Proceedings of the 34th Chinese Control Conference
July 28-30, 2015, Hangzhou, China

Shaping in Reinforcement Learning via Knowledge Transferred from Human-Demonstrations

WANG Guofang, FANG Zhou, LI Ping, LI Bo
School of Aeronautics and Astronautics, Zhejiang University, Hangzhou 310027, P. R. China
E-mail: {gfwang89, zfang, bli}@zju.edu.cn, pli@iipc.zju.edu.cn

Abstract: Transfer has been widely used to ameliorate the slow convergence speed of reinforcement learning (RL) by reusing the previous obtained knowledge from other related but distinct tasks. In this paper, we propose a framework to transfer knowledge learned directly from human-demonstration trajectories of source tasks to shape the RL algorithm in target task, so as to avoid the time-consuming training process of RL in source tasks and thus we expand the learning paradigm of transfer in RL domains. In our framework, rather than transferring the most common value function or policy, we adopt the visit frequencies of states in successful demonstration trajectories as the acquired knowledge, and then perform transfer via shared agent space. Simulation experiments in obstacle avoidance problems suggest that the transferred knowledge could accelerate the learning process in target task obviously. And as a case study, the experiments show the potential of our framework in knowledge transfer in RL tasks.

Key Words: Reinforcement learning, Human-demonstrations, transfer

1 Introduction

Over the past few decades, RL has attracted a lot of research due to its property of model-free and online optimal control. However, it may run into 'curse of dimensionality' when applied to deal with complex problems, and the learning speed may become an inevitable problem [21]. In addition, RL is regarded as a tabula rasa learning technique, meaning that the agent knows nothing about the environment at the start of learning and it has to take random action to explore the unknown environment. All of these will render RL time-consuming or even infeasible in practice [8].

In order to speed up the learning process, many techniques such as experience replay (ER) and transfer learning (TL) are introduced to RL domains. The essence of transfer learning is that the generalization of knowledge may occur not only within tasks but also across tasks [22]. And it is apparent that the more similar the source and target tasks are, the easier it is to transfer knowledge between them. With the purpose of designing good transfer methods, three aspects must be taken into account: firstly, when to transfer, i.e. the selection of source and target tasks where the agent could get and then reuse knowledge; secondly, what to transfer, i.e. the form of knowledge retained from source tasks; thirdly, how to transfer, i.e. the method by which the agent reuses the acquired knowledge in target tasks.

Transfer learning has initially been studied within the supervised learning framework since 1980s [20], and not until recent years does transfer in RL domains gain much attention [23]. However, there are still some limitations for transfer in RL domains:

First of all, many approaches [7, 15, 22] need the human to specially design relatively easy source tasks and then train RL to gain knowledge for transfer; for another, the application range of RL is limited as its design of reward function is a difficult problem and needs much expert knowledge; finally, RL is very sensitive to even slight changes of tasks [2, 3, 5]. In a word, sometimes RL would not be suitable to acquire knowledge from source tasks.

Supported by National Natural Science Foundation of China (61004066) and Zhejiang Provincial Natural Science Foundation (LY15F030005).

For such cases, in order to circumvent these limitations, we had better use some other methods such as learning from demonstration (LfD) to get knowledge, or to say, the expert knowledge is incorporated in human-demonstrations. LfD does not necessarily require the reward function and it is robust to changes of tasks. Moreover, as LfD avoids the time-costly random exploration in RL and the demonstration trajectories may be gathered during the lifetime of an agent, they are inexpensive but important date sets of knowledge, it is easy to bring about strong transfer (the total time of solving source task and target task with transfer is less than the time of solving the target task directly) [22]. In brief, it is meaningful and feasible to transfer knowledge included in demonstrations to shape the RL algorithm in target task.

As to what to transfer, in this paper the visit frequencies of states in demonstration trajectories of source tasks instead of the commonly used value function or optimal policy as the transferred knowledge, and finally, the knowledge is used to shape (one of the most popular methods to incorporate human knowledge) RL algorithm in target tasks directly via shared agent space [14]. To the best of our knowledge, our work is the first trial to transfer directly from demonstrations to RL.

This paper is organized as follows. Section 2 briefly presents RL and LfD. Section 3 introduces our method to transfer from demonstration trajectories to RL, and then it is used in section 4 to be evaluated. Section 5 presents and discusses the results and finally, Section 6 comes a conclusion and the future work.

2 Background

2.1 Reinforcement Learning

In RL, we usually model the underlying control problem as Markov Decision Process (MDP) described by a tuple $M = \langle S, A, P, R, \gamma \rangle$, where S denotes state space, A denotes action space, $P : S \times A \times S' \to [0, +\infty)$ is the probability of transition to state s' when taking action a in state s, $R : S \times A \times S' \to \Re$ is the reward the agent gets along with the transition, and γ is the discount factor that expresses the extent to which the agent prefers immediate reward over future rewards [21].

The goal of the agent is to find an optimal control policy that maximizes the long-term cumulative rewards of every state, also known as returns or values.

Temporal-difference (TD) learning combines Monte Carlo's directly learning from raw experience without dynamic models and bootstrapping (a technique in which estimates are updated based on other learned estimates) of dynamic programming (DP), and it is the most central part of RL. Among RL, Q learning [21] is one of the most famous algorithms to obtain optimal state-action value. It uses:

$$\begin{aligned} Q(s_t, a_t) \leftarrow & Q(s_t, a_t) \\ & + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right], \end{aligned} \tag{1}$$

to update the Q value function.

And we use reward shaping in Q learning as an example of RL.

2.2 Learning from Demonstration

LfD, also named apprenticeship learning, is a technique that develops optimal or sub-optimal policies from the demonstrated state to action mappings [1, 5], it can be seen as supervised learning from labelled data.

It can be used where it is difficult to define a formal specification of the control objective, for example, there is no obvious measuring for 'flying well' of unmanned aerial vehicle, or to say, the immediate reward function may be difficult to design and it requires considerable expertise knowledge [2]. Another advantage of LfD over RL is that the player may demonstrate task any times before a good policy could be derived even though the demonstrations are not perfect, and moreover it avoids millions of times of trial and error training as what is often done in RL.

Once we have got demonstration trajectories at hands, we can derive a policy by using techniques like mapping function, classification and regression. Or we can also reconstruct reward function from the demonstrations to assist the RL, which is called Inverse Reinforcement Learning (IRL) [18]. IRL is an ill-posed problem which may make the problem even more complex, let alone use the deduced knowledge to transfer.

To mine knowledge from the demonstration trajectories, there are two main limits that should be noted: the not-accessed states and poor quality demonstration trajectories. But in this paper, we use the statistical radius of some state in many demonstration trajectories to overcome these limitations.

3 Transfer from Demonstrations to RL

In this section, we present our specific framework to transfer knowledge deduced from the demonstration trajectories of source tasks to shape the RL algorithm in target task.

3.1 Reward Shaping

In RL, reward is the immediate cost of taking an action in a state. Reward shaping [19] refers to modification of the reward function to provide guidance or more 'hints' to an agent to help it learn faster.

Effects of reward shaping on Q learning can be expressed by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \begin{pmatrix} r_{t+1} + \gamma \max\limits_{a'} Q(s_{t+1}, a') \\ + F(s_t, a_t, s_{t+1}) - Q(s_t, a_t) \end{pmatrix}, \tag{2}$$

where $F(s_t, a_t, s_{t+1})$ is the shaping function, and shaping can help improve the learning performance by altering the updates.

Designing appropriate reward shaping function is very important. Ng et al. [17] propose an efficient way to construct potential-based reward shaping function:

$$F(s, a, s') = \gamma \phi(s') - \phi(s), \tag{3}$$

where $\phi(s)$ may be a function indicating the goal for the task or potential function, in another way, it is goal-oriented heuristic in navigation problems, and Ng et al. prove it would converge to the same optimal policy π^* while helping accelerate the process.

Transfer in RL domains could improve the learning performance exactly by autonomously construct this function appropriately.

3.2 Goal State Indicator

Goal state indicator is defined as the signal that can indicate where the goal state is or which state is preferred by the agent.

From this point of view, the value function is basically a formal goal indicator in sense that it is similar to potential function $\phi(s)$. For example, the region near the goal may have higher value in simple navigation problems. Knowledge transfer in RL domains by agent space between distinct but related tasks substantially transfer the goal indicator included in value function V or Q to assist the learning process in target tasks [13].

Except for value function, there are many other goal indicators, such as reward gradient [16], visit frequency [9], etc. The indicators may be used in hierarchical reinforcement learning (HRL) to construct options (the macro-action the agent can take) or in LfD to identify sub-goals. Their effects of accelerating the learning are determined by their performances of indicating the goal state. Inspired by this, the visit frequencies of states on successful trajectories are adopted as goal indicators. The reasons are as follows:

The discounted reward function $J(\pi)$ for a RL task can be denoted by [12]:

$$\begin{aligned} J(\pi) &= E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} | d_0, \pi \right\} \\ &= \int_S d_\gamma^\pi(s) \int_A \pi(s, a) R(s, a)\, da ds, \end{aligned} \tag{4}$$

where $d_\gamma^\pi(s) = \sum_{k=0}^{\infty} \gamma^k p(s_k = s | d_0, \pi)$ is the discounted state distribution under policy π [6] and $R(s, a)$ denotes the expected reward when the agent takes action a in state s.

During learning, the agent will have to estimate the cost-to-go function J for a given policy π. The resulting estimate of J is called the value function. The state value function:

$$\begin{aligned} v^\pi(s) &= E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{k+1} | x_0 = x, \pi \right\} \\ &= \int_S d_\gamma^\pi(s) \int_A \pi(s, a) R(s, a)\, da ds \end{aligned} \tag{5}$$

After the RL algorithm has converged to the optimal policy π^*, from equation (5), the state value can be written as:

$$v^{\pi^*}(s) = \int\limits_S d_\gamma^{\pi^*}(s) \int\limits_A \pi^*(s,a) R(s,a)\, da ds. \tag{6}$$

When the agent select the greedy action with probability 1 and equation (6) can be simplified as:

$$v^{\pi^*}(s) = \int\limits_S d_\gamma^{\pi^*}(s)\, R(s,a^*)\, ds, \tag{7}$$

where a^* is the greedy action according to policy π^*.

In some navigation problems, the start point is randomly distributed, $R(s,a^*)$ may be a simple reward function, for example, a constant negative reward after taking an action, so there is a positive correlation between $\int\limits_S d_\gamma^{\pi^*}(s)\, ds$ and $v^{\pi^*}(s)$ of a state. For state s, when the other states' distribution is fixed, if its distribution is high, its value may be high.

Sample the states with the distribution $d_\gamma^{\pi^*}(s)$ by N times, or to say, the player demonstrates the task for a few times. And the successful demonstration trajectories can be seen as the trajectories got by optimal or near optimal policy. We may get the visit frequencies of states:

$$n(s) = d_\gamma^{\pi^*}(s)\, N, \tag{8}$$

which is an unbiased sampling of state distribution, when the start point is randomly distributed, the more times it is visited, the more confident it is subgoal state or goal state.

Based on this, as value function is a good goal state indicator, there are reasons to regard visit frequencies of states in successful human-demonstrations as an indicator of goal state. In other words, the discount distribution of a state may be high when it is near goal state.

3.3 Learning Reward Shaping Function

Konidarias et al. [14] introduce the definition of agent space: the agent may be equipped with a very rich set of sensors such as laser range finders, temperature gauges, etc. In other words, in obstacle avoidance problems, the agent faces a series of tasks determined by parameter θ and θ determines the place of obstacles and the goal or start point of the task, then a sensor can be defined as a function mapping from θ and current place s_θ:

$$sensor \ : \qquad f(\theta, s_\theta) \rightarrow d.$$

When facing a particular task, the agent would construct a task-specific representation called the problem space that captures the essential features of tasks from those sensors. But all the sensors' readings denoted by d_i for the i^{th} state remain the same meaning and construct the agent space. Konidarias et al. prove RL could take place in problem space and transfer could occur via agent space. And the agent does not directly learn in agent space because the environment defined by them may not be Markov or the agent space is too large to learn in.

Knowledge transfer in RL domains needs the different but related tasks to be reward-linked [14] which means the reward function for all tasks is constructed by the same sensors. Similarly, the tasks should be feature-linked or the goal may be indicated by the same feature.

Select a number of demonstration episodes and then count visit times of every integer state's neighborhood inside a circle of radius r, denoted by n_i. We may estimate the state value function as:

$$v_i \propto cn_i, \tag{9}$$

where c is a coefficient that is unknown variant and v_i denotes the value of i^{th} state.

In RL, what we try to learn is a mapping function defined as:

$$V \ : \ s_i \rightarrow v_i.$$

The function V maps problem-specific state descriptors or state space variables to expected return, but it is not portable among tasks as the meaning of state is changed between different tasks. Here we define L to be another value like function, mapping from portable agent space descriptors to expected return based on that the agent space stays the same during the process:

$$L \ : \ d_i \rightarrow n_i.$$

Once the agent has completed some episodes of source tasks and counted visit times n_i for every integer state, it can use (d_i, n_i) pairs as training examples for supervised learning to get L. Since L is portable, and the tasks are feature linked, we could use it to provide a good initial estimate for v when facing a new different but related task. The flow of the framework is shown in Figure 1.

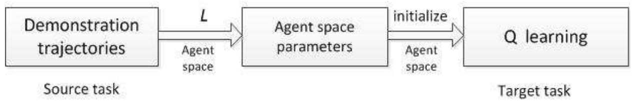

Fig. 1: The transfer framework

Take a typical navigation problem as a example, and assume that in the scenario there are five beacons the agent can detect as the invariable agent space. We place the first 4 beacons randomly in the state space and the 5^{th} near the goal location. Assume every beacon emits a signal that decays with the square of the Euclidean distance away from the beacon. Then the j^{th} reading of state *i* can be denoted by:

$$d_{ij} = \frac{1}{(\text{distance})_j^2 + 0.1}, \tag{10}$$

where the bias 0.1 is used to avoid 0-value denominator.

Thus we could learn the mapping L from the five beacon signals and a constant as features to fit the visit frequencies (requiring 6 parameters). A simple linear model of L can be written as:

$$n_i = \beta_0 d_{i0} + \beta_1 d_{i1} + \beta_2 d_{i2} + \beta_3 d_{i3} + \beta_4 d_{i4} + \beta_5 d_{i5}, \tag{11}$$

where $d_{i0} = 1$ denotes the constant feature.

A high signal level from the 5^{th} beacon predicts high value or high visit frequency implying the goal state, and the others should be ignored. This is very informative that could be easy to learn, and can be well approximated even with a linear L, and then, the heuristic can be transferred by this

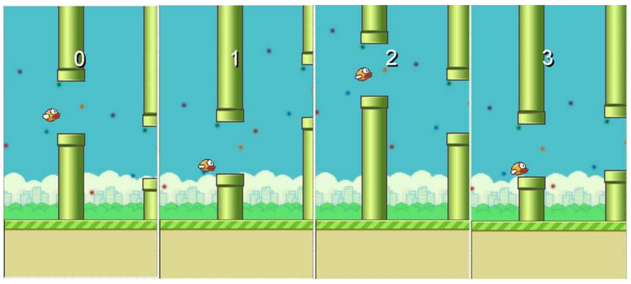

Fig. 2: The flappy bird game

simple function to initialize the state value in target task as the shaping function.

This is just one of the common selected agent space features, and there are some other formulation of agent space [14].

4 Experiment Setup

We select the popular game named flappy bird as the source task, and a simple maze grid as the target task.

The two tasks are very different both from the state space and the action space, let alone the dynamic models. But we could use the same agent equipped with a very rich set of sensors to solve the two tasks. Then these tasks are somehow 'related' and the agent could be able to reuse the knowledge transferred from the flappy bird to help solve the maze grid problem faster.

4.1 Source Task Setup

In the flappy bird problem as in Figure 2, when the bird touches the up edge or falls to the floor or collides with the tube, the game is over and the bird returns to the start point. And the dynamic model of the bird is:

$$\begin{cases} x_{t+1} = x_t + 1 \\ y_{t+1} = y_t + v_t \\ v_{t+1} = v_t + g \end{cases}, \tag{12}$$

where $0 \leq x_t \leq 80$ is the horizontal distance to start point of the bird, $0 \leq y_t \leq 200$ is the absolute height of bird, $g = 0.1356$ is the gravity coefficient and when the player presses the button the velocity denoted by v is set to 40.

Breaking the process into episodes from tube to tube, it is quit obvious that both the start point and end point are changing along with episodes. We cannot guarantee the RL algorithm to converge to optimal policy with probability one in every case and RL may converge to different sub-optimal policies. Or even, the agent should train the source task one by one to get knowledge and it is very time-consuming.

But we could manipulate the bird to cross many tubes, and get the position tracking of the bird assisted with agent space as the demonstration trajectories easily. The agent space is made up of 5 beacons' signals as the colorful stars show in figure 2. It should be point out that the 5^{th} beacon is near the goal state, the other four are randomly distributed in the state space. But from episode to episode, the 5 beacons' relative places to goal state are fixed.

4.2 Target Task Setup

The target problem is to find the goal location in a 60×60 grid maze, as shown in Figure 3. In this figure, beacon

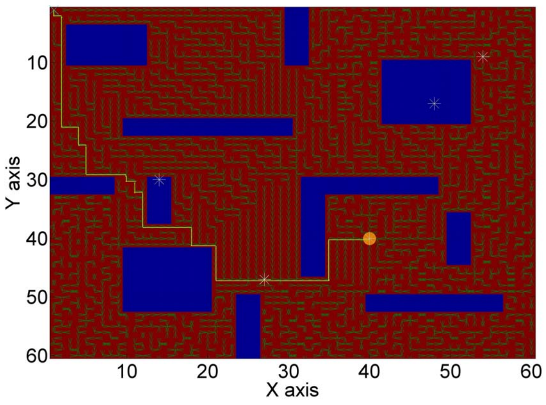

Fig. 3: The optimal action of every state for maze grid

locations are shown as bright stars, also, the 5^{th} beacon is near the goal, and the other four are randomly distributed in the state space. The start point is shown as a green cross, and the goal is shown as a large green dot. The blue space stands for the cliff meaning that if the agent gets into the region, a reward of -100 is given and the agent returns to the start point directly. Only when the agent finds the goal does one episode end. The action space of maze grid includes up, down, right and left. After taking an action, position of the agent moves one grid in the corresponding direction. The direction of arrows in Figure 3 denote the greedy action of every state after Q learning. From the figure, we can find the optimal policy from the start point easily.

We could use Q-learning to obtain the optimal policy, but the parameters needed to store are $60 \times 60 \times 4$=14400 and finding the exact state action value function needs many transition samples. There is a need to introduce transfer learning to accelerate the learning.

As the action spaces of the two problems are different, we may transfer state value V instead of state-action value Q. Before learning in the target task, we should initialize the Q state-action function value with the transferred V transformed by the equation:

$$Q(s,a) = E\left\{ r + \gamma V(s') \,|s,a \right\}. \tag{13}$$

Then we can evaluate the transfer performance.

5 Simulation Results and Analysis

Demonstration trajectories for flappy bird game by a skilled player can be found in Figure 4. In the figure, x axis denotes the horizontal distance to the right edge of the next tube and y axis denotes the vertical distance to the middle line of the next window. The blue stars and black circles stand for the points that a demonstration trajectory has visited, and the difference is that at the black circle point the player presses the button to raise the flappy bird.

From Figure 4, we know that the start points of different episodes are randomly distributed on the left side, and as the time goes on they converge to a small region on the right. We could count the visit times of every integer state to identify the goal region. The more times a state is visited, the more confident we are that it is the goal or sub-goal in the task.

In figure 5, we could find that the visit frequencies in demonstration trajectories of the middle height state are big-

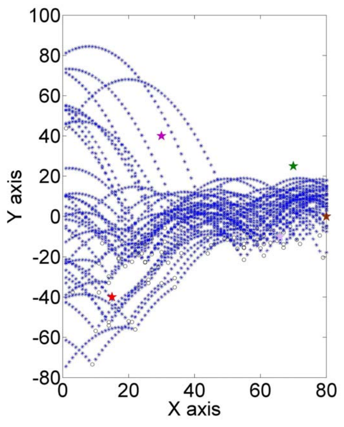

Fig. 4: Demonstration trajectories of flappy bird game

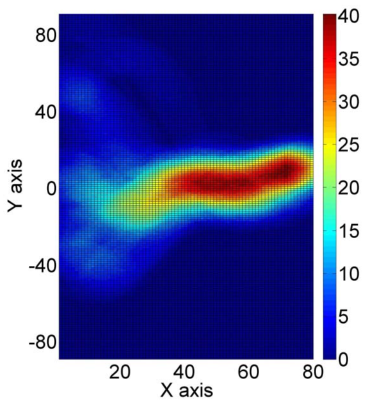

Fig. 5: Visit frequencies of different states in source task

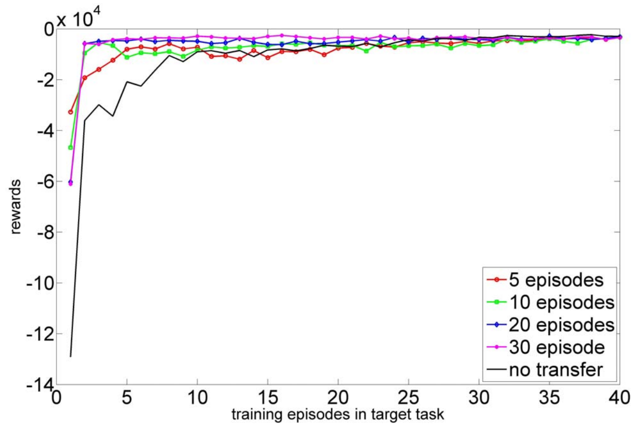

Fig. 6: Comparison with different demonstration episodes

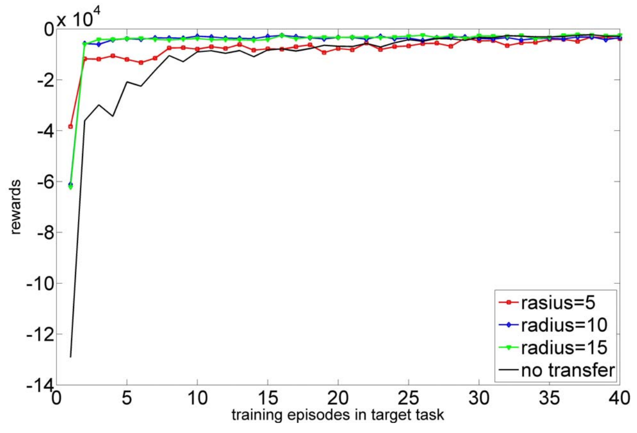

Fig. 7: Comparison with different counting radius

ger than that of states of the same x. This implies that the bird should get close to the middle height as quickly as possible.

Figure 6 shows the rewards (averaged over 40 runs) of Q learning to reach the goal as the agent repeats episodes in the target task (note that L is never updated in the target task), and it also compares the rewards of transfer from different number of demonstration trajectories, and Q learning without transfer.

From the comparison in Figure 6, we know that transfer can significantly reduce the rewards the agent gets to find the goal in target task in all transfer cases in the first few episodes, reducing from over -100,000 to at most just over -60,000 and after 3 episodes to be under -20,000. We come to a conclusion that the learned shaping function can significantly improve performance during the first few episodes of learning as expected. It also shows that the first episode to find the goal may cost more in transfer from more demonstration trajectories than transfer from less demonstration trajectories, but the performance of transfer from more demonstration episodes progresses faster than transfer from less demonstration trajectories. The more demonstration trajectories, the more accurate visit frequencies of states we obtain, so the Progressive performance of transferring from more demonstration trajectories should be better.

Figure 7 shows the rewards (averaged over 40 runs) of Q learning to reach the goal as the agents repeat episodes in the target task, comparing the rewards of different radius for counting visit frequencies and Q learning without transfer.

It is clear from Figure 7 that counting radius in a small scale can improve the RL performance significantly during the first few episodes of learning in target tasks. It also implies that the first episode to find the goal may cost more in transfer with mid-counting radius, but the performance progress faster than with small or large counting radius. And the heuristic by transfer from state value of RL and by counting with small radius may hinder the learning after few episodes compared with counting with mid-radius.

In summary, this transfer method can improve the performance of RL in some degree, and that it is robust to different counting radius and demonstration episodes. But there are some limitations before widespread use of the method. The selected indicator must have the ability to reflect the goal state, and it is especially fit for goal search in navigation problems.

6 Related Work

As far as we know, there has not yet research in transfer from demonstration trajectories to RL, but it is similar to transfer in RL domain in some sense. And many existing research is related to us.

Taylor et al. use an inter-task mapping to transfer knowledge [22], no matter through value function, Q-Value Reuse and so on. But most of the time it is the task of a domain expert to hand code these mapping, and that this mapping only relates features so it is difficult to obtain transfer between potentially very different function approximators.

Related to this, Fernandez [11] suggest we can reuse the learned policies from the source task to the target task. But there is a fatal limit that the state and action must be same or in other words only the reward function can change.

Fachantidis et al. [10] transfer the model of the transition and reward functions of a source task to a relevant but different target task, and then the agent takes a hybrid approach, implementing both model-free and model-based learning. But all the two methods are limited by the need of inter-task mapping. Though several methods have been proposed to get the inter-task mapping autonomously [4], but the computation is expensive and transfer performance may be not obvious.

Lazaric et al. [15] transfer samples in source task to target task as additional data to improve performance. Transition samples from source task are used along with a small set of sample transitions in a new task to compute a similarity measure, and are sampled according to the similarity. But reusing such samples requires their state descriptors to be the same.

Konidaris et al. [13, 14] introduce agent space, and explains what does related but different tasks mean. Then they show us that RL happens in problem space, but transfer in knowledge form or in option form can take place through agent space.

Pieter et al. [3] leverage expert demonstrations to efficiently learn good controllers for tasks being demonstrated by an expert, they first use the inverse RL to learn the reward function, and then use RL to learn.

7 Conclusion and Future Work

We have presented a framework for transfer from demonstration trajectories in a simple task to RL in another complex task via shared agent space. The framework broadens the use of knowledge transfer which now mostly transfers value function or optimal policy between two RL algorithms in different but related tasks. Through our method, we can integrate LfD, transfer learning and RL to solve a complex task. Then we empirically show that this framework can be successfully applied to improve learning speed, and it helps us build agents that are capable of improving their own intelligence to solve related tasks by reusing the experience during their lifetime.

Proposals for future work include finding other ways of goal identification or even when there are multi-subgoals in source tasks. We could also broaden the method and theory by using the knowledge deduced from source tasks with uncertain dynamic.

References

[1] Abbeel, P.; Coates, A.; Quigley, M.; and Ng, A. Y. 2007. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems* 19:1.

[2] Abbeel, P.; Coates, A.; and Ng, A. Y. 2010. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*.

[3] Abbeel, P. 2008. *Apprenticeship learning and reinforcement learning with application to robotic control*. ProQuest.

[4] Ammar, H. B.; Taylor, M. E.; Tuyls, K.; and Weiss, G. 2012. Reinforcement learning transfer using a sparse coded inter-task mapping. In *Multi-Agent Systems*. Springer. 1–16.

[5] Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

[6] Bhatnagar, S.; Sutton, R. S.; Ghavamzadeh, M.; and Lee, M. 2009. Natural actor–critic algorithms. *Automatica* 45(11):2471–2482.

[7] Busoniu, L.; Babuska, R.; De Schutter, B.; and Ernst, D. 2010. *Reinforcement learning and dynamic programming using function approximators*. CRC press.

[8] Croonenborghs, T.; Driessens, K.; and Bruynooghe, M. 2008. Learning a transfer function for reinforcement learning problems. In *Belgian-Dutch Conference on Machine Learning (Benelearn08), Spa, Belgium*, 19–2.

[9] Digney, B. L. 1998. Learning hierarchical control structures for multiple tasks and changing environments. In *Proceedings of the fifth international conference on simulation of adaptive behavior on From animals to animats*, volume 5, 321–330.

[10] Fachantidis, A.; Partalas, I.; Tsoumakas, G.; and Vlahavas, I. 2013. Transferring task models in reinforcement learning agents. *Neurocomputing* 107:23–32.

[11] Fernandez, F., and Veloso, M. 2006. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, 720–727. ACM.

[12] Grondman, I.; Busoniu, L.; Lopes, G. A.; and Babuska, R. 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 42(6):1291–1307.

[13] Konidaris, G., and Barto, A. 2006. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, 489–496. ACM.

[14] Konidaris, G.; Scheidwasser, I.; and Barto, A. G. 2012. Transfer in reinforcement learning via shared features. *The Journal of Machine Learning Research* 13(1):1333–1371.

[15] Lazaric, A.; Restelli, M.; and Bonarini, A. 2008. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, 544–551. ACM.

[16] McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. *Computer Science Department Faculty Publication Series* 8.

[17] Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, 278–287.

[18] Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *Icml*, 663–670.

[19] Ng, A. Y. 2003. *Shaping and policy search in reinforcement learning*. Ph.D. Dissertation, University of California, Berkeley.

[20] Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on* 22(10):1345–1359.

[21] Sutton, R. S., and Barto, A. G. 1998. *Introduction to reinforcement learning*. MIT Press.

[22] Taylor, M. E., and Stone, P. 2009. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research* 10:1633–1685.

[23] Taylor, M. E., and Stone, P. 2011. An introduction to intertask transfer for reinforcement learning. *AI Magazine* 32(1):15.