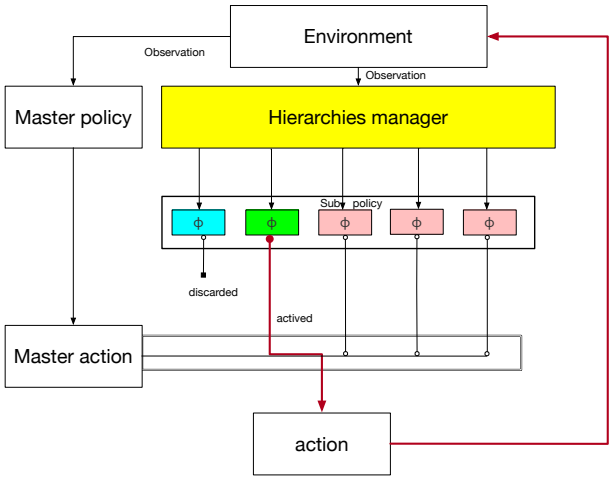

Figure 3: Our proposed method: Hierarchies manager will mark the idle sub-policies and make it dormant.

Table 1: Number of actived times with 6 sub policies.

	$sub1$	$sub2$	$sub3$	$sub4$	$sub5$	$sub6$
$2hours$	162	169	158	161	166	162
$4hours$	380	375	405	376	329	372
$8hours$	776	748	799	746	681	692

In the warmup update period, it will record D time steps of experience. Then it actives the sub-policies and maximizing reward by an arbitrary reinforcement learning algorithm (such as DQN [3]) and use this reward to update θ.

In the joint update period, master policy will be created as an extension of the environment, it contains collect experience and optimize θ in the warmup period, then reuse the same experience viewed from the perspective of the sub-policies. For each episode, it only updates the parameters of the sub-policy that had been activated by the master policy.

3 Problem Statement

MLSH keep training the sub-policies iteratively to keep agent can reach the high reward. One thing is that each sub-policy start in the unseen scene, we can't guarantee each sub-policy can reach average performance at first. Therefore we expect an appropriate number of sub-policies to cover the whole environment. Too little number of sub-policies will lead the master-policy hardly getting convergence. In the same way, if we keep a huge number of sub-policies, most of them will go idle.

Table 1 shows the results we train the MLSH in a suitable number of sub-policies. This result shows that when the sub-policies could cover the distributed tasks, each sub-policy can play an effective role. But in actual operation, we can not estimate the number of sub-policies we need.

Table 2 shows the results is train the MLSH in the same tasks, but we increase the number of sub-policies. After a period of training time, this balance is broken because master-policy will prefer the sub-policy whose action is better in the subsequent study. This vicious cycle will cause some of the sub-policy been idle after several attempts. While agent tries to train a new master-policy, it also costs time repeat this process. Each sub-policy has the ability to finish a task by itself, it means in the last sub-policies will bring large body and new master-policy will cost more time to getting convergence. On many occasions, one sub-policy could replace other. Accordingly, we desire an easy way to solve this problem. We will prove our hypothesis in the experiment section.

4 Proposed Method

We would like to make the potential sub-policies have greater chance to be chosen from master-policy. Oppositely, we need to prune the idle one. As the modification to MLSH two-level hierarchical architecture, we propose a sub-policy pruning method to solve such problems. This architecture is similar to hierarchical-DQN [4], we use a hierarchical manager to receive extrinsic reward and decide the rule to prune the sub-policies with low-activity. As a preliminary, we assume having access to a rule detector that provides plausible candidates.

At first, We train a large number of sub-policies in the unseen distributed tasks. The hierarchical manager prunes the efficient policies and idle policies after several episodes until reaching saturation. Training new master policy ceaselessly to find a way to let activate sub-policies to take the place of others. Our method focuses on 2 points: keep the same efficiency and find the appropriate number of sub-policies.

Table 2: Number of actived times with 10 sub policies.

	$sub1$	$sub2$	$sub3$	$sub4$	$sub5$	$sub6$	$sub7$	$sub8$	$sub9$	$sub10$
$2hours$	125	118	119	164	173	116	125	115	111	147
$8hours$	430	494	467	503	505	544	564	665	464	803
$16hours$	990	1008	943	1149	1011	1427	1507	1789	937	1885

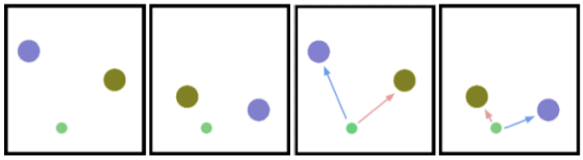

Figure 4: The gym of 2D moving bandits (This diagram is taken from [1]). The gym provides green dot as the agent. The agent's goal is to get as close as possible to the blue and yellow dots. Different arrows shows the action generated by different active sub-policies.

5 Experiment

We design the experiments to confirm two things: 1) An appropriate number of sub-policies can accelerate the speed of convergence. 2) Our proposed method could reduce the redundant sub-policies and make the agent stay the same efficiency. As a preliminary experiment, we conduct sampled tasks from 2D moving bandits, shown in Figure 4.

At first, we run the MLSH in the 2D moving bandits with a different number of sub-policies and let hierarchical manager assume the appropriate values. Then we pre-train our sub-policies with ten and save the checkpoint to finish our contrast experiment. we record the checkpoint that agent starts getting convergence. At last, we start two agents at the same checkpoint: our proposed method and the old one. We set the warm update period as a 10 time step and joint update period as a 20 time step. Train the agent for 434 trails and record the result.

6 Result

From Figure 5, we can see the first agent which use appropriate number of sub-policies converges faster.

Figure 6 shows that we can prune lowly utilized sub policies without sacrificing the performance. We prune 30% sub-policies at first the first vertical line, after several trials when the agent reaches a new balanced point. Then we successfully prune the 10% of sub-policies in the second vertical line. After that, the system still works well and each sub-polices have balanced chance to be chosen.

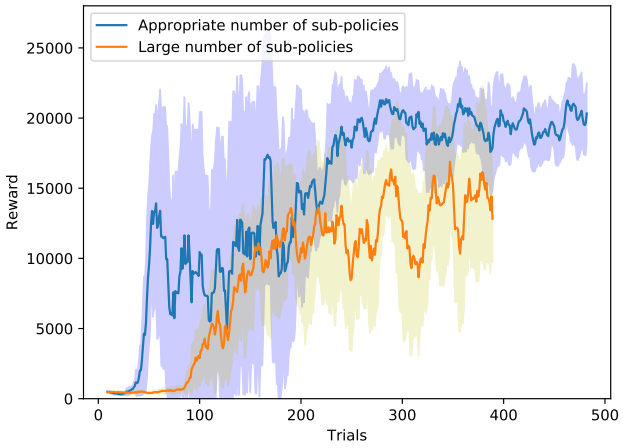

Figure 5: Comparison with two sub-policies with ten sub-policies. The x-axes denotes the number of trials, while the y-axes denotes the reward.

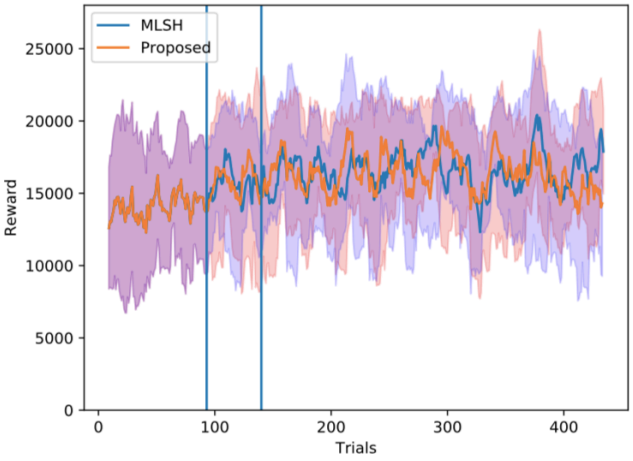

Figure 6: Comaparison with MLSH and the proposed method. We prune the sub-policies in two trial point: 93 and 140 (the vertical line above the x-axes).

7 Conclusion

As a primary experiment, we successfully conducted two experiments to confirm our hypothesis. Different tasks have the different capacity of sub-policies, exceeding the capacity will slow the converging speed. On the other hand, the too small quantity of sub-policies can hardly meet our needs. It is necessary to find a good way to determine the appropriate number of sub-policies before we explore an unseen task. By deploying a large number of sub-polices and prune the low-activity sub-policies is an efficient way to lower the upper limit and then help the hierarchies manager to assume the best range of sub-policies which could brace the hole environment.

In this paper, we used proposed method to prune the low-activity sub-policies. However, some sub-policies always perform identically. We need to find a better way to detect them and performs further pruning in order to reach the minimum number of sub-policies.

Acknowledgement

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

This work was supported by JSPS KAKENHI Grant Number JP16K00116.

References

[1] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *CoRR*, abs/1710.09767, 2017.

[2] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *CoRR*, abs/1609.05140, 2016.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[4] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016.