Efficient Deep Reinforcement Learning via Adaptive Policy Transfer

Tianpei Yang [1,2], **Jianye Hao** [1,2,3 *], **Zhaopeng Meng** [1], **Zongzhang Zhang** [4], **Yujing Hu** [5], **Yingfeng Chen** [5], **Changjie Fan** [5], **Weixun Wang** [1], **Wulong Liu** [2], **Zhaodong Wang** [6], **Jiajie Peng** [1]

[1]College of Intelligence and Computing, Tianjin University
[2]Noah's Ark Lab, Huawei
[3]Tianjin Key Lab of Machine Learning
[4]Nanjing University
[5]Fuxi AI Lab in Netease
[6]JD Digits
{tpyang,jianye.hao,mengzp}@tju.edu.cn, zzzhang@nju.edu.cn,
{huyujing,chenyingfeng1,fanchangjie}@corp.netease.com, wxwang@tju.edu.cn,
liuwulong@huawei.com, zhaodong.wang@jd.com, jiajiep@gmail.com

Abstract

Transfer learning has shown great potential to accelerate Reinforcement Learning (RL) by leveraging prior knowledge from past learned policies of relevant tasks. Existing approaches either transfer previous knowledge by explicitly computing similarities between tasks or select appropriate source policies to provide guided explorations. However, how to directly optimize the target policy by alternatively utilizing knowledge from appropriate source policies without explicitly measuring the similarities is currently missing. In this paper, we propose a novel Policy Transfer Framework (PTF) by taking advantage of this idea. PTF learns when and which source policy is the best to reuse for the target policy and when to terminate it by modeling multi-policy transfer as an option learning problem. PTF can be easily combined with existing DRL methods and experimental results show it significantly accelerates RL and surpasses state-of-the-art policy transfer methods in terms of learning efficiency and final performance in both discrete and continuous action spaces.

1 Introduction

Recent advance in Deep Reinforcement Learning (DRL) has obtained expressive success of achieving human-level control in complex tasks [Mnih *et al.*, 2015; Lillicrap *et al.*, 2016]. However, DRL is still faced with sample inefficiency problems especially when the state-action space becomes large, which makes it difficult to learn from scratch. Transfer learning has shown great potential to accelerate Reinforcement Learning (RL) [Sutton and Barto, 1998] via leveraging prior knowledge from past learned policies of relevant tasks [Laroche and Barlier, 2017; Rajendran *et al.*, 2017]. One major direction of transfer in RL focused on measuring the similarity between two tasks either through mapping the state spaces between two tasks [Brys *et al.*, 2015], or computing the similarity of two Markov Decision Processes (MDPs) [Song *et al.*, 2016], and then transferring value functions directly according to their similarities.

Another direction of policy transfer focuses on selecting a suitable source policy for explorations using a probabilistic exploration strategy [Fernández and Veloso, 2006] or multi-armed bandit methods [Li and Zhang, 2018]. However, such single-policy transfer cannot be applied to cases when one source policy is only partially useful for learning the target task. Although some transfer approaches utilized multiple source policies during the target task learning, they suffer from either of the following limitations, e.g., Brunskill and Li [2014] restricted source policies to be learned in a Probably Approximately Correct (PAC)-learning way; Mann et al. [2014] required a model of the environment to learn which is infeasible for large-scale problems; Laroche and Barlier [2017] assumed that all tasks share the same transition dynamics and differ only in the reward function. Recently, Li et al. [2019] relaxed the above drawbacks by proposing Context-Aware Policy reuSe (CAPS) for multi-policy transfer by involving the option framework to select source policies for explorations. However, CAPS is prone to the imperfection of source policies since it learns an intra-option policy over these source policies. Furthermore, it requires manually adding primitive policies to the policy library which limits its generality and cannot be applied to problems with continuous action spaces.

To address the above problems, we propose a novel Policy Transfer Framework (PTF) which combines the above two directions of policy reuse. Instead of using source policies as guided explorations in a target task, we adaptively select a suitable source policy during target task learning and use it as a complementary optimization objective of the target policy. The backbone of PTF can still use existing DRL algorithms to update its policy, and the source policy selection

*Corresponding author.

problem is modeled as the option learning problem. In this way, PTF does not require any source policy to be perfect on any subtask and can still learn toward an optimal policy in case none of the source policy is useful. Besides, the option framework allows us to use the termination probability as a performance indicator to determine whether a source policy reuse should be terminated to avoid negative transfer. In summary, the main contributions of our work are: 1) PTF learns when and which source policy is the best to reuse for the target policy and when to terminate it by modeling multi-policy transfer as the option learning problem; 2) we propose an adaptive and heuristic mechanism to ensure the efficient reuse of source policies and avoid negative transfer; and 3) both existing value-based and policy-based DRL approaches can be incorporated and experimental results show PTF significantly boosts the performance of existing DRL approaches, and outperforms state-of-the-art policy transfer methods both in discrete and continuous action spaces.

2 Background

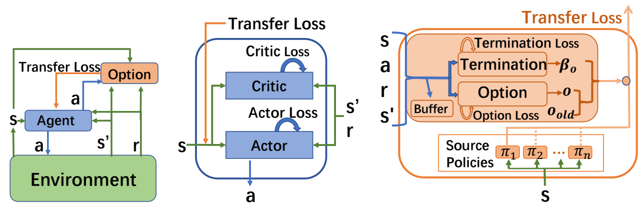
Formally, an RL task can be specified by a Markov Decision Process (MDP), which is a tuple $< S, A, T, R >$, where S is the state set; A is the action set; T is the state transition function: $S \times A \times S \rightarrow [0, 1]$ and R is the reward function: $S \times A \times S \rightarrow \mathbb{R}$. A policy π is a probability distribution over actions conditioned on states: $S \times A \rightarrow [0, 1]$. The goal is to find an optimal policy π^* maximizing the total expected return with a discount factor γ: $U = \sum_{i=t}^{T} \gamma^{i-t} r_i$.

Policy Gradient (PG) algorithms are another choice for dealing with RL tasks, which is to directly optimize the policy π parameterized by θ. PG methods optimize the objective $J(\theta) = \mathbb{E}_{s \sim P^\pi, a \sim \pi_\theta}[U]$ by taking steps in the direction of $\nabla_\theta J(\theta)$. Using Q-function, then the gradient of the policy can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim P^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)], \qquad (1)$$

where P^π is the state distribution given π. Several practical PG algorithms differ in how they estimate Q^π. Several practical PG algorithms differ in how they estimate Q^π. One major category is to learn an approximation of the action-value function $Q^\pi(s, a)$; $Q^\pi(s, a)$ is called the critic and leads to a variety of actor-critic algorithms [Mnih *et al.*, 2016; Schulman *et al.*, 2017].

The option framework [Sutton *et al.*, 1999] formalized the idea of temporally extended actions as an option. An option $o \in \mathcal{O}$ is defined as a triple $\{\mathcal{I}_o, \pi_o, \beta_o\}$ in which $\mathcal{I}_o \in \mathcal{S}$ is an initiation state set, π_o is an intra-option policy and $\beta_o : \mathcal{I}_o \rightarrow [0, 1]$ is a termination function that specifies the probability an option o terminates at state $s \in \mathcal{I}_o$. An MDP endowed with a set of options becomes a Semi-Markov Decision Process (Semi-MDP), which has a corresponding optimal option-value function over options learned using intra-option learning. The option framework considers the *call-and-return* option execution model, in which an agent picks option o according to its option-value function $Q(s, o)$, and follows the intra-option policy π_o until termination, then selects a next option and repeats the procedure.



(a) PTF (b) Agent Module (c) Option Module

Figure 1: An illustration of the policy transfer framework.

3 Policy Transfer Framework (PTF)

3.1 Motivation

One major direction of previous works focuses on transferring value functions directly according to the similarity between two tasks [Brys *et al.*, 2015; Song *et al.*, 2016; Laroche and Barlier, 2017]. However, this way often assumes a well-estimated model for measurement which causes computational complexity and is infeasible in complex scenarios. Another direction of policy transfer methods focuses on selecting appropriate source policies based on the performance of source policies on the target task to provide guided explorations during each episode [Fernández and Veloso, 2006; Li and Zhang, 2018; Li *et al.*, 2019]. However, most of these works are faced with the challenge of how to select a suitable source policy, since each source policy may only be partially useful for the target task. Furthermore, some of them assume source policies to be optimal and deterministic which restricts the generality. How to directly optimize the target policy by alternatively utilizing knowledge from appropriate source policies without explicitly measuring the similarities is currently missing in previous work.

According to the above analysis, in this paper, we firstly propose a novel Policy Transfer Framework (PTF) to accelerate RL by taking advantage of this idea and combining the above two directions of policy reuse. Instead of using source policies as guided explorations in a target task, PTF adaptively selects a suitable source policy during target task learning and uses it as a complementary optimization objective of the target policy. In this way, PTF does not require any source policy to be perfect on any subtask and can still learn toward an optimal policy in case none of the source policy is useful. Besides, we propose a novel way of adaptively determining the degree of transferring the knowledge of a source policy to the target one to avoid negative transfer, which can be effectively used in cases when only part of source policies share the same state-action space as the target one.

3.2 Framework Overview

Figure 1(a) illustrates the proposed Policy Transfer Framework (PTF) which contains two main components, one (Figure 1(b)) is the agent module (here is an example of an actor-critic model), which is used to learn the target policy with guidance from the option module. The other (Figure 1(c)) is the option module, which is used to learn when and which source policy is useful for the agent module. Given a

Algorithm 1 PTF-A3C

```
 1: Initialize: option-value network parameters $\theta_o$, termination network parameters $\theta_\beta$, replay buffer $\mathcal{D}$, global parameters $\theta$ and $\theta_v$, thread-specific parameters $\theta'$ and $\theta'_v$, step t ← 1
 2: for each thread do
 3:     Reset gradients: $d\theta \leftarrow 0, d\theta_v \leftarrow 0$
 4:     Assign thread-specific parameters: $\theta' = \theta, \theta'_v = \theta_v$
 5:     Start from state $s$, $t_{start} = t$
 6:     Select an option $o \leftarrow \epsilon$-greedy$(Q_o(s, o|\theta_o))$
 7:     repeat
 8:         Perform an action $a \sim \pi(s|\theta')$
 9:         Observe reward $r$ and new state $s'$
10:         $t \leftarrow t + 1$
11:         Store transition $(s, a, r, s')$ to replay buffer $\mathcal{D}$
12:         Choose another option if $o$ terminates
13:     until $s$ is terminal or $t - t_{start} == t_{max}$
14:     $R = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ V(s, \theta'_v) & \text{otherwise} \end{cases}$
15:     for $i \in \{t-1, \cdots, t_{start}\}$ do
16:         $R \leftarrow r_i + \gamma R$
17:         Calculate gradients w.r.t. $\theta'_v$:
            $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i|\theta'_v))^2/\partial\theta'_v$
18:         Calculate gradients w.r.t. $\theta'$:
            $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i, \theta')(R - V(s_i|\theta'_v)) + \rho\nabla_{\theta'}\mathrm{H}(\pi(s_i|\theta')) + f(\beta_o, t)\mathrm{L_H}$
19:         Update$(Q_o(s, o|\theta_o))$ (see Algorithm 2)
20:         Update $\beta(s, o|\theta_\beta)$ w.r.t. $\theta_\beta$ (Equation 4)
21:     end for
22:     Asynchronously update $\theta$ using $d\theta$ and $\theta_v$ using $d\theta_v$
23: end for
```

set of source policies $\Pi_s = \{\pi_1, \pi_2, \cdots, \pi_n\}$ as the intra-option policies, the PTF agent first initializes a set of options $\mathcal{O} = \{o_1, o_2, \cdots, o_n\}$ together with the option-value network with random parameters. At each step, it selects an action following its policy, receives a reward and transitions to the next state. Meanwhile, it also selects an option o_i according to the policy over options and the termination probabilities. For the update, the PTF agent introduces a complementary loss, which transfers knowledge from the intra-option policy π_i through imitation, weighted by an adaptive adjustment factor $f(\beta_o, t)$. The PTF agent will also update the option-value network and the termination probability of o_i using its own experience simultaneously. The reuse of the policy π_i terminates according to the termination probability of o_i and then another option is selected for reuse following the policy over options. In this way, PTF efficiently exploits the useful information from the source policies and avoids negative transfer through the call-and-return option execution model. PTF could be easily integrated with both value-based and policy-based DRL methods.

We describe PTF applying in A3C [Mnih *et al.*, 2016]: PTF-A3C as an example in detail. The whole learning process of PTF-A3C is shown in Algorithm 1. First, PTF-A3C initializes network parameters for the option-value network, the termination network (which shares the input and hidden layers with the option-value network and holds a different output layer), and A3C networks (Line 1). For each episode, the PTF-A3C agent first selects an option o according to the policy over options (Line 6); then it selects an action following the current policy $\pi(s|\theta')$, receives a reward r, transits to the next state s' and stores the transition to the replay buffer $\mathcal{D}$ (Lines 8-11). Another option will be selected if the option o is terminated according to the termination probability of o (Line 12).

Algorithm 2 Update$(Q_o(s, o|\theta_o))$

```
1: Sample a batch of $N$ transitions $(s, a, r, s')$ from $\mathcal{D}$
2: for $o \in O$ do
3:     if $\pi_o$ selects action $a$ at state $s$ then
4:         Update $U(s', o|\theta_o)$ (Equation 2)
5:         Set $y \leftarrow r + \gamma U(s', o|\theta_o)$
6:         Update option by minimizing the loss:
           $L \leftarrow \frac{1}{N}\sum_i (y_i - Q_o(s_i, o|\theta_o))^2$
7:     end if
8: end for
9: Copy $\theta_o$ to the target network $Q'_o$ every $\tau$ steps
```

For the update, the agent computes the gradient of the temporal difference loss for the critic network (Line 17); and calculates the gradients of the standard actor loss, and also the extra loss of difference between the source policy π_o inside the option o and the current policy $\pi(\theta')$, which is measured by the cross-entropy loss: $\mathrm{L_H} = \mathrm{H}(\pi_o \parallel \pi(\theta'))$. $\mathrm{L_H}$ is used as the supervision signal, weighted by an adaptive adjustment factor $f(\beta_o, t)$. To ensure sufficient explorations, an entropy bonus is also considered [Mnih *et al.*, 2016], weighted by a constant factor ρ (Line 18). Then it updates the option-value network following Algorithm 2 and the termination network accordingly (Lines 19, 20) which is described in detail in the following section.

3.3 Update the Option Module

The remaining issue is how to update the option-value network which is given in Algorithm 2. Since options are temporal abstractions [Sutton *et al.*, 1999; Bacon *et al.*, 2017], U is introduced as the option-value function *upon arrival*. The expected return of executing option o upon entering next state s' is $U(s', o|\theta_o)$, which is correlated to $\beta(s', o|\theta_\beta)$, i.e., the probability that option o terminates in next state s':

$$U(s', o|\theta_o) \leftarrow (1 - \beta(s', o|\theta_\beta))Q'_o(s', o|\theta'_o) + \beta(s', o|\theta_\beta)\max_{o' \in O} Q'_o(s', o'|\theta'_o). \tag{2}$$

Then, PTF-A3C samples a batch of N transitions from the replay buffer $\mathcal{D}$ and updates the option-value network by minimizing the loss (Line 6 in Algorithm 2). Each sample can be used to update the values of multiple options, as long as the option allows to select the sampled action (for continuous action space, this is achieved by fitting action a in the source policy distribution with a certain confidence interval). Thus the sample efficiency can be significantly improved in an off-policy manner.

PTF-A3C learns option-values in the call-and-return option execution model, where an option o is executed until it terminates at state s based on its termination probability $\beta(s, o|\theta_\beta)$. The policy of selecting an option is ϵ-greedy to the option-value Q_o. Specifically, with a probability of $1-\epsilon$, the option with the highest option-value is selected (random selection in case of a tie); and random choices are made with probability ϵ to explore other options with potentially better performance.

According to the call-and-return option execution model, the termination probability controls when to terminate the current selected option and select another option accordingly. The objective of learning the termination probability is to maximize the expected return U, so we update the termination network parameters by computing the gradient of the discounted return objective with respect to the initial condition (s_1, o_1) [Bacon *et al.*, 2017]:

$$\frac{\partial U(s_1, o_1|\theta_o)}{\partial \theta_\beta} = -\sum_{s',o} \mu(s', o|s_1, o_1) \frac{\partial \beta(s', o|\theta_\beta)}{\partial \theta_\beta} A(s', o|\theta_o), \tag{3}$$

where $A(s', o|\theta_o)$ is the advantage function which can be approximated as $Q_o(s', o|\theta_o) - \max_{o' \in O} Q_o(s', o'|\theta_o)$, and $\mu(s', o|s_1, o_1)$ is a discounted factor of state-option pairs from the initial condition (s_1, o_1): $\mu(s', o|s_1, o_1) = \sum_{t=0}^{\infty} \gamma^t P(s_t = s', o_t = o|s_1, o_1)$. $P(s_t = s', o_t = o|s_1, o_1)$ is the transition probability along the trajectory starting from the initial condition (s_1, o_1) to (s', o) in t steps. Since $\mu(s', o|s_1, o_1)$ is estimated from samples along the on-policy stationary distribution, we neglect it for data efficiency [Thomas, 2014; Li *et al.*, 2019]. Then $\beta(s, o|\theta_\beta)$ is updated w.r.t. θ_β as follows [Bacon *et al.*, 2017; Li *et al.*, 2019]:

$$\theta_\beta \leftarrow \theta_\beta - \alpha_\beta \frac{\partial \beta(s', o|\theta_\beta)}{\partial \theta_\beta} \left(A(s', o|\theta_o) + \xi\right), \tag{4}$$

where α_β is the learning rate, ξ is a regularization term. The advantage term is 0 if the option is the one with the maximized option value, and negative otherwise. In this way, all termination probabilities would increase if the option value is not the maximized one. However, the estimation of the option-value function is not accurate initially. If we multiply the advantage to the gradient, the termination probability of an option with the maximize true option value would also increase, which would lead to a sub-optimal policy over options. The purpose of ξ is to ensure sufficient exploration that the best one could be selected.

3.4 Transfer from Selected Source Policy

Next, we describe how to transfer knowledge from the selected source policy. The way to transfer is motivated from policy distillation [Rusu *et al.*, 2016] which exploits multiple teacher policies to train a student policy. Namely, a teacher policy π_t is used to generate trajectories x, each containing a sequence of states $(x_t)_{t \geq 0}$. The goal is to match student's policy $\pi_s(\theta)$, parameterized by θ, to π_t. The corresponding loss function term for each sequence at each time step t is: $\mathrm{H}(\pi_t(a|x_t) \parallel \pi_s(a|x_t, \theta))$, where $\mathrm{H}(\cdot \parallel \cdot)$ is the cross-entropy loss. For value-based algorithms, e.g., DQN, we can measure the difference of two Q-value distributions using the Kullback-Leibler (KL) divergence.

Kickstarting [Schmitt *et al.*, 2018] trains a student policy that surpasses the teacher policy on the same task set by adding the cross-entropy loss between the teacher and student policies to the RL loss. However, it does not consider learning a new task that is different from the teacher's task set. Furthermore, the way using Population Based Training (PBT) [Jaderberg *et al.*, 2017] to adjust the weighting factor of the cross-entropy loss increases the computational complexity, lack of adaptive adjustment.

To this end, we propose an adaptive and heuristic way to adjust the weighting factor $f(\beta_o)$ of the cross-entropy loss. The option module contains a termination network that reflects the performance of options on the target task. If the performance of the current option is not the best among all options, the termination probability of this option grows, which indicates we should assign a higher probability to terminate the current option. Therefore, the termination probability of a source policy can be used as a performance indicator of adjusting its exploitation degree. Specifically, the probability of exploiting the current source policy π_o should be decreased as the performance of the option o decreases. And the weighting factor $f(\beta_o, t)$ which implies the probability of exploiting the current source policy π_o should be inversely proportional to the termination probability. Specifically, we propose adaptively adjust $f(\beta_o, t)$ as follows:

$$f(\beta_o, t) = f(t)(1 - \beta(s_t, o|\theta_\beta)), \tag{5}$$

where $f(t)$ is a discount function. When the value of the termination function of option o increases, it means that the performance of the option o is not the best one among all options based on the current experience. Thus we decrease the weighting factor $f(\beta_o, t)$ of the cross-entropy loss $\mathrm{H}(\pi_o \parallel \pi(\theta))$ and vice versa. $f(t)$ controls the slow decrease in exploiting the transferred knowledge from source policies which means at the beginning of learning, we exploit source knowledge mostly. As learning continues, past knowledge becomes less useful and we focus more on the current self-learned policy. In this way, PTF efficiently exploits useful information and avoids negative transfer from source policies.

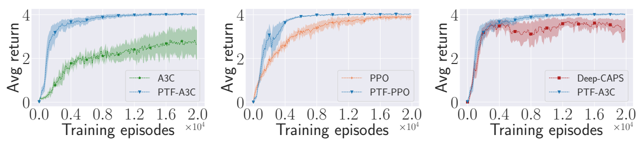
4 Experimental Results

In this section, we evaluate PTF on three domains, grid world [Li *et al.*, 2019], pinball [Bacon *et al.*, 2017] and reacher [Tassa *et al.*, 2018] compared with several DRL methods learning from scratch (A3C [Mnih *et al.*, 2016] and PPO [Schulman *et al.*, 2017]); and the state-of-the-art policy transfer method CAPS [Li *et al.*, 2019], implemented as a deep version (Deep-CAPS). Results are averaged over 20 random seeds. We set $\xi = 0.005$, $f(t) = 0.5 + \tanh(3 - 0.001 * t)/2$[1].

4.1 Grid World

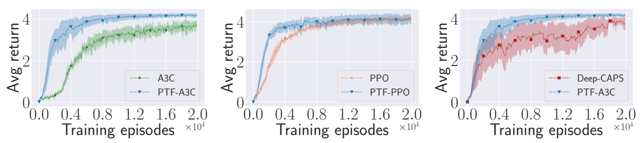
In a grid world W (see Figure 1 in [Li *et al.*, 2019]), with an agent starting from any of the grids, and choosing one of four actions: up, down, left and right. Each action makes the agent move to the corresponding direction with one step size. $G1, G2, G3$ and $G4$ denote goals of source tasks, g and g' represent goals of target tasks. As noted, g is similar to one of the source tasks $G1$ since their goals are within a close

[1]The source code and supplementary materials are put on https://github.com/PTF-transfer/Code_PTF.



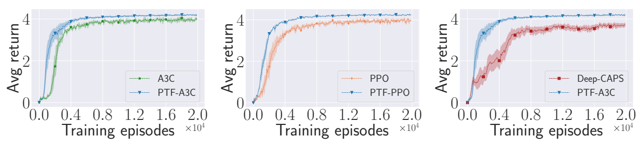
(a) A3C (b) PPO (c) Deep-CAPS

Figure 2: Average return when learning task g on grid world W.



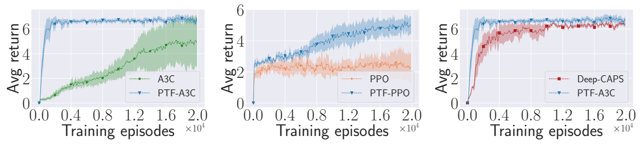
(a) A3C (b) PPO (c) Deep-CAPS

Figure 4: Average return when learning task g' on grid world W'.



(a) A3C (b) PPO (c) Deep-CAPS

Figure 3: Average return when learning task g' on grid world W.



(a) A3C (b) PPO (c) Deep-CAPS

Figure 5: Average return when learning g on pinball.

distance; while g' is different from each source task due to the far distance among their goals. The game ends when the agent approaches the target grid or the time exceeds a fixed period. The agent receives a reward of $+5$ after approaching the goal grid. The source policies are trained using A3C learning from scratch. We also manually design 4 primitive policies for deep-CAPS following its previous settings (i.e., each primitive policy selects the same action for all states), which is unnecessary for our PTF.

We first investigate the performance of PTF when the target task g is similar to one of the source tasks, $G1$ (i.e., the distance between their goal grids is very close). Figure 2 presents the average return of various methods when learning task g on grid world W. We can see from Figure 2(a) that PTF-A3C significantly accelerates the learning process and outperforms A3C. Similar results can be found in Figure 2(b). The reason is that PTF quickly identifies the optimal source policy and exploits useful information from source policies, which efficiently accelerates the learning process than learning from scratch. Figure 2(c) shows the performance gap between PTF-A3C and deep-CAPS. This is because the policy reuse module and the target task learning module in PTF are loosely decoupled, apart from reusing knowledge from source policies, PTF is also able to utilize its own experience from the environment. However, in deep-CAPS, these two parts are highly decoupled, which means its explorations and exploitations are fully dependent on the source policies inside the options. Thus, deep-CAPS needs higher requirements on source policies than our PTF, and finally achieves lower performance than PTF-A3C.

Next, we investigate the performance of PTF when all source tasks are not quite similar to the target task (i.e., the distance between their goal grids is very far). Figure 3 presents average return of various methods when learning task g'. We can see from Figure 3(a), (b) that both PTF-A3C and PTF-PPO significantly accelerate the learning process and outperform A3C and PPO. The reason is that PTF identifies which source policy is optimal to exploit and when to terminate it, which efficiently accelerates the learning process than learning from scratch. The lower performance of deep-CAPS than PTF-A3C (Figure 3(c)) is due to similar reasons as described before, that its explorations and exploitations are fully dependent on source policies, thus needs higher requirements on source policies than PTF, and finally achieves lower performance than PTF-A3C.

To verify that PTF works as well in situations where transitions between source and target tasks are different, we conduct experiments on learning on a grid world W' (see Figure 6 in [Li *et al.*, 2019]), whose map is much different from the map for learning source tasks. Figure 4 shows that PTF still outperforms other methods even if only some parts of source policies can be exploited. PTF identifies and exploits useful parts automatically.

4.2 Pinball

In pinball (see Figure 4 in [Bacon *et al.*, 2017]), a ball is guided through a maze of arbitrarily shaped polygons to a designated target location. The state space is continuous over the position and velocity of the ball in $x-y$ plane. The action space is continuous in $[-1, 1]$ range, which controls the increment of the velocity in the vertical or horizontal direction. A drag coefficient of 0.995 effectively stops ball movements after a finite number of steps when the null action is chosen repeatedly. Each thrust action incurs a penalty of -5 while taking no action costs -1. The episode terminates with a $+10000$ reward when the agent reaches the target. We interrupt any episode taking more than 500 steps and set the discount factor to 0.99. These rewards are all normalized to ensure more stable training. The source policies are trained using A3C learning from scratch. We also design 5 primitive policies for deep-CAPS, which is unnecessary for our PTF.

Figure 5 depicts the performance of PTF when learning task g on Pinball, which is similar to source task $G1$ (i.e., the distance between their goal states is very close). We can see that PTF significantly accelerates the learning process of A3C and PPO (Figure 5(a) and (b)); outperforms deep-CAPS (Figure 5(c)). The advantage of PTF is similar with that in grid world: PTF efficiently exploits the useful in-

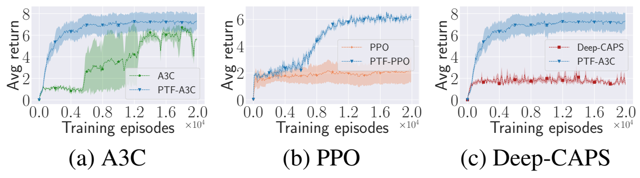


Figure 6: Average return when learning g' on pinball.

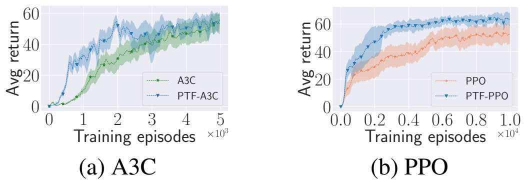


Figure 7: The performance of PTF on Reacher.

formation from source policies to optimize the target policy, thus achieves higher performance than learning from scratch. Deep-CAPS achieves lower average return than PTF but outperforms vanilla A3C and PPO. This indicates it still exploits useful information from source policies when the target task is very similar with one of source tasks. However, it fully depends on source policies for explorations, and a continuous action space is hard to be fully covered even with the manually added primitive policies. Therefore, deep-CAPS achieves lower performance than PTF in such a domain.

We further verify whether PTF works well in the same setting as in the grid world that all source tasks are not quite similar to the target task g' (i.e., the distance between their goal states is very far). From Figure 6 we can see that PTF outperforms other methods even if only some parts of source policies can be exploited. This is because PTF identifies when and which source policy is optimal to exploit and when to terminate it, thus efficiently accelerates the learning process. However, due to the drawbacks described above, deep-CAPS fails when the target task is quite dissimilar with source tasks.

4.3 Reacher

To further validate the performance of PTF, we provide an alternative scenario, Reacher [Tassa *et al.*, 2018], which is qualitatively different from the above two navigation tasks. Reacher is one of robot control problems in MuJoCo [Todorov *et al.*, 2012], equipped with a two-link planar to reach a target location. The episode ends with the $+1$ reward when the end effector penetrates the target sphere, or ends when it takes more than 1000 steps. We design several tasks in Reacher which are different from the location and size of the target sphere. Since deep-CAPS performs poorly in the above continuous domain (pinball), we only compare PTF with vanilla A3C and PPO in the following sections.

Figure 7(a) shows that PTF-A3C efficiently achieves higher average return than A3C. Similar results can be found in PTF-PPO and PPO shown in Figure 7(b). This is because PTF efficiently exploits the useful knowledge in source tasks, thus accelerates the learning process compared with vanilla

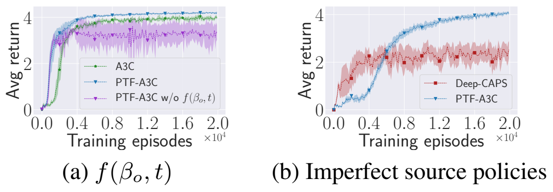


Figure 8: Different impacts on the final performance.

methods. These results further show the robustness of PTF.

4.4 Ablation Study

Next, we provide an ablation study to investigate the influence of the weighting factor $f(\beta_o,t)$ (Equation 5) on the performance of PTF, which is the key factor. Figure 8 shows the influence of different parts of the weighting factor on the performance of PTF-A3C. We can see from Figure 8 (a) when the extra loss is added without the weighting factor $f(\beta_o,t)$, although it helps the agent at the beginning of learning compared with A3C learning from scratch, it leads to a sub-optimal policy because of focusing too much on mimicking the source policies. In contrast, introducing the weighting factor $f(\beta_o,t)$ allows us to terminate exploiting source policies in time and thus achieves the best transfer performance.

We further investigate whether PTF can efficiently avoid negative transfer. Figure 8 (b) shows the average return of PTF-A3C and deep-CAPS when source policies are not optimal towards source tasks. As we described before, deep-CAPS is fully dependent on source policies for explorations and exploitations on the target task, when source policies are not optimal towards source tasks, which means they are not deterministic at all states. Thus, deep-CAPS cannot avoid the negative and stochastic impact of source policies, which confuses the learning of the option-value network and finally obtains lower performance than PTF-A3C.

5 Conclusion and Future Work

In this paper, we propose a Policy Transfer Framework (PTF) which can efficiently select the optimal source policy and exploit the useful information to facilitate the target task learning. PTF also efficiently avoids negative transfer through terminating the exploitation of current source policy and selects another one adaptively. PTF can be easily combined with existing deep DRL methods. Experimental results show PTF efficiently accelerates the learning process of existing state-of-the-art DRL methods and outperforms previous policy reuse approaches. As a future topic, it is worthwhile investigating how to extend PTF to multiagent settings, and how to learn abstract knowledge for fast adaptation in new environments.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (Grant Nos.: 61702362, U1836214, 61876119), the new Generation of Artificial Intelligence Science and Technology Major Project of Tianjin under grant: 19ZXZNGX00010, and the Natural Science Foundation of Jiangsu under Grant No. BK20181432.

References

[Bacon *et al.*, 2017] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 1726–1734, 2017.

[Brunskill and Li, 2014] Emma Brunskill and Lihong Li. PAC-inspired option discovery in lifelong reinforcement learning. In *Proceedings of International Conference on Machine Learning*, pages 316–324, 2014.

[Brys *et al.*, 2015] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. Policy transfer using reward shaping. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 181–188, 2015.

[Fernández and Veloso, 2006] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

[Jaderberg *et al.*, 2017] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017.

[Laroche and Barlier, 2017] Romain Laroche and Merwan Barlier. Transfer reinforcement learning with shared dynamics. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 2147–2153, 2017.

[Li and Zhang, 2018] Siyuan Li and Chongjie Zhang. An optimal online method of selecting source policies for reinforcement learning. In *Proceedings of AAAI Conference on Artificial Intelligence*, pages 3562–3570, 2018.

[Li *et al.*, 2019] Siyuan Li, Fangda Gu, Guangxiang Zhu, and Chongjie Zhang. Context-aware policy reuse. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems*, pages 989–997, 2019.

[Lillicrap *et al.*, 2016] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of International Conference on Learning Representations*, 2016.

[Mann *et al.*, 2014] Timothy Mann, Daniel Mankowitz, and Shie Mannor. Time-regularized interrupting options. In *Proceedings of International Conference on Machine Learning*, pages 1350–1358, 2014.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Mnih *et al.*, 2016] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of International Conference on Machine Learning*, pages 1928–1937, 2016.

[Rajendran *et al.*, 2017] Janarthanan Rajendran, Aravind S Lakshminarayanan, Mitesh M Khapra, P Prasanna, and Balaraman Ravindran. Attend, adapt and transfer: Attentive deep architecture for adaptive transfer from multiple sources in the same domain. In *Proceedings of International Conference on Learning Representations*, 2017.

[Rusu *et al.*, 2016] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *Proceedings of International Conference on Learning Representations*, 2016.

[Schmitt *et al.*, 2018] Simon Schmitt, Jonathan J. Hudson, Augustin Zídek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Küttler, Andrew Zisserman, Karen Simonyan, and S. M. Ali Eslami. Kickstarting deep reinforcement learning. *arXiv preprint arXiv:1803.03835*, 2018.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Song *et al.*, 2016] Jinhua Song, Yang Gao, Hao Wang, and Bo An. Measuring the distance between finite Markov decision processes. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems*, pages 468–476, 2016.

[Sutton and Barto, 1998] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.

[Sutton *et al.*, 1999] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211, 1999.

[Tassa *et al.*, 2018] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. DeepMind Control Suite. *CoRR*, abs/1801.00690, 2018.

[Thomas, 2014] Philip Thomas. Bias in natural actor-critic algorithms. In *Proceedings of International Conference on Machine Learning*, pages 441–448, 2014.

[Todorov *et al.*, 2012] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.