Effective Transfer via Demonstrations in Reinforcement Learning: A Preliminary Study

Zhaodong Wang
School of EECS
Washington State University
zhaodong.wang@wsu.edu

Matthew E. Taylor
School of EECS
Washington State University
taylorm@eecs.wsu.edu

Abstract

There are many successful methods for transferring information from one agent to another. One approach, taken in this work, is to have one (source) agent demonstrate a policy to a second (target) agent, and then have that second agent improve upon the policy. By allowing the target agent to observe the source agent's demonstrations, rather than relying on other types of direct knowledge transfer like Q-values, rules, or shared representations, we remove the need for the agents to know anything about each other's internal representation or have a shared language. In this work, we introduce a refinement to HAT, an existing transfer learning method, by integrating the target agent's confidence in its representation of the source agent's policy. Results show that a target agent can effectively 1) improve its initial performance relative to learning without transfer (jumpstart) and 2) improve its performance relative to the source agent (total reward). Furthermore, both the jumpstart and total reward are improved with this new refinement, relative to learning without transfer and relative to learning with HAT.

Introduction

Reinforcement learning (Sutton and Barto 1998) (RL) methods have been successfully applied to both virtual and physical robots. In some complex domains, learning speed may be too slow in practice to be feasible. One common speed up method is transfer learning (Taylor and Stone 2009), where one (source) agent is used to speed up learning in a second (target) agent. Unfortunately, many transfer learning methods make assumptions about the source and/or target agent's internal representation, learning method, prior knowledge, etc. Instead of requiring a particular type of knowledge to be transferred, our past work on the Human Agent Transfer (Taylor, Suay, and Chernova 2011) (HAT) algorithm allowed the source agent to demonstrate its policy, the target agent to bootstrap based on this policy, and then the target agent to improve its performance over that of the source agent. Without restriction of how the source agent should perform demonstration, HAT could have a specific way of recording data from the source as state-action pairs, though this process might vary from one domain to another. In this work the source agent is actually a human, underlying how different the source and target agents can be. We also note that this approach is different from much of the existing learning from demonstration (Argall et al. 2009) approaches, as the target agent can autonomously improve upon the source agent's policy via RL.

The HAT algorithm can be briefly summarized in three steps. First, the source agent acts for a time in the task and the target agent records a set of demonstrations. Second, a decision tree learning method (e.g., J48 (Quinlan 1993)) summarizes the demonstrated policy as a static mapping from states to actions. Third, these rules are used by the target agent as a bias in the early stages of its learning.[1] The key component of HAT is that it uses the decision tree to bias its exploration. Initially, the target task agent follows the decision tree, attempting to mimic the source agent. Over time, it incorporates random exploration and exploitation of its learned knowledge with exploiting the decision tree, effectively improving its performance relative to the source agent.

Immediately after performing transfer, it is unlikely that the target agent will be optimal due to multiple sources of error. First, the source agent may be suboptimal. Second, the source agent may be inconsistent, resulting in an inability to correctly classify states into actions selected by the agent. Third, the source data must be summarized, not memorized. Because the decision tree will not exhaustively memorize all possible states, when it combines multiple (similar) states, some states may be classified with incorrect actions. Fourth, the source agent typically cannot exhaustively demonstrate all possible state action pairs — the learned decision tree must generalize to unseen states, which may be incorrect. Different types and qualities of demonstrations may be more or less effective in HAT, depending on these four types of potential errors.

Error types two (inconsistent actions) and three (state aggregation), and possibly error type four (action generalization), may be addressed by considering the uncertainty in the classifier. Rather than blindly following a decision tree to select an action in a given state, as is done in the cur-

[1] If the source agent and target agent are in different domains, an inter-task mapping may be used between steps two and three in order to translate such rules (Taylor and Stone 2007).

rent version of HAT, we believe the transferred information should be weighted by the estimated uncertainty in the classification.

This work takes a critical first step in this direction by introducing GPHAT, an enhancement to the HAT algorithm that uses Gaussian Processes. We evaluate GPHAT using the domain of simulated robot soccer and empirically show it outperforms both HAT and learning without transfer. Even when low amounts of demonstration data are used, the initial performance (jumpstart) and total reward are significantly improved. By leveraging uncertainty in the estimate of the source agent's policy, GPHAT may be useful in domains where initial performance is critical, but demonstration from a trained agent (or human) is available but non-trivial to collect.

Background

This section will present some basic techniques discussed in the paper: reinforcement learning, demonstration, and human agent transfer.

Reinforcement Learning

Reinforcement Learning is such a process that learning through agent's experience exploring the environment. RL's basic model is Markov decision processes (MDPs). MDPs work as follows: Once the agent observes its current state, it will make some decision or action and then update its state. For a MDP, A is a set of actions, S is a set of states. There are two main functions within this process:

1. Transition Function

$$T : S \times A \mapsto S$$

2. Reward Function

$$R : S \times A \mapsto R$$

Different RL algorithms have different ways of learning to maximize the expected reward. Here we focus on an on-policy time difference learning algorithm: SARSA (Rummery and Niranjan 1994; Singh and Sutton 1996) in our domain, which is learning an action-value function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$$

This $Q(s, a)$ function will map state-action pairs to a numerical expectation value. Our RL agent will explore by selecting the action that maximizes $Q(s, a)$ with probability $1 - \epsilon$ and select a random action with probability ϵ.

Demonstration

Demonstration data are presented in a form of vector of state-action pairs as $\langle \vec{x}, a \rangle$, in which $\vec{x}$ is the state vector and a is the corresponding action.

Because of the simple expression of demonstration data, they could be generated by a human or by another agent and via different ways of realisation, and there are many approaches of collecting the demonstration data such as visual observation or letting the agent write down those state-action pairs.

In cases where the state is continuous or very large, Q can not be represented as a table. In such cases, some type of function approximation is needed. In this paper we use a CMAC tile coding function approximator (Albus 1981), where a state is represented in terms of a vector of state variables.

Human Agent Transfer

HAT has been proposed to help bootstrap the target agent's RL performance. Its contribution is to build a common bridge between source and target agents, since it might not possible to simply copy a human agent's behavior to some RL agent.

The transfer part of HAT is *rule transfer* (Taylor and Stone 2007) — the source knowledge is summarized via a decision tree which will provide an action choice for a given state. The following steps summarize HAT:

1: Learn a policy from the source task. A human agent has some policy ($\pi : S \mapsto A$) on a task, and takes actions upon current state following the policy. Meanwhile, those state-action pairs will be stored as demonstration data.

2: Train a decision tree. Upon the learned policy data, a decision tree will be learned to summarize the state-action pairs. The decision tree could be understood as a static set of rules.

3: Bootstrap target task with the decision tree. Instead of randomly exploring, an agent could make use of the rules to take actions at the beginning. There are three ways of using the decision tree to improve learning performance: 1) value bonus, 2) extra action, or 3) probabilistic policy reuse (PPR). This paper focuses on PPR. In PPR, there is a self-decaying probability Φ deciding whether to follow the rules or use SARSA. An agent will reuse the transferred rule policy with a probability of Φ, act randomly with a probability of ϵ, and exploit its Q-values with probability $1 - \Phi - \epsilon$. Φ typically starts near 1 and decays exponentially, forcing the target to initially follow the source policy and leverage its learned Q-values over time.

Gaussian Process Human Agent Transfer

In order to improve HAT, especially in continuous and complicated domains, we would prefer a more continuous model to summarize the demonstration data and provide better decisions. Meanwhile, we may want to leverage the agent's confidence in the decision suggested by the prior knowledge. Through the measurement of confidence, the agent should probably follow those certain decisions and ignore other uncertain ones.

Our new approach (GPHAT) leverages Gaussian Processes as the kernel function of the summarized data, which is continuous and will come with numerical confidence of suggested action choice upon encountered states.

A typical Gaussian process model is defined as:

$$P(\omega_i|x) = \frac{1}{\sqrt{2\pi|\Sigma_i|}} \exp\{-\frac{1}{2}(x - \mu_i)^T \Sigma_i^{-1}(x - \mu_i)\}$$

where ω_i is the predicted label, Σ_i is the covariance matrix of data of class i, and μ_i is the mean of data of class i.

Considering Bayes decision rules (BDR), we have the classifier:

$$\begin{aligned} \omega_i &= \arg\max_{\omega_i}[\ln P(\omega_i|x) + \ln P(\omega_i)] \\ &= \arg\max_{\omega_i}[-\tfrac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i) \\ &\quad -\tfrac{1}{2}\ln 2\pi|\Sigma_i| + \ln P(\omega_i)] \\ &= \arg\min_{\omega_i}[d_i(x,\mu_i) + \alpha_i] \end{aligned}$$

where $d_i(x,\mu_i) = (x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)$ and $\alpha_i = \ln 2\pi|\Sigma_i| - 2\ln P(\omega_i)$.

Notice that here in the Gaussian classifier, the distance is measured with the Mahalanobis distance, $d_i(x,\mu_i)$, and this is because the state variables are all hand-selected and they are not equally the same in terms of numerical property.

This classifier is generated from Bayes decision rules and it optimizes the boundary of the data with different labels. If we directly use the above classifier, we can only receive a binary decision. Instead, we define a confidence function with the classification:

$$C_i = \frac{\exp\{-d_i(x,\mu_i) - \alpha_i\}}{Normalization} \qquad (1)$$

Notice that a typical GP maps from input space (data) to output space (class), but this still just provide classification result. Additionally, what we want is the confidence along with the classification output, and thus we take advantage of original GP and then define the above confidence function to calculate confidence.

A learning agent could use this confidence in multiple ways. In this work, we have unbalanced data — one action is executed much more often than others. When our target agent attempts to exploit source knowledge, it will execute the action suggested by the classifier if it is above some confidence threshold. Otherwise, it will execute the "default" action. Notice that we have implemented probabilistic policy reuse in the method, letting a decaying probabilistic variable Φ decide whether the agent should use the prior knowledge or not at each step. Step one in GPHAT is the same as in HAT. Step two of GPHAT trains a Gaussian classifier instead of a decision tree. Step 3 of GPHAT is outlined in Algorithm 1.

In addition to the above, we can also use clustering to help determine the number of Gaussian classifiers to use (which can be greater than or equal to the number of actions). We first roughly cluster these data using Expectation-maximization (EM) algorithm (Celeux and Govaert 1992) of Weka 3.6 (Witten and Frank 2005) into N groups and train N Gaussian classifiers for this class. We determine the number of N used in actual learning by comparing the average performance of the first few episodes. As shown in Figure 1, we could have several smaller groups of data by clustering data first, which could potentially increase the precision of Gaussian classifiers.

Algorithm 1: GPHAT: Bootstrap target learning

```
Input: Classifier $GP$, confidence threshold $T$, PPR
       initial probability $\Phi_0$, PPR decay $\Phi_D$
$\Phi \leftarrow \Phi_0$
for each episode do
    Initialize state $s$ to start state
    for each step of an episode do
        $a \leftarrow \emptyset$
        if rand() $\leq \Phi$ then
            Use $GP$ to compute $C_i$ as shown in (1) for
              each action
            if $max\ Ci \geq \Phi$ then
                $a \leftarrow$ corresponding $a_i$
        if $a == \emptyset$ then
            if rand() $\leq \epsilon$ then
                $a \leftarrow$ random action
            else
                $a \leftarrow$ action that maximizes $Q$
    Execute action $a$
    Observe new state $s'$ and reward $r$
    Use SARSA to update $Q$
    $\Phi \leftarrow \Phi \times \Phi_D$
```

Experimental Setting

This section explains the Keepaway (Stone et al. 2006a) domain and then discusses the experimental methodology.

Keepaway Simulation

Keepaway is a simulated soccer game domain. In this paper, we use the Robocup Soccer Server (Noda et al. 1998) (version 9.4.5) and version 3.3 UvA Keepaway players (Stone et al. 2006b). In this simulation, there are five players: 3 keepers and 2 takers playing within a bounded square. Keepers learn to keep control of the ball as long as possible but takers are following hard-coded rules to chase after the ball to intercept it. An episode of the simulated game starts with an initial state and ends with an interception by the takers or the ball going out of bounds.

The game is mapped into a discrete time sequence to make it possible to control every players. We use a continuous 13-tuple vector to represent the states containing relative distances and angles. Figure 2 shows how we determine those variables. Figure 3 is a screen capture of the Keepaway simulation. Once a keeper get the ball, it will make a decision among three actions: 0) *Hold*: hold the ball, 1) *Pass*$_1$ pass the ball to the closer teammate, 2) *Pass*$_2$ pass the ball to the further teammate, while the takers and the other two keepers currently without the ball will just follow hand-coded policies to try to get open for a pass. The reward is $+1$ per time step for every keeper.

Methodology Setup

To collect demonstration data from human teacher, we show the visualized game (shown in Figure 3) to a human, and let

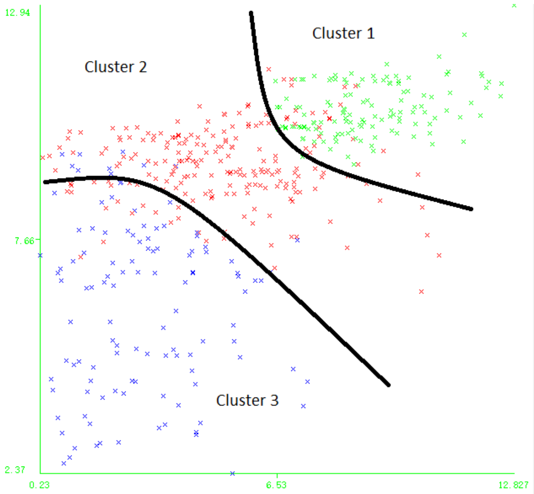

Figure 1: This figure shows an example of clustering same class data into three smaller groups.

the player execute actions (*Hold*, *Pass*$_1$, *Pass*$_2$). Meanwhile we record the state-action pairs as demonstration data.

For HAT, we use J48 tree with the default parameters of Weka 3.6. For our GP method, we train Gaussian classifiers only on actions *Pass*$_1$ and *Pass*$_2$, since action *Hold* is executed roughly 70% of the time, making the data unbalanced. Notice that the Gaussian classifier for *Pass*$_1$ is trained by treating other action data as same outliers, and so for *Pass*$_2$. To make fair comparisons, we further define double DTs, where two decision trees are trained exclusively for *Pass*$_1$ and *Pass*$_2$.

Different RL algorithms could be applied into HAT or GPHAT but we restrict ourselves to considering SARSA in this paper. Here we use: $\alpha = 0.05$, $\epsilon = 0.10$, $\gamma = 1$. To use the prior knowledge, we have a probabilistic parameter Φ determining whether to listen to prior knowledge. Φ will be multiplied by a decay factor, Φ_D, on every time step. Among $\{0.9, 0.99, 0.999, 0.9999\}$, preliminary experiments found $\Phi_D = 0.999$ to be the best.

The learning performance is measured by jumpstart and total reward. Jumpstart is the average duration of episodes using only transferred prior knowledge. Higher jumpstart indicates that the transferred knowledge could be more helpful to an agent. Total reward could be compared through the area under an learning curve, which suggests the overall learning performance within a certain period.

Results

Here we show the improvement brought by human demonstrations and reveal how the different demonstration data influences the RL performance. We also evaluate the different performances of different human demonstrations and explain the possible reason for this.

To make comparisons between different human demonstrations, we consider two kinds of demonstrations with distinct differences in game strategy. For each of them, we let a human player play Keepaway using the keyboard.

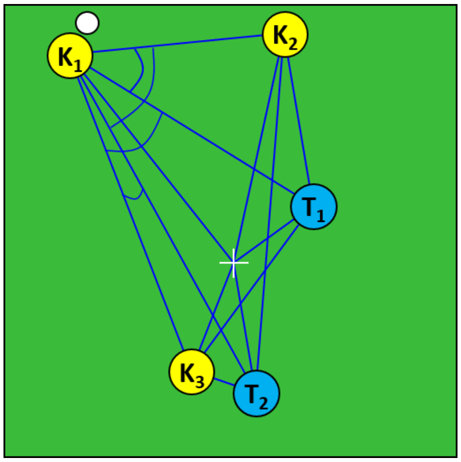

Figure 2: This figure shows the state variables used as the 13-tuple state vector. These Euclidean variables are made up relative angles and distances. While playing, the keeper (K_i) currently with the ball needs to decide whether to hold the ball or to pass it to one of its two teammates. The two hard-coded takers (T_1 and T_2) are chasing after the ball and the three keepers will learn to keep control of the ball.

Figure 3: This is a snapshot of the real-time running game. All players are restricted in the square court with the white line bound.

The human player attempted to follow two policies:

1. *Simple-Response demonstrations:* The human player always holds the ball until the takers approach very near to the keeper with the ball. The player only chooses to pass the ball to another teammate when necessary. The average demonstration duration is 10.5s with a standard deviation of 3.5s.

2. *Complex-Strategy demonstrations:* The human player is more flexible and active in this setting. The player will try to pass the ball more often, and thus the keepers need to zigzag to chase the ball. However, the player also tries to act inconsistently when possible, so that the player would not always take the same action as long as those actions are also rational. The average demonstration duration is 10.1s with a standard deviation of 3.8s.

Each demonstration contains state-action pairs of 20 episodes. Note that these demonstrations are on the order

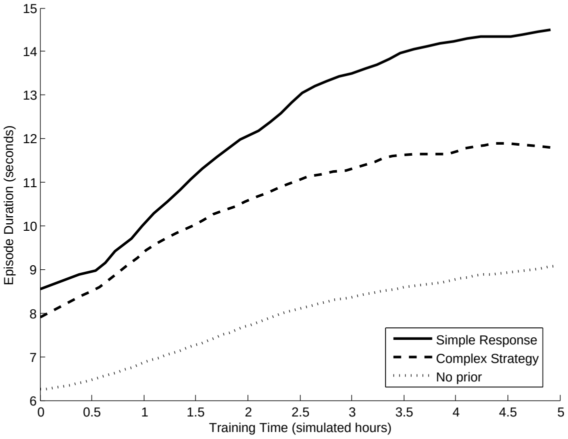

Figure 4: This figure presents the learning curves of different learning methods. The episode duration is calculated by the average of a sliding window of 500 episodes. "No prior" represents the curve of RL without any prior knowledge. "Complex Strategy" represents HAT using the Complex-Strategy demonstrations. "Simple Response" represents HAT using the Simple-Response demonstrations. Episode durations are averaged over 10 independent trials.

of minutes in the simulator, while learning will occur over hours of simulator interaction.

HAT

For different learning implementations, we record every episode's duration time while training the agent of Keepaway, and show the results as in Figure 4. As expected, both sets of demonstrations allow HAT to outperform learning without any prior.

However, notice that there is a large difference between the two demonstrations. We hypothesize that the "Complex Strategy" was more difficult to summarize due to its inconsistent policy, resulting in worse performance.

Since there is difference between these two kinds of demonstrations and we are using J48 decision tree to classify the state-action pairs, the classified outcome of the two demonstrations are different. Indeed, while we are trying to capture the true distribution of human demonstrations, a more complicated set of demonstration data will lead to a lower-confidence hypothesis, compared to a same-size set of data with a more straight-forward or distinct distribution.

Table 1 shows some parameters of the J48 pruned tree that we have used for summarizing human demonstration. Depth and accuracy indicate that the Complex Strategy needs a decision tree with more complexity relative to the Simple Response. 10-fold cross-validation is used to report the accuracy of the different methods.

GPHAT

As we can see from previous section, the distribution of demonstration data could surly affect the effectiveness of summarized prior knowledge and thus influence the agent's learning performance. This section we will show the advantage of applying Gaussian processes.

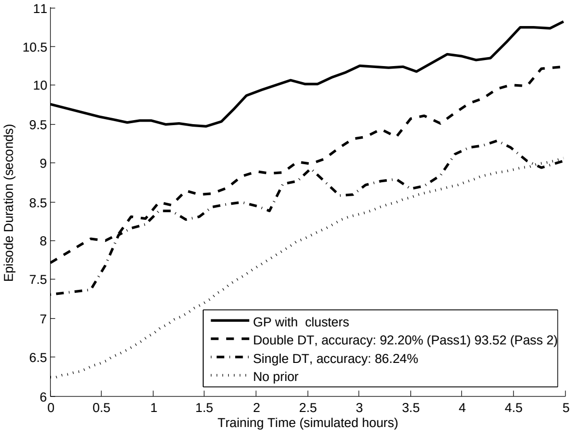

Figure 5: This figure shows the learning performance using "Novice" demonstrations. We compare GPHAT with HAT using one or double decision trees. Performance is averaged over 10 independent trials.

Consider an even worse case where we have a demonstration with an average performance of 7.45s with a standard deviation of 2.2s, which is only slightly better than a random policy. This "Novice" demonstration will often provide poor actions. We will compare the robustness of using decision trees and Gaussian processes.

Figure 5 shows the learning performance comparison. To make the comparison fair, we have trained 2 decision trees to individually classify action 1 and 2, just like we have 2 sets of Gaussian classifiers for action 1 and 2 respectively. Table 1 is the comparison of jumpstart among GP and DTs.

We conclude that with Gaussian process, the transferred prior knowledge is more robust even if the quality of human demonstration is limited. We also see that the performance of using prior knowledge is better than the original performance of the teacher. One way of explaining this is that even if the demonstrator fails to take an action 1 or 2 to pass the ball, which will lead to an end of current episode, we still record many correct actions. Thus the only bad effects would be more redundant data of action 0 (holding the ball).

Tuning GPHAT's Confidence Threshold

Previous sections have used a fixed confidence threshold of 0.9 for GPHAT. In this section, we are using the demonstration data generated by a learned agent and show the impact of this parameter. The learned agent has been doing reinforcement learning for a while and its episode duration is 12.4 seconds with a standard deviation of 3.6s.

Generally speaking, higher threshold of confidence could guarantee the cautiousness of taking a suggested action, while a lower one could bring more randomness. What should be noticed is that this kind of randomness is not uniform — in the Keepaway domain, randomness results in ex-

Demonstration	Single DT			Double DTs			GPHAT		
	Jumpstart	Depth	Accuracy	Jumpstart	Depth	Accuracy	Jumpstart	Clusters	Accuracy
Simple Response	+2.23	4	87.52%	+2.53	4	90.61%	+3.42	2	83.21%
Complex Strategy	+1.76	7	67.21%	+2.26	6	84.36%	+3.37	3	80.16%
Novice	+1.13	5	86.24%	+1.44	5	92.86%	+3.49	2	84.37%
Learned Agent	+3.18	4	88.67%	+3.26	4	91.22%	+4.55	2	86.12%

Table 1: The table shows overall comparisons among different methods upon different demonstration data. For double DTs, depth and accuracy are averaged over the two trees. For Gaussian processes, the confidence threshold is 0.9.

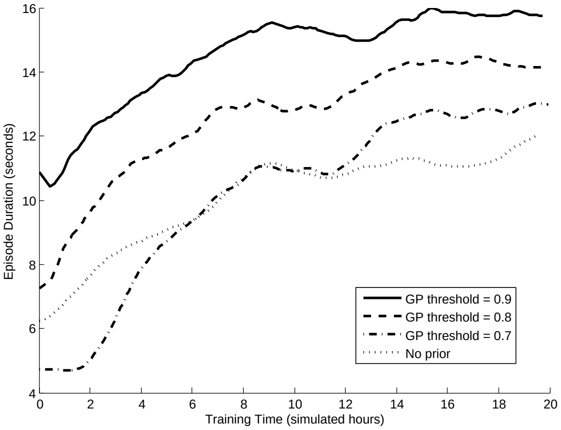

Figure 6: This figure shows the performance of same GP model with different threshold settings. Here, we show not only the start performance, but also the long-term performance. Episode duration is averaged over 10 trials. The demonstration data set contains consecutive state-action pairs of 20 episodes, with an average Episode duration of 12.4 seconds and a standard deviation of 3.56s.

ecuting the pass action more often, often resulting in worse performance than holding the ball (see Figure 6 and Table 2). But if the threshold is set too high (e.g., 0.99), the player will hardly pass the ball, which will prevent itself from reinforcement learning. In order to guarantee the effectiveness of this GP method, we need an appropriate confidence threshold. To achieve this, we could select from different thresholds according to the jumpstart performance and this parameter tuning could be realized by agent itself, trying different values.

As shown above, the confidence threshold of GPHAT remains a crucial and sensitive parameter. In this Keepaway domain, we found 0.9 to be a decent value. But it would unnecessarily be the same in other domains. The accuracy of GPHAT in Table 1 is directly affected by the confidence threshold.

Table 1 summarizes the results of the different learning methods when using four different demonstration sets. We can conclude that our novel method of using two decision trees, one for each pass action, is better than using a single decision tree. Furthermore, GPHAT outperforms decision trees, even though the classification accuracy may be lower.

Method	Jumpstart
No Prior	N/A
GP thershold = 0.9	+4.55
GP thershold = 0.8	+1.04
GP thershold = 0.7	-1.46

Table 2: This table shows the jumpstart of GPHAT with different confidence threshold.

Since double decision trees are trained to distinguish data with same action label from others (like holding the ball), while with GPHAT we only focus on the distribution of active action (passing the ball) without considering other type of data, the difference upon redundant and passive action (holding the ball) could affect the accuracy of these classifiers. So the accuracy would not necessary be a direct feature of the performance.

Conclusion and Future Work

This paper has introduced and evaluated GPHAT. Results suggest that GPHAT can successfully transfer data from one agent to another more effectively than the existing HAT method.

Because GPHAT can result in significant jumpstarts and improvement in total reward, this method could be particularly useful in domains that RL is slow to learn in.

We have also shown that integrating Gaussian process could provide a more robust learning performance despite the limited quality of demonstration, which means there are still some improvements brought by GPHAT even if the demonstration is poor.

Future work will consider how to set the confidence threshold, how to incorporate this confidence factor into action selection, and whether the confidence factor could be used to target where additional human demonstrations are needed. We are also interested using GPHAT to learn from multiple agents — we expect that the confidence of a classifier will be useful when the target agent is deciding which source agent to follow. We have also shown how, in the Keepaway domain, the actions executed are unbalanced and have unequal importance. To potentially make transfer more efficient, the demonstration data could be modified to focus on the most important data by eliminating redundant data. Future work will also aim to discover adaptive methods that could take advantage of judging the significance of demonstration data, further improving learning performance.

References

Albus, J. 1981. Brains, behavior. *& Robotics. Peterboro, NH: Byte Books* 1.

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.

Celeux, G., and Govaert, G. 1992. A classification em algorithm for clustering and two stochastic versions. *Computational statistics & Data analysis* 14(3):315–332.

Noda, I.; Matsubara, H.; Hiraki, K.; and Frank, I. 1998. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence* 12(2-3):233–250.

Quinlan, R. 1993. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Rummery, G. A., and Niranjan, M. 1994. On-line q-learning using connectionist systems.

Singh, S. P., and Sutton, R. S. 1996. Reinforcement learning with replacing eligibility traces. *Machine learning* 22(1-3):123–158.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006a. Keepaway Soccer: From Machine Learning Testbed to Benchmark. In Noda, I.; Jacoff, A.; Bredenfeld, A.; and Takahashi, Y., eds., *RoboCup-2005: Robot Soccer World Cup IX*, volume 4020. Berlin: Springer-Verlag. 93–105. 28% acceptance rate at RoboCup-2005.

Stone, P.; Kuhlmann, G.; Taylor, M. E.; and Liu, Y. 2006b. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*. Springer. 93–105.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Taylor, M. E., and Stone, P. 2007. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th international conference on Machine learning*, 879–886. ACM.

Taylor, M. E., and Stone, P. 2009. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research* 10(1):1633–1685.

Taylor, M. E.; Suay, H. B.; and Chernova, S. 2011. Integrating reinforcement learning with human demonstrations of varying ability. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 22% acceptance rate.

Witten, I. H., and Frank, E. 2005. *Data Mining: Practical Machine Learning Tools and Techniques*. San Francisco: Morgan Kaufmann, 2nd edition.