Context-Aware Policy Reuse

Siyuan Li
Tsinghua University
sy-li17@mails.tsinghua.edu.cn

Fangda Gu
Tsinghua University
gfd15@mails.tsinghua.edu.cn

Guangxiang Zhu
Tsinghua University
guangxiangzhu@outlook.com

Chongjie Zhang
Tsinghua University
chongjie@tsinghua.edu.cn

Abstract

Transfer learning can greatly speed up reinforcement learning for a new task by leveraging policies of relevant tasks. Existing works of policy reuse either focus on only selecting a single best source policy for transfer without considering contexts, or cannot guarantee to learn an optimal policy for a target task. To improve transfer efficiency and guarantee optimality, we develop a novel policy reuse method, called *Context-Aware Policy reuSe* (CAPS), that enables multi-policy transfer. Our method learns when and which source policy is best for reuse, as well as when to terminate its reuse. CAPS provides theoretical guarantees in convergence and optimality for both source policy selection and target task learning. Empirical results on a grid-based navigation domain and the Pygame Learning Environment demonstrate that CAPS significantly outperforms other state-of-the-art policy reuse methods.

1 Introduction

Reinforcement learning (RL) [1] has recently shown considerable successes of achieving human-level control in challenging tasks [2, 3]. However, learning each task independently and from scratch requires vast experiences, and thus is inefficient for practical problems. Transfer learning has been actively studied for accelerating RL by making use of prior knowledge [4]. Extensive transfer learning research aims to reuse source policies to speed up the learning on a new target task [5, 6, 7, 8].

Many existing policy reuse approaches focus on finding a single best source policy for transfer, e.g., by measuring MDP similarity [9, 10], through online exploration using multi-armed bandit methods [11, 12], or via an optimism-under-uncertainty approach [13]. However, such single-policy transfer is not efficient enough, because it is more often that multiple source policies are partly useful for learning a new task. Although some multi-policy transfer methods have been proposed to concurrently utilize multiple source policies in the target task learning, those methods suffer from limitations, e.g., restricting the way of obtaining source policies [14], converging to locally optimal termination functions [15], or requiring a model of learning environment [16].

In this paper, we propose a novel model-free multi-policy reuse method, called *Context-Aware Policy reuSe* (CAPS). Our approach learns an optimal source selection policy, which specifies the most appropriate source policy to reuse based on contexts (i.e., a subset of states). In addition, CAPS provides a convergence guarantee to an optimal target policy, agnostic to the usefulness of the source policies and how to acquire them (which can be either learned from prior experience or provided by advisors). To improve transfer efficiency and support temporally-extended policy reuse, CAPS utilizes a *call-and-return* execution model and concurrently learns when to reuse which source policy, as well as when to terminate its reuse. CAPS assumes that the state-action space is shared between source and target tasks, which can be relaxed if the states and actions in the source tasks could be

Preprint. Work in progress.

mapped to a target task. Empirical results in a grid-based navigation domain as well as the Pygame Learning Environment (PLE) [17] show that CAPS (i) learns the optimal source selection policy with temporally-extended policy reuse and speeds up the target task learning significantly even when its transition function is different from those of source tasks; (ii) outperforms state-of-the-art transfer algorithms remarkably when multiple source policies are useful; and (iii) achieves the same performance as, if not better than, single-policy reuse methods in situations where only one source policy is useful.

2 Preliminaries and Problem Statement

This paper focuses on RL tasks, whose environments can be modeled by Markov Decision Processes (MDP). An MDP consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition function P, and a reward function $\mathcal{R}$. At each time step, an agent chooses and executes an action a on the current state s, and then receives a reward $R(s,a)$ and observes the next state s' according to transition function $P(s'|s,a)$. A policy $\pi : \mathcal{S} \to \mathcal{A}$ specifies an action for each state and its state-action value function $Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s, a_0 = a]$ is the expected return for executing action a on state s and following policy π afterwards. $\gamma \in [0,1)$ is a discount factor. A greedy policy π with respect to a value function Q is given by $\pi(s) = \text{argmax}_{a \in \mathcal{A}} Q(s,a)$ for all state $s \in \mathcal{S}$. The goal of an RL algorithm on an MDP is to find an optimal policy that maximizes the expected return. Q-learning learns the optimal Q function, which yields an optimal policy [18].

A Policy Reuse Problem is defined as following: given a set of source policies $\Pi_s = \{\pi_1, \pi_2, ..., \pi_n\}$ and a new target task g, the goal is to quickly learn an optimal policy for the target task by exploiting knowledge from source policies. Source policies can be either learned for different but relevant tasks or heuristically designed by humans. In this paper, we assume that each source policy and the target task have the same state-action space. This assumption can be relaxed if the states and actions in the source tasks could be mapped to a target task. The policy reuse problem has two related objectives: finding the optimal policy π_g^* for the target task and learning the optimal source selection policy $\pi_{\Pi_s}^* : \mathcal{S} \to \Pi_s$ for reusing source policies during the learning. The optimal source selection policy $\pi_{\Pi_s}^*$ should be consistent with π_g^*, that is, if source policy $\pi_i \in \Pi_s$ is selected by $\pi_{\Pi_s}^*$ for state s, then it should select the same action on state s as the optimal target policy, i.e., $\pi_i(s) = \pi_g^*(s)$.

3 Approach

We aim to enable an agent to quickly learn an optimal policy for a new target task by leveraging knowledge from multiple source policies. Selecting a single policy to reuse is not efficient when provided with a source policy library where multiple policies are partly useful. Therefore, it is essential to identify both when (i.e., on which states) and which source policy is the most appropriate to reuse.

In this paper, we develop a novel policy reuse method, called *Context-Aware Policy reuSe* (CAPS), for multi-policy transfer learning. By exploiting the option framework [19], CAPS formulates source policy selection as an inter-option learning problem, whose solution is called a *source selection policy* specifying the choice of a source policy to reuse for each state. In the formulation, the source policy library is expanded with primitive policies to ensure the optimality of the learned target policy no matter whether the usefulness of source policies is sufficient or not. To improve transfer efficiency and support temporally-extended policy reuse, CAPS uses the *call-and-return* model for reusing selected policies, where the execution of a selected policy is returned until completion according to its termination function. CAPS simultaneously learns the source selection policy and the termination function for each source policy and primitive policy. Theoretical guarantees in convergence and optimality are provided for CAPS in learning both the source selection policy and the target task policy.

In the rest of this section, we will first describe our formulation of multi-policy reuse as inter-option learning, then present the CAPS learning algorithm, and finally analyze the theoretical guarantees.

3.1 Formulation as Inter-Option Learning

We formulate multi-policy reuse as an inter-option learning problem. Options are temporally-abstracted policies for taking actions over a period of time [19]. An option $o \in \mathcal{O}$ is defined by a triple $(\pi_o, \mathcal{I}, \beta_o)$, where π_o is an intra-option policy, $\mathcal{I} \subseteq \mathcal{S}$ is an initiation state set, and $\beta_o : \mathcal{S} \rightarrow [0, 1]$ is a termination function that specifies the probability of option o terminating on each state $s \in \mathcal{S}$. Any MDP endowed with a set of options becomes a semi-MDP, which has a corresponding optimal option-value function $Q_{\mathcal{O}}(s, o)$ over options.

In our formulation, we create a set of source options $\mathcal{O}_s$ from the given source policy library Π_s. For each source policy $\pi \in \Pi_s$, we instantiate an option $o = (\pi_o, \mathcal{I}, \beta_{\theta_o})$, where its intra-option policy $\pi_o = \pi$, its initiation set $\mathcal{I}$ is the whole state space, and its termination function β_{θ_o} is defined by a sigmoid function with a differentiable parameter θ_o: $\beta_{\theta_o}(s) = \frac{1}{1+e^{-\theta_o(s)}}$.

With this formulation, an inter-option policy corresponds to a source selection policy. Reusing a selected policy is applying its action selection on the target task. However, such policy reuse will lead to a suboptimal policy for the target task when the source policy library is not sufficient (i.e., The actions of an optimal target policy for all states are not identical to any actions of source policies.). To enable the optimality guarantee for learning the target task, we augment the source policy library Π_s with primitive policies $\Pi_p = \{\pi_1, \pi_2, ..., \pi_{|A|}\}$, where policy $\pi_i \in \Pi_p$ takes action $a_i \in A$ for all states. Correspondingly, the source option set $\mathcal{O}_s$ is expanded to an option set $\mathcal{O}$ by including primitive options $\mathcal{O}_p$ created from primitive policies Π_p. Such augmentation ensures that all actions are available to all states.

To obtain an optimal source selection policy, we need to learn the option-value function $Q_{\mathcal{O}}^{\pi_{\mathcal{O}}}(s, o)$, which is defined as the expected discounted future reward starting in $s \in \mathcal{I}$, taking o, and henceforth following an inter-option policy $\pi_{\mathcal{O}} : \mathcal{S} \rightarrow \mathcal{O}$,

$$Q_{\mathcal{O}}^{\pi_{\mathcal{O}}}(s, o) = E\left\{r_{t+1} + \gamma r_{t+2} + \ldots | s_t = s, o_t = o, \pi_{\mathcal{O}}\right\}.$$

The optimal option value is defined as $Q_{\mathcal{O}}^{*}(s, o) = \max_{\pi_{\mathcal{O}}} Q_{\mathcal{O}}^{\pi_{\mathcal{O}}}(s, o)$. As we use the *call-and-return* model of option execution [20], the option-value function $Q_{\mathcal{O}}^{\pi_{\mathcal{O}}}$ also depends on when the execution of selected options terminates. Therefore, in addition to learning the optimal inter-option policy, we need to learn the termination functions for all options as well.

3.2 Context-Aware Policy Reuse

Given the inter-option learning formulation above, we here present our algorithm for learning both the optimal source selection policy and the termination functions for source policies and primitive policies during temporally-extended policy reuse.

Algorithm 1 Context-Aware Policy Reuse

1: Instantiate options $\mathcal{O}$ from source and primitive policies
2: Initialize option-value function $Q_{\mathcal{O}}$
3: Initialize termination function β_{θ_o} for all option $o \in \mathcal{O}$
4: **for** episode$= 1..M$ **do**
5: $s \leftarrow$ initial state
6: $o \leftarrow \epsilon$-greedy$(Q_{\mathcal{O}}, \epsilon, \mathcal{O}, s)$
7: **while** s is not terminal **do**
8: Execute $a = \pi_o(s)$ and obtain next state s' and reward r
9: $Q_{\mathcal{O}} \leftarrow$Update$Q_{\mathcal{O}}$ $(Q_{\mathcal{O}}, \beta_\theta, \mathcal{O}, s, a, s', r)$
10: Update termination function β_{θ_o} with $Q_{\mathcal{O}}$
11: **if** Option o terminates according to $\beta_{\theta_o}(s')$ **then**
12: $o \leftarrow \epsilon$-greedy$(Q_{\mathcal{O}}, \epsilon, \mathcal{O}, s')$
13: **end if**
14: $s \leftarrow s'$
15: **end while**
16: **end for**

Algorithm 1 illustrates CAPS using a tabular action-value function representation, which is also applicable with a function approximation. First, a set of policies $\mathcal{O}$ with parameterized termination functions are created based on the given source policy library Π_s and primitive policies (Line 1), as described in the previous subsection. Then we learn option-value function $Q_{\mathcal{O}}$ in the *call-and-return* model of option execution, where an option o is executed until it terminates based on its termination function β_{θ_o} and then a next option is selected by a policy over options $\pi_{\mathcal{O}}$, which is ϵ-greedy to $Q_{\mathcal{O}}$.

Learning Source Selection Policy

Assuming the options are Markov, $Q_{\mathcal{O}}$ is updated in Algorithm 2 satisfying the Bellman equation in an analogous way with intra-option Q-learning. Since options are temporal abstractions, the expected return of next state $U^*(s', o)$ is proportional to $\beta(s')$, the probability that option o terminates in state s'.

$$U^*(s', o) = (1 - \beta(s'))Q^*_{\mathcal{O}}(s', o) + \beta(s') \max_{o' \in \mathcal{O}} Q^*_{\mathcal{O}}(s', o').$$

Algorithm 2 Update$Q_{\mathcal{O}}$ $(Q_{\mathcal{O}}, \beta_\theta, \mathcal{O}, s, a, s', r)$

1: **for** $o_i \in \mathcal{O}$ **do**
2: **if** $a = \pi_{o_i}(s)$ **then**
3: $Q_{\mathcal{O}}(s, o_i) \leftarrow (1 - \alpha)Q_{\mathcal{O}}(s, o_i) + \alpha(r + \gamma U(s', o_i))$

$U(s', o_i) = (1 - \beta_{\theta_{o_i}}(s'))Q_{\mathcal{O}}(s', o_i) + \beta_{\theta_{o_i}}(s') \max_{o' \in \mathcal{O}} Q_{\mathcal{O}}(s', o')$

4: **end if**
5: **end for**
6: **return** $Q_{\mathcal{O}}$

Algorithm 3 ϵ-greedy $(Q_{\mathcal{O}}, \epsilon, \mathcal{O}, s)$

1: **if** random() $< \epsilon$ **then**
2: **return** an option o randomly selected from set $\mathcal{O}$
3: **else**
4: **return** $\text{argmax}_{o \in \mathcal{O}} Q_{\mathcal{O}}(s, o)$
5: **end if**

The value function $U(s', o)$ is an estimate of $U^*(s', o)$. We update option-value functions for all the options which select the same action as the current action a in order to make full use of experiences. Since $\beta_{\theta_{o_i}}(s')$ and $Q_{\mathcal{O}}(s', o_i)$ are different for each o_i satisfying the condition, $Q_{\mathcal{O}}(s, o_i)$ is updated differently for those options.

CAPS chooses a proper option o by utilizing ϵ-greedy strategy according to $Q_{\mathcal{O}}$ (Line 6, 12). With a probability of ϵ, we randomly choose an option, and with a probability of $1 - \epsilon$, we choose the option with the maximum Q-value. As ϵ never equals 0, all state-option pairs will be visited infinitely often. The pseudo code of ϵ-greedy is given in Algorithm 3.

Learning Termination Functions for Policy Reuse

As the selected source policy cannot be all the same as an optimal target policy, CAPS also needs to learn when to terminate the selected policy. CAPS learns termination functions in a similar way to [21], aiming to solve a multi-policy transfer learning problem, instead of option discovery. The objective of learning termination functions is to maximize the expected return U, so we can update the parameters of the termination functions with the following gradient:

$$\frac{\partial U(s_1, o_0)}{\partial \theta_{o_0}} = -\sum_{s', o} \mu_{\mathcal{O}}(s', o | s_1, o_0) \frac{\partial \beta_{\theta_o}(s')}{\partial \theta_o} A_{\mathcal{O}}(s', o), \tag{1}$$

where

$$\mu_{\mathcal{O}}(s', o | s_1, o_0) = \sum_{t=0}^{\infty} \gamma^t P(s_{t+1} = s', o_t = o | s_1, o_0).$$

$P(s_{t+1} = s', o_t = o | s_1, o_0)$ is the transition probability from initial condition (s_1, o_0) to (s', o) in t steps. Advantage function $A_{\mathcal{O}}(s', o)$ is approximated with $Q_{\mathcal{O}}(s', o) - \max_{o' \in \mathcal{O}} Q_{\mathcal{O}}(s', o')$.

The transition probability in Equation (1) is estimated from samples of the stationary on-policy distribution. For data efficiency, the discounted factor γ is neglected. So we update the parameter θ_o of termination function as follows:

$$\theta_o \leftarrow \theta_o - \alpha_\beta \frac{\partial \beta_{\theta_o}(s')}{\partial \theta_o} (Q_{\mathcal{O}}(s', o) - \max_{o' \in O} Q_{\mathcal{O}}(s', o')) \tag{2}$$

to identify transfer contexts (Line 10).

If Q-value of the current option o is not the largest among all the options, its termination probability grows, so the agent has a higher probability to switch to other better source policies. The termination probability of non-optimal options will converge to 1 eventually. CAPS achieves context identification autonomously by learning termination functions of the formulated options.

3.3 Theoretical Analysis

In this section, we provide theoretical analysis for CAPS. As guaranteed by Theorem 1, CAPS learns an optimal source selection policy that chooses the best source policy or primitive policy for each state.

Theorem 1 (Optimality on Source Selection Policy).

Given any source policy library Π_s*, bounded rewards* $|r_n| \leq R$*, learning rates* $0 \leq \alpha_t \leq 1$*, and* $\sum_{i=1}^{\infty} \alpha_{t^i}(s,a) = \infty$*,* $\sum_{i=1}^{\infty} \alpha_{t^i}^2(s,a) < \infty$ *for all* $(s,a) \in \mathcal{S} \times \mathcal{A}$*, then the CAPS algorithm converges w.p.1 to an optimal source selection policy.*

In addition, CAPS is able to converge to an optimal policy for a target task no matter what source policies are given. This theoretical guarantee is provided in Theorem 2.

Theorem 2 (Optimality on Target Task Learning).

Given any source policy Π_s*, bounded rewards* $|r_n| \leq R$*, learning rates* $0 \leq \alpha_t \leq 1$*, and* $\sum_{i=1}^{\infty} \alpha_{t^i}(s,a) = \infty$*,* $\sum_{i=1}^{\infty} \alpha_{t^i}^2(s,a) < \infty$ *for all* $(s,a) \in \mathcal{S} \times \mathcal{A}$*, then the CAPS algorithm converges w.p.1 to an optimal policy* π_g^* *for any target task* g*.*

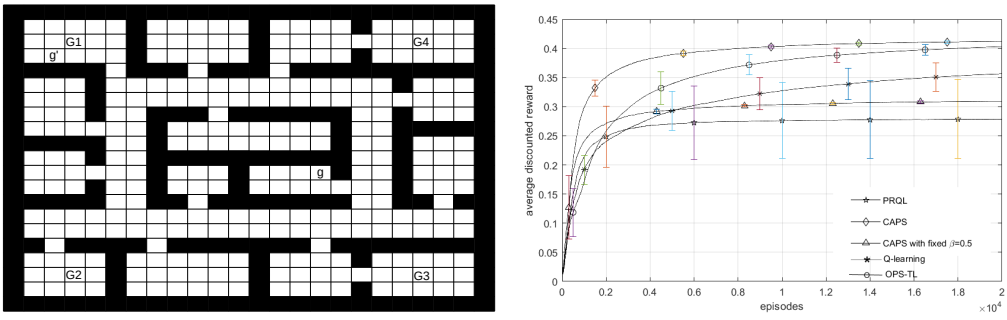
The proof of the theorems is given in the supplementary material. Although our basic proof structure of Theorem 1 is based on that of Q-learning, it extends the convergence and optimality guarantees of Q-learning to a more general setting of temporally-extended inter-option learning. In addition, Theorem 2 also extends such theoretical guarantees of traditional RL to a more general learning setting with or without reusing source policies.

4 Empirical Results

We compare CAPS with state-of-the-art policy reuse algorithms, PRQL [8] and OPS-TL [12], in addition to Q-learning and CAPS with fixed termination functions. We first consider a grid-based navigation domain used by [8] and [12]. We then augment all approaches with deep neural networks for function approximation and evaluate them in Pygame Learning Environment (PLE) [17]. In addition, we also do experiments in situations where transition functions of source and target tasks are different. The neural network structure and details of parameter settings are in the supplementary material.

4.1 Grid-based Navigation Domain

In the grid-based navigation domain, we define states of an agent by grids. Figure 1(a) shows goals of source and target tasks in the map. Initial states are randomly set and $G1, G2, G3, G4$ denote goals of source tasks. g and g' represent goals of different target tasks. Action space consists of *up, down, left and right*, four actions, which make an agent move in the corresponding direction with a step size of 1. The position of an agent is added a noise after each action to make stochastic MDP environment.

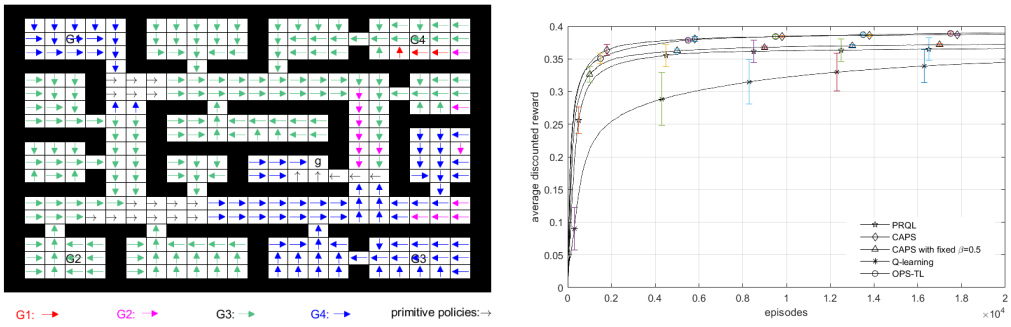


(a) (b)

Figure 1: (a) Source and target tasks in the map. (b) Average discounted rewards of CAPS, PRQL, OPS-TL and Q-learning on target task g.

Each learning process has been executed 10 times and the maximum episode length H is set as 100. The learning rates are set to 0.5 for Q_O and 0.2 for termination. The hyperparameters of PRQL and OPS-TL are consistent with their original papers [8, 12]. γ is set as 0.95. Q-learning is executed using an ϵ-$greedy$ strategy with $\epsilon = 1 - k/(k + 800)$, where k is the episode number.

Figure 1(b) shows the learning curves for the target task g, where all source tasks are not quite similar to the target task. CAPS significantly accelerates the learning process and dramatically outperforms OPS-TL and PRQL. Furthermore, PRQL results in negative transfers compared to Q-learning in this circumstance. With termination probability fixed as 0.5, the initial value for β, CAPS converges to a suboptimal policy, so it is necessary to learn when to terminate reusing the selected policies.

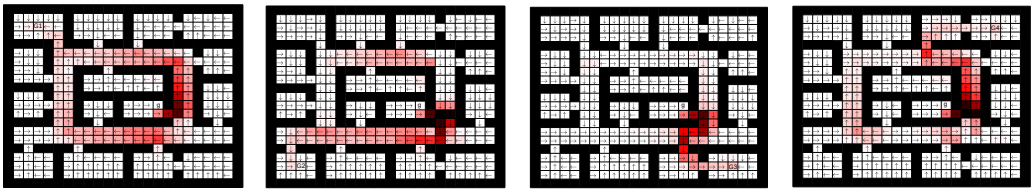


(a) (b)

Figure 2: (a) Options selected by CAPS to solve task g. (b) Average discounted rewards of CAPS, PRQL, OPS-TL and Q-learning on target task g'.

To better understand this significant outperformance, we illustrate how CAPS reuses source policies in target task g. From Figure 2(a), we can see that CAPS learns to choose the optimal policy to reuse for different contexts. The colors of the arrows represent the options CAPS takes in a greedy strategy. The directions of arrows denote the policies of selected options. For states around goal g, no source policy is useful for the target task, so CAPS chooses primitive options. For other states, CAPS chooses different source options. For example, source policy π_{G3} selected by CAPS navigates an agent out of the room in the lower left corner. We can say that, in some sense, CAPS decomposes a target task to subtasks according to existing knowledge.

Figure 3 illustrates the learned termination function for each source policy. The darker colors encode higher termination probabilities. The arrows denote the actions taken by each source policy in different states. The colors are darker in the states, where source policies are not consistent with π^*_g. The reuse of source policies needs to be terminated with a higher probability on states nearby goal g.



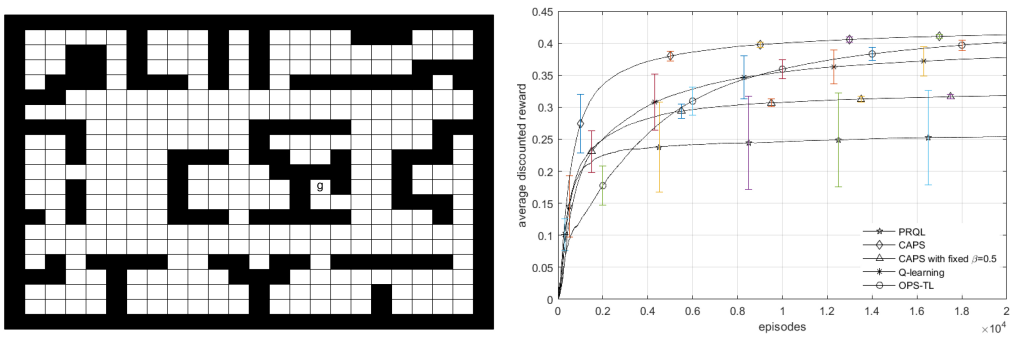
(a) source $G1$ (b) source $G2$ (c) source $G3$ (d) source $G4$

Figure 3: Termination probabilities of source policies when solving task g.

Figure 2(b) shows the learning curves for target task g', where there is a single best policy π_{G1} for reuse. Both CAPS and OPS-TL provably select π_{G1}. CAPS still performs slightly better than OPS-TL, because OPS-TL's ad hoc hyperparameter for specifying the termination function is hard to tune in practice while CAPS automatically learns termination functions during policy reuse. This slight outperformance indicates that concurrently learning to identify transfer contexts and selecting the best source policy does not sacrifice the learning performance.

To verify CAPS works as well in situations where transitions between source and target tasks are different, we conduct experiment on target task in Figure 4(a), whose map is much different from the

map of sources. The results shows that CAPS outperforms other methods even if only some parts of source policies can be reused. CAPS identifies the useful parts based on contexts automatically.



(a) (b)

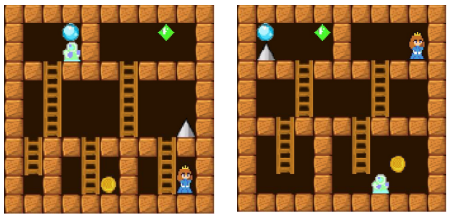
Figure 4: (a) Target task of different transitions from source tasks. (b) Average discounted rewards of CAPS, PRQL, OPS-TL and Q-learning on the left target task.

4.2 Pygame Learning Environment

4.2.1 Experimental Settings

Monsterkong of PLE is a complex navigation problem with simulated gravity. Two experimental settings are shown in Figure 5. The character under the blue gem in Figure 5(a) is an agent, whose initial position is randomly set on bricks. If the agent reaches a goal, it receives a reward of 1. Otherwise, it receives no reward. The action space consists of *up, down, left, right, jump* and *no-op*, six actions. The agent can move up and down only when it is on a ladder. Ineffective actions are treated as *no-op*s. An episode terminates in three cases: the agent reaches the goal, the agent touches a triangle spike or the timesteps exceed horizon $H = 1200$.

To illustrate the robustness of CAPS, we choose different objects as goals for source and target tasks in the two settings. In Figure 5(a), the green diamond, the blue gem, and the yellow coin are goals for source tasks. The princess is the goal for target task $g1$. As for task $g1$, there is no explicitly similar source task. In Figure 5(b), the green diamond, the yellow coin, and the princess are goals for source tasks. The blue gem is the goal for target task $g2$, which is in the same room with the green diamond. So there is one remarkably similar source task in the library to target task $g2$.



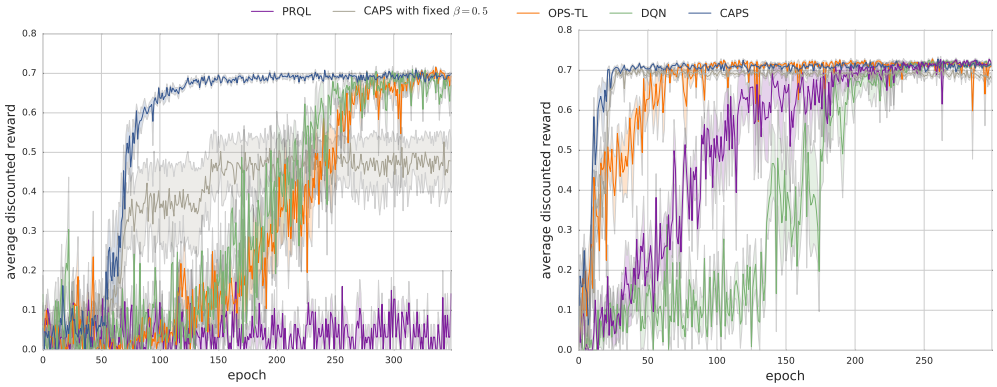
(a) (b)

Figure 5: Two different experimental settings. (a) The goals of source and target tasks are all in different rooms. (b) The goals of one source task and the target task are in the same room.

4.2.2 Results

Shown in Figure 6(a), the average rewards of solving task $g1$ are evaluated 5000 steps every 12500 training steps (one epoch) for CAPS and other baseline methods The hyperparameters of all methods are tuned to give the best performance for this experiment. Each learning process has been executed for 5 times.

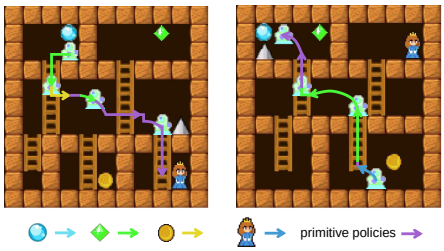
The learning curve of CAPS starts to rise at about 50 epochs and converges to the optimal value in 125 epochs, which is significantly faster than other methods. Previous methods can only benefit from one source task, so they perform poorly when source tasks are much different from the target task. PRQL even suffers from negative transfers. With fixed $\beta = 0.5$, CAPS converges to a suboptimal policy, which illustrates the importance of a proper termination to the policies selected. Since the rewards are averaged for only one time evaluation of 5000 steps, the reward curves in this experiment shake more severely than those of last experiment, which are averaged from the start.



(a) (b)

Figure 6: Average discounted rewards of CAPS, PRQL, OPS-TL and DQN on target tasks $g1$ (a) and $g2$ (b) for 5000-step evaluation per epoch.

Moreover, we show the performance comparison of solving task $g2$ in Figure 6(b). CAPS converges to the optimal policy the fastest when there is a source task remarkably similar to the target task. Since the source knowledge in this setting is more useful than that of task $g1$, the learning performance of CAPS is significantly better. PRQL and OPS-TL also show positive transfers in this experimental setting. We provide videos of the learned policies by CAPS for those two tasks at `https://goo.gl/gY68x1`. We also validate CAPS on target tasks with different transitions from sources on Pygame. The results are in the supplementary material.



(a) (b)

Figure 7: Trajectories of the agent for an episode to solve task $g1$ (a) and $g2$ (b).

We further demonstrate how CAPS choose policies to reuse in Figure 7. The arrows in each figure show a complete trajectory of the agent from an initial position to the goal. The colors of arrows denote different policies the agent selects. In Figure 7(a), at the beginning, the agent chooses a source policy with the green diamond as its goal to navigate out of the room. After that, the agent switches to another source policy to get closer to the princess. Finally, since goals of source and target tasks are all in different rooms, the agent has to utilize primitive policies to reach the goal of task $g1$.

5 Related Work

Policy Reuse. Most state-of-the-art policy reuse methods lack theoretical guarantees and analysis. [22] proposes a policy reuse method, mainly working on short-lived sequential policy selection without learning a full policy. On the contrary, our method converges to the optimal policy for a target task. To reuse a selected source policy, [8, 12] combine it with a random policy according to the episode length. Such an ad hoc reuse strategy has a great effect on the transfer performance and its hyperparameters are hard to tune. In contrast, CAPS automatically learns when and which source policy to reuse. [23] reuses transition samples obtained from one task to accelerate learning of another, but it is constrained with the assumption that transition samples can be shared across tasks. [24] initializes a target policy with a single mapped source policy via unsupervised manifold alignment, which is unable to reuse multiple source policies.

Multi-Task Learning (MTL) co-learns a set of tasks jointly via some shared knowledge [25, 26, 27], so an agent needs the environment information of all the tasks. However, our approach does not require to know source task models. MLSH is a hierarchical MTL method learning a master policy and several sub-policies with fixed length [28]. However, unlike our method, it cannot learn when to

terminate the sub-policies autonomously. Moreover, MLSH has no theoretical guarantee for optimal convergence.

Option Learning. In contrast to the works of option discovery [21, 29, 30, 31], CAPS focuses on multi-policy reuse. Although some methods have been proposed to reuse options, they suffer from several limitations. For example, [19] assumes the given options are fixed and their reuse cannot be adapted to a target task structure for more efficient transfer. Source policies in [14] are restricted to be learned in a PAC-learning way to obtain a ϵ-optimal option library. [16] learns terminations for policies via value iteration with an environment model. [15] assumes the given policy library is sufficient and converges to a locally optimal termination function. However, CAPS has no requirement to an environment model, sufficiency of the source policy library or how it is acquired. In addition, CAPS adaptively learns terminations for source policies.

6 Summary

In this paper, we develop a novel multi-policy reuse method, called *Context-Aware Policy reuSe* (CAPS), that leverages knowledge from multiple source policies and greatly accelerates RL. CAPS automatically learns to identify which contexts are the most appropriate for reuse which source policy. To our best knowledge, CAPS is the first multi-policy transfer reinforcement learning approach that not only learns optimal source selection, but also provides the same optimality guarantee of the target policy learning as Q-learning. Results from both toy experiments and deep-learning experiments show that CAPS significantly outperforms other state-of-the-art policy reuse methods, and verify that effectively and concurrently utilizing multiple source policies is crucial to improve transfer efficiency. CAPS can grow the policy library by adding the newly learned policy for the target task represented in the option framework. To support lifelong learning, it is also important to identify whether the new policy is necessary to be added to the policy library, which is part of future work.

7 Supplementary

7.1 Proof of Theorem 1

Theorem 1 (Optimality on Source Selection Policy).

Given any source policy library Π_s, bounded rewards $|r_n| \leq R$, learning rates $0 \leq \alpha_t \leq 1$, and $\sum_{i=1}^{\infty} \alpha_{t^i}(s,a) = \infty$, $\sum_{i=1}^{\infty} \alpha^2_{t^i}(s,a) < \infty$ for all $(s,a) \in \mathcal{S} \times \mathcal{A}$, then the CAPS algorithm converges w.p.1 to an optimal source selection policy.

Proof. We apply the update rule of $Q_{\mathcal{O}}$ to each option o that takes the same action with the current action a taken by an agent:

$$Q_{\mathcal{O}}(s,o) \leftarrow (1-\alpha)Q_{\mathcal{O}}(s,o) + \alpha(r + \gamma U(s',o)). \tag{3}$$

Then we subtract $Q^*_{\mathcal{O}}(s,o)$ from both sides of the update function (1) and defining $\Delta_t(s,o) = Q_{\mathcal{O},t}(s,o) - Q^*(s,o)$ together with

$$F_t(s,o) = r + \gamma U(s',o) - Q^*_{\mathcal{O}}(s,o).$$

The learning rule of $Q_{\mathcal{O}}$ can be seen as the iterative process of Theorem 1 in [32].

$$
\begin{aligned}
&|E\left\{F_t\left(s,o\right)\right\}| = \left| r + \gamma \sum_{s'} P(s'|s,a)U(s',o) - Q^*_{\mathcal{O}}(s,o) \right| \\
=&\left| r+\gamma \sum_{s'} P(s'|s,a)U(s',o) - (r+\gamma \sum_{s'} P(s'|s,a)U^*(s',o)) \right| \\
=&\Bigg| \gamma \sum_{s'} P(s'|s,a) \Big[(1-\beta_o(s'))(Q_{\mathcal{O}}(s',o) - Q^*_{\mathcal{O}}(s',o)) \\
&+\beta_o(s')(\max_{o'\in\mathcal{O}} Q_{\mathcal{O}}(s',o') - \max_{o'\in\mathcal{O}} Q^*_{\mathcal{O}}(s',o')) \Big] \Bigg| \\
\leq&\gamma \sum_{s'} P(s'|s,a) \max_{s'',o''} \left| Q_{\mathcal{O}}(s'',o'') - Q^*_{\mathcal{O}}(s'',o'') \right| \\
=&\gamma \max_{s'',o''} \left| Q_{\mathcal{O}}(s'',o'') - Q^*_{\mathcal{O}}(s'',o'') \right|
\end{aligned}
$$

As a result, $E\{F_t(s,o)\}$ has a contraction property.

$$var[F_t(s,o)|\mathcal{F}_t] = var[r + \gamma U(s',o)|\mathcal{F}_t],$$

where $\mathcal{F}_t = \{\Delta_t, \Delta_{t-1}, ..., F_{t-1}, ...\alpha_{t-1}, ..., 1-\alpha_{t-1}, ...\}$ represents the past at step t. Because r is bounded,verifies

$$var[F_t(s,o)|\mathcal{F}_t] \leq C(1 + ||\Delta_t||^2_W),$$

where C is some constant and $||\cdot||_W$ denotes some weighted maximum norm. Since $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$ and ϵ in Algorithm 2 never equals 0, all the conditions of Theorem 1 in [32] are satisfied. $Q_{\mathcal{O}}$ converges w.p.1 to the optimal Q-function. Following a greedy strategy to $Q_{\mathcal{O}}$ ($o(s) = \text{argmax}_{o\in\mathcal{O}} Q_{\mathcal{O}}(s,o)$) , CAPS converges to an optimal source selection policy for policy library Π. □

7.2 Proof of Theorem 2

Theorem 2 (Optimality on Target Task Learning).

Given any source policy Π_s, bounded rewards $|r_n| \leq R$, learning rates $0 \leq \alpha_t \leq 1$, and $\sum_{i=1}^{\infty} \alpha_{t^i}(s,a) = \infty$, $\sum_{i=1}^{\infty} \alpha^2_{t^i}(s,a) < \infty$ for all $(s,a) \in \mathcal{S}\times\mathcal{A}$, then the CAPS algorithm converges w.p.1 to an optimal policy π^*_g for any target task g.

Proof. The termination function of $\forall o \in \mathcal{O}$ is defined as:

$$\beta_{\theta_o}(s) = \frac{1}{1+e^{-\theta_o(s)}}.$$

The update rule of θ is:

$$
\begin{aligned}
\theta_o^{t+1} &= \theta_o^t - \alpha_\beta \frac{\partial \beta_{\theta_o^t}(s)}{\partial \theta_o^t} (Q_{\mathcal{O}}(s,o) - \max_{o'} Q_{\mathcal{O}}(s,o')) \\
&= \theta_o^t - \alpha_\beta \beta_{\theta_o^t}(s)(1-\beta_{\theta_o^t}(s))(Q_{\mathcal{O}}(s,o) - \max_{o'} Q_{\mathcal{O}}(s,o'))
\end{aligned}
$$

Let $\theta_o^{t+1} = f(\theta_o^t)$ and $o_g(s) = \text{argmax}_{o_i\in\mathcal{O}} Q_{\mathcal{O}}(s,o_i)$. Because $f(\theta^t_{o_i})$ monotonically increases and $\theta^{t+1}_{o_i} > \theta^t_{o_i}$ for $\forall o_i \in \mathcal{O}\backslash\{o_g\}, s \in \mathcal{S}$, if all state-option pairs can be visited infinitely often,

$$\lim_{t\to\infty} \theta^t_{o_i}(s) \to \infty.$$

$$\lim_{t\to\infty} \beta_{\theta^t_{o_i}}(s) = \frac{1}{1+e^{-\lim\limits_{t\to\infty} \theta^t_{o_i}(s)}} = 1.$$

According to Theorem 1, $Q_{\mathcal{O}}$ converges to $Q^*_{\mathcal{O}}$. Because the termination functions of the non-optimal options converge to 1, the greedy policy obtained in the call-and-return option execution model is an optimal policy for task g. □

7.3 Neural Network Structure and Parameter Settings

CAPS is also applicable with a function approximation. We use a deep neural network to approximate option-value function $Q_{\mathcal{O}}$ and termination function β. Our network structure has the same convolutional structure as DQN [3]. There are 3 convolutional layers followed by 2 fully-connected layers shown in Figure 8.

Q_O is trained off-policy with experience replay and target network, while β is trained online with fixed learning rate as 0.00025. We assume the output of the last but one layer as the learned representations of states, so we only train the last layer when learning β. We also employ double Q network [33] and gradient clipping [34].

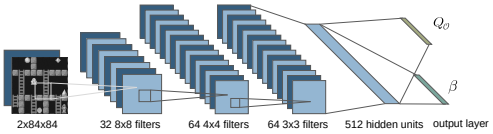
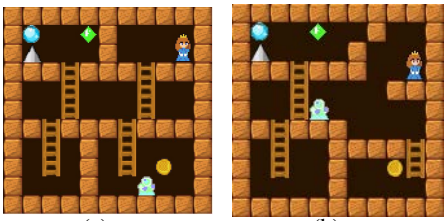


Figure 8: Neural Network Structure.

We perform a training step on $Q_{\mathcal{O}}$ each step with minibatches of size 32 randomly sampled from a replay buffer of one million transitions every 4 transitions encoded into the replay buffer. The learning rate of $Q_{\mathcal{O}}$ is annealed piecewise linearly from 10^{-4} to 5×10^{-5} over the first 2.5 million training steps, then fixed at 5×10^{-5} after that. The training process of $Q_{\mathcal{O}}$ and β begins after 5×10^4 transitions. ϵ is annealed piecewise linearly from 1 to 0.05 over the first 4.375 million training steps. γ is set as 0.99. We add a regularization $\rho = 0.005$ to the approximated advantage function in the update function of termination probability in a similar way to [21]. Since the bricks surrounding the images in Figure 2 are useless, we clip the bricks and down-sample the remaining part to 84×84. Then we convert the preprocessed images to gray-scale, stack the last two and feed them to the network.

7.4 Experiments with Different Transitions in PLE

To test the performance of CAPS when source and target tasks have different transitions, we transfer the source policies of task $g2$ in Section 4.2 to learn a target task $g2'$ shown in Figure 9(b). The blue gem is the goal for task $g2'$.



(a) (b)

Figure 9: Experiment settings of source tasks (a) and target task $g2'$ (b).

Shown in Figure 3, CAPS converges at about 50 epochs, while other policy reuse methods almost cannot accelerate learning in this experiment. Our transfer performance is a little worse than that of task $g2$ in Section 4.2, but still converges faster than other methods. We believe the robustness to the changes of state space between source and target tasks is due to the generalization ability of neural networks. The results illustrate that CAPS works as well when transition functions of source and target tasks are different.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[2] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

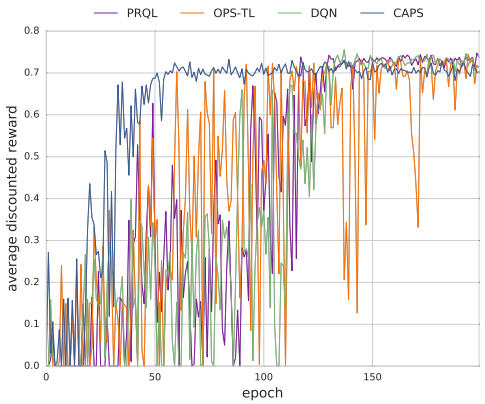


Figure 10: Discounted rewards of CAPS, PRQL, OPS-TL and DQN on target tasks $g2'$ for 5000-step evaluation per epoch.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Lisa Torrey and Jude Shavlik. Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 1:242, 2009.

[5] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017.

[6] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, David Silver, and Hado P van Hasselt. Successor features for transfer in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4058–4068, 2017.

[7] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[8] Fernando Fernandez and Manuela M Veloso. Learning domain structure through probabilistic policy reuse in reinforcement learning. *Progress in Artificial Intelligence*, 2(1):13–27, 2013.

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

[10] Haitham Bou Ammar, Eric Eaton, Matthew E Taylor, Decebal Constantin Mocanu, Kurt Driessens, Gerhard Weiss, and Karl Tuyls. An automated measure of MDP similarity for transfer in reinforcement learning. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[11] Eric Mazumdar, Roy Dong, Vicenç Rúbies Royo, Claire Tomlin, and S Shankar Sastry. A multi-armed bandit approach for online expert selection in markov decision processes. *arXiv preprint arXiv:1707.05714*, 2017.

[12] Siyuan Li and Chongjie Zhang. An optimal online method of selecting source policies for reinforcement learning. *arXiv preprint arXiv:1709.08201*, 2017.

[13] Mohammad Gheshlaghi Azar, Alessandro Lazaric, and Emma Brunskill. Regret bounds for reinforcement learning with policy advice. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 97–112. Springer, 2013.

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

[15] Gheorghe Comanici and Doina Precup. Optimal policy switching algorithms for reinforcement learning. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 709–714. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[16] Timothy Mann, Daniel Mankowitz, and Shie Mannor. Time-regularized interrupting options (trio). In *International Conference on Machine Learning*, pages 1350–1358, 2014.

[17] Norman Tasfi. Pygame learning environment. `https://github.com/ntasfi/PyGame-Learning-Environment`, 2016.

[18] Chris Watkins and Peter Dayan. Technical note Q-learning. *Machine Learning*, 8:279–292, 1992.

[19] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[20] Doina Precup, Richard S Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In *European conference on machine learning*, pages 382–393. Springer, 1998.

[21] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

[22] Benjamin Rosman, Majd Hawasly, and Subramanian Ramamoorthy. Bayesian policy reuse. *Machine Learning*, 104(1):99–127, 2016.

[23] Romain Laroche and Merwan Barlier. Transfer reinforcement learning with shared dynamics. In *AAAI*, pages 2147–2153, 2017.

[24] Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew E Taylor. Unsupervised cross-domain transfer in policy gradient reinforcement learning via manifold alignment. pages 2504–2510, 2015.

[25] Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998.

[26] Anestis Fachantidis, Ioannis Partalas, Matthew E Taylor, and Ioannis Vlahavas. Transfer learning with probabilistic mapping selection. *Adaptive Behavior*, 23(1):3–19, 2015.

[27] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821*, 2013.

[28] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

[29] Anna Harutyunyan, Peter Vrancx, Pierre-Luc Bacon, Doina Precup, and Ann Nowe. Learning with options that terminate off-policy. *arXiv preprint arXiv:1711.03817*, 2017.

[30] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. *arXiv preprint arXiv:1709.04571*, 2017.

[31] Nicholas K Jong, Todd Hester, and Peter Stone. The utility of temporal abstraction in reinforcement learning. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 299–306. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[32] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994.

[33] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, pages 2094–2100, 2016.

[34] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.