HAT: Hierarchical Alternative Training for Long Range Policy Transfer

Wei-Cheng Tseng [1] **Jin-Siang Lin** [1] **Yao-Min Feng** [1] **Min Sun** [1]

Abstract

Humans can master a new task within a few trials by drawing upon skills acquired through prior experience. To mimic this capability, hierarchical models combining primitive policies learned from prior tasks have been proposed. However, these methods fail short comparing to the human's range of transferability. We propose Hierarchical Alternative Training (HAT), which leverages the hierarchical structure to train the combination function and adapt the primitive polices alternatively, to efficiently produce a range of complex behaviors on challenging new tasks. We demonstrate that our method outperforms recent policy transfer methods by combining and adapting these reusable primitives in tasks with continuous action space. The experiment results further show that HAT provides a broader transferring range.

1. Introduction

Reinforcement learning (RL) has lots of success in various applications, such as game playing (Brockman et al., 2016; Silver et al., 2017; Mnih et al., 2015), robotics control (Tassa et al., 2018; Coumans & Bai, 2016–2019), molecule design (You et al., 2018), and computer system optimization (Mao et al., 2019a;b). Typically, researchers use RL to solve each task independently and from scratch, which makes RL confronted with sample efficiency. However, compared with humans, the transferability of RL is limited. Especially, humans can learn to solve complex continuous problems (both state space and action space are continuous) efficiently by utilizing prior knowledge. In this work, we want agents to efficiently solve the complex continuous problem by exploiting prior experiences that provide structured exploration based on effective representation.

To this end, we formulate transfer learning in RL as following. We train a policy with one of the RL optimization strategies on the pre-training task. Then, we intend to leverage the policy to master the transferring task. However, transfer learning in RL may face some fundamental problems. First, unlike supervised learning, the transitions and trajectories are sampled during the training phase based on the interacted policy (Rothfuss et al., 2019). Since the reward distributions are different between the pre-training task and the transferring task, directly finetuning the pre-training policy on transferring tasks may make the agent perform biased structured exploration and get stuck in many low reward trajectories. Second, dynamics shifts between pre-training and transferring tasks may induce the pre-training policy to perform unstructured exploration (Clavera et al., 2019; Nachum et al., 2019). Although domain randomization (Tobin et al., 2017; Nachum et al., 2019) in the pre-training phase may mitigate this problem, we prefer the pre-trained policies to gradually fit the transferring tasks consistently.

In this work, we propose Hierarchical Alternative Training (HAT), a training strategy for transfer learning with hierarchical policy. Our pre-training method leverages existing hierarchical structure in the policy consisting of a combination function and a set of primitive policies. Notice that we do not use reference data since we expect our method to be generally applicable to all tasks. In many cases, such as flying creatures (Won et al., 2018), Laikago robot[1] or D'Kitty robot[2], reference data is hard to obtain. During the transferring phase, we alternatively train the combination function and the primitive policies. This training procedure makes the training not only stable but also flexible in exploration. When training the combination function and freezing primitives in the transferring phase, it utilizes the benefit of the hierarchical structure that abstracts the exploration space. When training the primitives and fixing the combination function, the primitives can be adapted to the transferring task. In our experiment, we demonstrate that training hierarchical policy with HAT significantly increases sample efficiency compared to previous work (Peng et al., 2019). Moreover, our method provides a better transferring range.

[1]Department of Electrical Engineering, National Tsing Hua University. Correspondence to: Wei-Cheng Tseng <weichengtseng@gapp.nthu.edu.tw>.

Accepted by BIG Workshop in ICML, 2020.

[1]http://www.unitree.cc/e/action/ShowInfo.php?classid=6&id=1

[2]https://www.trossenrobotics.com/d-kitty.aspx

2. Preliminaries

We consider a multi-task RL framework for transfer learning, consisting of a set of pre-training tasks and transferring tasks. An agent is trained from scratch on the pre-training tasks. Then it applies any skills learned during pre-training to the transferring tasks. Our objective is to obtain and leverage a set of reusable skills learned from the pre-training tasks to enable the agent to be more effective at the later transferring tasks. We denote s as a state, a as an action, r as a reward, and τ as a trajectory consisting of actions and states. Each task is represented by a dynamics model $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$ and a reward function $r_t = r(s_t, a_t, g)$, where g is the task-specific goal such as the target location that an agent intends to reach and a terrain that an agent needs to pass. In multi-task RL, goals $\{g\}$ are sampled from a distribution $p(g)$. Given a goal g, a trajectory $\tau = \{s_0, a_0, s_1, ..., s_T\}$ with time horizon T is sampled from a policy $\pi(a|s, g)$. Our objective is to learn an optimal policy π^* that maximizes its expected return $J(\pi) = \mathbb{E}_{g\sim p(g),\tau\sim p_\pi(\tau|g)}[\Sigma_{t=0}^{T}\gamma^t r_t]$ over the distribution of goals $p(g)$ and trajectories $p_\pi(\tau|g)$, where $\gamma \in [0, 1]$ is the discount factor. The probability of the trajectory τ is calculated as $p_\pi(\tau|g) = p(s_0)\prod_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi(a_t|s_t, g)$, where $p(s_0)$ is the probability of the initial state s_0. In transfer learning, despite the same state and action space, the goal distributions, reward functions, and dynamics models in pre-training and transferring tasks are subjected to be different. The difference between the pre-training and transferring tasks is referred to as the range of transfer.

3. Method

We will first introduce our core hierarchical training method, Hierarchical Alternative Method (HAT), in section 3.1. Then, we show how to apply HAT to the existing hierarchical policy in section 3.2.

3.1. Hierarchical Alternative Training

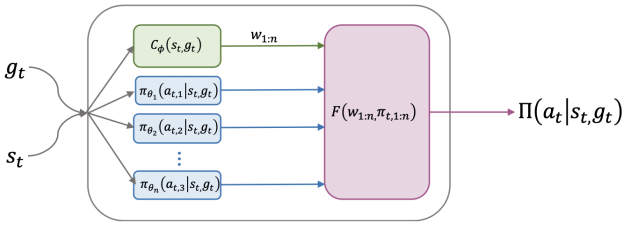

Figure 1. The hierarchical policy architecture.

We propose a training strategy, hierarchical alternative training (HAT). It can be combined with an existing hierarchical policy framework to increase the transferring efficiency and extend the transferring range. Let's consider a basic form of a hierarchical policy (see figure 1). It will contain: 1) A set of primitive policies $\pi_{\theta_1}(a|s, g), \pi_{\theta_2}(a|s, g), ..., \pi_{\theta_n}(a|s, g)$ with parameters $\theta_{1:n}$, and each primitive is an independent policy that output action distribution base on s and g. 2) A combination function $C_\phi(s, g)$ with parameter ϕ outputs weight $w_{i:n}$, where w_i specifies the importance of primitive π_{ϕ_i}. $F(\pi_{1:n}, w_{1:n})$ specifies how to combine primitives with specified weight $w_{1:n}$. Typically, the larger the weight w_i, the more contributions from primitive π_{θ_i}.

Algorithm 1 Applying HAT

```
Initialize θ_{1:n} and φ;
while not converged do
  Disable the gradient of primitives;
  Enable the gradient of combination function ;
  for i = 1 to p do
    train combination function φ;
  end for
  Disable the gradient of combination function;
  Enable the gradient of primitives;
  for i = 1 to p do
    train primitives θ_{1:n};
  end for
end while
```

During the pre-training phase, we train the hierarchical policy end-to-end. The behavior of the policy will be decomposed into a set of primitives, and the combination function will learn to compose the primitives to form a complete behavior. During the transferring phase, we reinitialize the combination function and leverage the primitive directly from pre-trained in the pre-training task. We first update the combination function and freeze the primitives. This views the combination function as the policy that learns to combine the primitives to master the transferring task. As illustrated in the motivating example, the range of transfer between the pre-training and transferring tasks is likely to limit the performance of only training the combination function. Therefore, after p iterations, we switch to finetune the primitives with the combination function fixed. This method makes the primitives become more applicable to the transferring task. After p iterations, we freeze primitives and train combination function again. This is to prevent the skills in the primitives to be severely forgotten during finetuning. The strategy is repeated several times until the hierarchical policy converges (see algorithm 1).

3.2. Applying HAT to the Policy Architecture

We leverage the multiplicative combination rule (Peng et al., 2019)

$$F(\pi_{\theta_{1:n}}, w_{1:n}) = \frac{1}{Z(s, g)} \prod_{i=1}^{k} \pi_{\theta_i}(a|s)^{w_i}, \qquad (1)$$

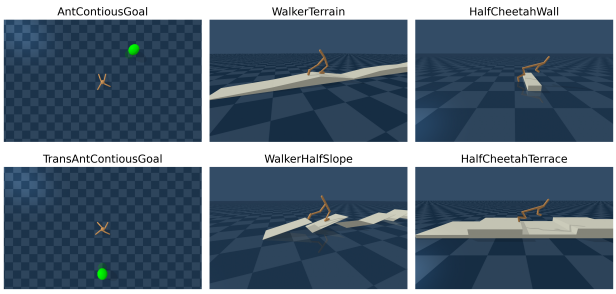

Figure 2. Environments used to evaluate our method. The first row is the pre-training tasks, and the second row is the transferring tasks.

where $\pi_{\theta_{1:n}}(a|s)$ is the primitive policies and $w_{1:n}$ is generated from combination function $C_\phi(s, g)$. $Z(s, g)$ is the partition function that ensures the composite distribution is normalized. $F(\pi_{\theta_{1:n}}, w_{1:n})$ multiplies the primitive policies along with their corresponding weights. The weights determine the importance of each primitive policies to compose the action distribution at a time step, with a larger weight representing a larger influence. Note that to make the primitives task-agnostic (i.e., more transferable), we restrict the primitives $\pi_{\theta_{1:n}}(a|s)$ to only get s. During the pre-training phase, the combination function and primitive policies are trained in an end-to-end fashion. During the transferring phase, we train the combination function and primitive policies alternatively as described in section 3.1.

4. Experiments

In this section, we introduce the evaluation tasks in section 4.1 and list the baselines that we intend to compare in section 4.2. The results of the methods evaluated in our environments are discussed in section 4.3. Aside from option-critic (Bacon et al., 2017), all the experiments are trained with PPO (Schulman et al., 2017) and Generalized Advantage Estimation (GAE) (Schulman et al., 2016). We further show that our method has a broader transferring range compared with other baselines in section 4.4.

4.1. Tasks

We consider three agents (see figure 2): quadruped (ant) with 12 DoF and 8 actuators; 2dwalker with 6 DoF and 6 actuators; halfcheetah with 6 DoF and 6 actuators. All the task is built with PyBullet (Coumans & Bai, 2016–2019).

4.1.1. Pre-training tasks

AntContinuousGoal: An ant needs to move to the target position which is the task-specific information g, and the target direction is sampled from $[\frac{-\pi}{2}, \frac{\pi}{2}]$ with radius 5. Once the ant reaches the goal position, the goal position will be re-sampled in the same way.

WalkerTerrain: A 2d walker needs to move forward on terrain, and the slope of the terrain is sampled from a specific range. The task-specific information g is the terrain in front of the agent. Therefore, the 2d walker needs to learn how to walk smoothly on planes with different slopes.

HalfCheetahWall: A halfcheetah needs to move forward, and there is a wall every 3 to 5 meters. The height of the wall is sampled from a specific range. The task-specific information g is the terrain in front of the agent. Therefore, the halfcheetah needs to climb or jump over the wall.

4.1.2. Transferring tasks

TransAntContinuousGoal: An ant needs to move to the target position, and the target position is sampled from $[\frac{5\pi}{6}, \frac{7\pi}{6}]$ with radius 5. The target direction does not overlap with that of the pre-training task. Once the ant reaches the goal position, the goal position will be re-sampled in the same way. Therefore, in this case, the difference between pre-training task and transferring task is goal distribution $p(g)$.

WalkerHalfSlope: A 2d walker needs to move forward in terrain, but there are cliffs between each plane. Therefore, the 2d walker should be robust to these cliffs. Therefore, in this case, the differences are goal distribution $p(g)$ and dynamics $p(s_{t+1}|s_t, a_t)$.

HalfCheetahTerrace: A halfcheetah needs to move forward on a terrace that is formed with lots of horizontal platforms with target speed, so it needs to jump or climb up to the higher platform and does not fall to the lower platform. The difference in height between two platforms is sampled from a specific range. Therefore, in this case, the differences are goal distribution $p(g)$ and dynamics $p(s_{t+1}|s_t, a_t)$.

4.2. Baselines

We define the baselines that will be discussed in the following sections. **Scratch**: We directly train a policy on the transferring task, and it is one of the most straightforward methods to tackle a task. **Finetune**: We first train a policy in the pre-training task. Then, we directly finetune the policy in the transferring task. It is the other most straightforward method to tackle a task. **MCP** (Peng et al., 2019): We end-to-end train a hierarchical policy with multiplicative combination rule in pre-training task. During the transferring phase, we freeze the primitives trained in pre-training task and only reinitialize and train the combination function. **Option-Critic** (Bacon et al., 2017): We learn intra-option policies, termination conditions of options, and the policy over options in pre-training task. One option works for several timesteps until being stopped by the termination

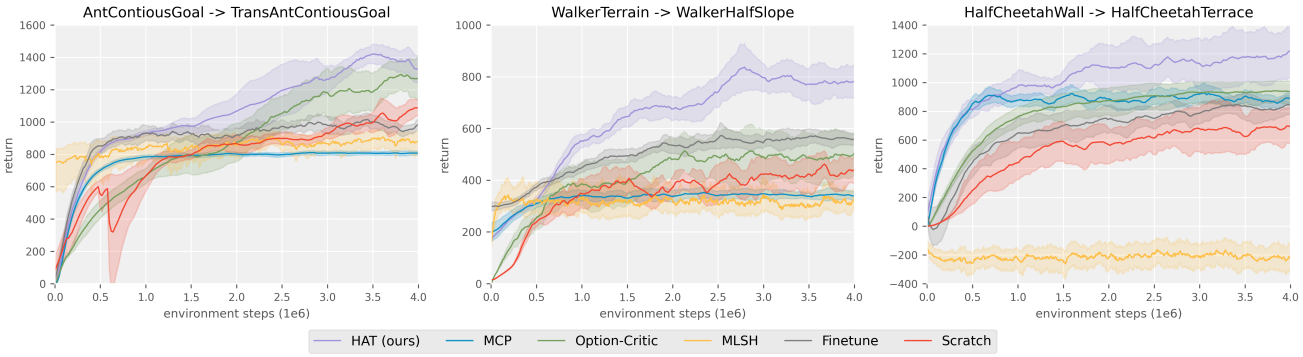

Figure 3. Performance of different transferring methods. For better visualization, we use exponential moving average to smooth the learning curve, and each learning curve is grouped with three random seeds. From these figures, we show that our method achieves better performance than other methods.

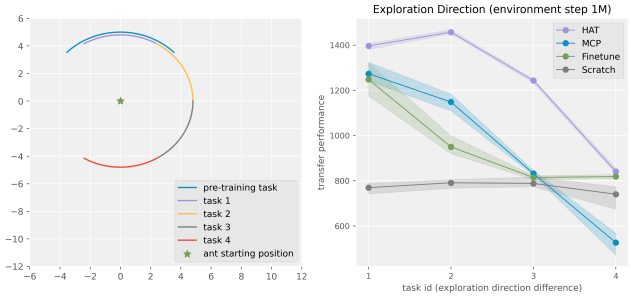

Figure 4. We show that our method achieves better transfer range than other methods. (a): the goal position sampling range for each task. From task 1 to task 4, the task difference between transferring task and pre-training task becomes larger. (b): the result of each method on different transferring tasks.

function. Then, it is directly finetuned to the transferring task. **MLSH** (Frans et al., 2018): We learn a hierarchical policy where the master policy switches between a set of sub-policies in the pre-training task. The master policy chooses a sub-policy every N timesteps, and the selected sub-policy is then executed for N timesteps. During the transferring phase, we freeze the sub-policies and only train the master policy, which makes the master policy learn how to utilize the fixed primitives.

4.3. Comparisons with Baseline

We run our method and other baselines in the three continuous problems described in the previous section. For figure 3, we find that HAT outperforms other methods in the transferring phase. MCP tends to perform well at the beginning of training, which may be caused by hierarchical abstraction. However, freezing primitive policies may restrict the transferring range of MCP, which induces that MCP cannot converge to a better result. Since HAT allows the primitives to adapt to the transferring task, it achieves significantly better reward the fastest among all the baseline algorithms. With the knowledge learned from the pretraining tasks, finetuning performs better than training from scratch. This is because training from scratch is required to learn everything, including task distribution and dynamics. However, HAT significantly outperforms finetuning by striking a better trade-off between combining the primitives to efficiently exploring the new task and gradually adapting the primitive to the new task. MLSH doesn't perform well in our transfer task since it chooses the primitives serially, which makes the primitives not decomposed well.

4.4. Transferring Range

To demonstrate the transferring range of each method, we redesign the goal position of AntContinuousGoalEnv (see figure 4 (a)). The goal position of the pre-training task is sampled from an arc where the center angle is $[\frac{-1\pi}{4}, \frac{1\pi}{4}]$ and radius 5 meters. As for transferring tasks, we design four transferring tasks, and the goal position of these four transferring tasks are sampled from $[\frac{-1\pi}{6}, \frac{1\pi}{6}]$, $[\frac{1\pi}{6}, \frac{3\pi}{6}]$, $[\frac{3\pi}{6}, \frac{5\pi}{6}]$ and $[\frac{5\pi}{6}, \frac{7\pi}{6}]$, respectively. In other words, the four transferring tasks are ordered by the scale of the difference between the pre-training task and transferring tasks.

In figure 4 (b), We find that HAT has a larger transferring range compared with other methods. Note that we report the performance at 1 million environment steps. All the other transferring methods get worse as exploration direction becomes different. As the exploration direction difference increases, MCP may perform worse than finetuning. It may be caused by fixing the primitive policies, which may limit the ability to adapt to the transferring task. The experiment result implies that HAT has a better transferring range than other methods.

References

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17, pp. 1726–1734. AAAI Press, 2017.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym, 2016.

Clavera, I., Nagabandi, A., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., and Finn, C. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In International Conference on Learning Representations, 2019.

Coumans, E. and Bai, Y. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016–2019.

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. META LEARNING SHARED HIERARCHIES. In International Conference on Learning Representations, 2018.

Mao, H., Negi, P., Narayan, A., Wang, H., Yang, J., Wang, H., Marcus, R., addanki, r., Khani Shirkoohi, M., He, S., Nathan, V., Cangialosi, F., Venkatakrishnan, S., Weng, W.-H., Han, S., Kraska, T., and Alizadeh, D. Park: An open platform for learning-augmented computer systems. In Wallach, H., Larochelle, H., Beygelzimer, A., dÁlché-Buc, F., Fox, E., and Garnett, R. (eds.), Advances in Neural Information Processing Systems 32, pp. 2494–2506. Curran Associates, Inc., 2019a.

Mao, H., Venkatakrishnan, S. B., Schwarzkopf, M., and Alizadeh, M. Variance reduction for reinforcement learning in input-driven environments. In International Conference on Learning Representations, 2019b.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. Nature, 518:529–33, 02 2015. doi: 10.1038/nature14236.

Nachum, O., Ahn, M., Ponte, H., Gu, S., and Kumar, V. Multi-agent manipulation via locomotion using hierarchical sim2real, 2019.

Peng, X. B., Chang, M., Zhang, G., Abbeel, P., and Levine, S. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In Advances in Neural Information Processing Systems 32, pp. 3686–3697. Curran Associates, Inc., 2019.

Rothfuss, J., Lee, D., Clavera, I., Asfour, T., and Abbeel, P. ProMP: Proximal meta-policy search. In International Conference on Learning Representations, 2019.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In Proceedings of the International Conference on Learning Representations (ICLR), 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. CoRR, abs/1707.06347, 2017.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of go without human knowledge. Nature, 550: 354–359, 10 2017. doi: 10.1038/nature24270.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., de Las Casas, D., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., and Riedmiller, M. DeepMind control suite. Technical report, DeepMind, January 2018.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30, 2017.

Won, J., Park, J., and Lee, J. Aerobatics control of flying creatures via self-regulated learning. ACM Trans. Graph., 37(6), December 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275023.

You, J., Liu, B., Ying, Z., Pande, V., and Leskovec, J. Graph convolutional policy network for goal-directed molecular graph generation. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), Advances in Neural Information Processing Systems 31, pp. 6410–6421. Curran Associates, Inc., 2018.