Improving Reinforcement Learning with Confidence-Based Demonstrations

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

Reinforcement learning has had many successes, but in practice it often requires significant amounts of data to learn high-performing policies. One common way to improve learning is to allow a trained (source) agent to assist a new (target) agent. The goals in this setting are to 1) improve the target agent's performance, relative to learning unaided, and 2) allow the target agent to outperform the source agent. Our approach leverages source agent demonstrations, removing any requirements on the source agent's learning algorithm or representation. The target agent then estimates the source agent's policy and improves upon it. The key contribution of this work is to show that leveraging the target agent's uncertainty in the source agent's policy can significantly improve learning in two complex simulated domains, Keepaway and Mario.

1 Introduction

Reinforcement learning [Sutton and Barto, 1998] (RL) methods have been successfully applied to both virtual and physical robots. In some complex domains, the learning speed may be too slow 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, past work on the Human Agent Transfer [Taylor *et al.*, 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. So that there are no restrictions on how the source agent learns, HAT records data from the source agent as state-action pairs. In this work the source agent could be either a human or a virtual agent, underlying how different the source and target agents can be.[1]

[1]Because of the small number of sub-optmial demonstrations from source agents, experience replay [Lin, 1992] would have limited use in complex tasks.

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 (and outperform) 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., Quinlan's J48 [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.

The key component of HAT is that it uses the learned classifier to bias its exploration. Initially, the target task agent follows the classifier, attempting to mimic the source agent. Over time, it integrates exploration and exploitation of its learned knowledge with exploiting the classifier, 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 (or source human) may be inconsistent, resulting in an inability to correctly summarize the source agent's policy. 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 incorrectly. 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, depending on these four types of potential errors.

Error types two and three, and possibly error type four, 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 by HAT, this paper shows the benefits of leveraging the measured uncertainty in the transferred information.

This work takes a critical first step in this direction by introducing CHAT (confidence-HAT), an enhancement to the HAT algorithm leveraging confidence-based demonstration. We evaluate CHAT using the domains of simulated robot soccer and Mario, empirically showing it outperforms both HAT and learning without transfer. We have three function approximators for CHAT: Gaussian process (GPHAT),

neural network (NNHAT) and decision tree (DTHAT), to show that uncertainty measurement helps. Even when low amounts of demonstration data are used, the initial performance (jumpstart) and overall performance (total reward) are significantly improved. By leveraging uncertainty in the estimate of the source agent's policy, CHAT may be useful in domains where initial performance is critical, but demonstrations from a trained agent (or human) are available but non-trivial to collect.

2 Background

This section will present some basic techniques discussed in the paper: reinforcement learning, learning from demonstration, and the HAT algorithm.

2.1 Reinforcement Learning

Reinforcement learning is a process where an agent learns through experience by exploring the environment. RL algorithms typically leverage the Markov decision process (MDP) formulation. In an MDP, A is a set of actions an agent can take and S is a set of states. There are two (initially unknown) functions within this process: a transition function ($T : S \times A \mapsto S$) and a reward function ($R : S \times A \mapsto R$).

Different RL algorithms have different ways of learning to maximize the expected reward. In this paper, we use ϵ-greedy action selection with SARSA [Rummery and Niranjan, 1994; Singh and Sutton, 1996]:

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

and Q-learning [Watkins and Dayan, 1992]:

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

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 by a vector of state variables.

2.2 Demonstration

Demonstrations are typically recorded as a vector of state-action pairs as $\langle \vec{x}, a \rangle$, in which $\vec{x}$ is the state vector (where multiple state features are composed to describe a state s) and a is the corresponding action. There are many ways of collecting this data, from visual observation to directly recording actions during teleoperation.

Learning from demonstration methods typically try to mimic this collected data. HAT differs from much of the existing work [Argall *et al.*, 2009] because its goal is to improve upon the initial demonstration data. Most relevant to this work is that of Chernova and Veloso [2009], which showed that a nearest neighbor distance metric provides a measurement of confidence in pure LfD, allowing the agent to know when to use existing demonstrations and when to request additional demonstrations from a human expert. This paper focuses on leveraging confidence measures to help RL agents select actions to improve upon demonstrated data.

2.3 Human Agent Transfer

HAT's goal is to leverage data from a source agent or source human, and then improve upon its performance with RL. HAT leverages *rule transfer* [Taylor and Stone, 2007] and the demonstrated knowledge is summarized via a decision tree. The following steps summarize HAT:

1. Learn a policy from the source task: A source agent has some policy ($\pi : S \mapsto A$) in a task, and takes actions following a policy. The state-action pairs are stored as demonstration data.
2. Train a decision tree: A decision tree is trained to summarize the state-action pairs. The decision tree is essentially a static set of rules.
3. Bootstrap the target task with the decision tree: Instead of randomly exploring, the agent will use the learned rules to guide action selection. There are three ways of using the decision tree to improve learning performance but this paper focuses on probabilistic policy reuse (PPR). In PPR, there is a parameter Φ that determines whether the learning agent should follow the classifier: the RL 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 agent to initially follow the source policy and leverage its learned Q-values over time.

3 Confidence Measurement of HAT (CHAT)

In this section we introduce improved methods (CHAT) leveraging the confidence of demonstration based on three models: Gaussian process (GPHAT), neural network (NNHAT) and decision tree (DTHAT). Once calculated, a learning agent could use this confidence in multiple ways. When our agent attempts to exploit source knowledge, it will execute the action suggested by the provided demonstration if it's confidence is above some confidence threshold. Otherwise, it will execute the "default" action (null action or random exploration). We build upon PPR, letting Φ decay. To implement CHAT, we 1) record data from a source policy, 2) train a confidence-aware classifier on this dataset, and 3) use Algorithm 1 to learn the task.

Gaussian Process (GPHAT) A Gaussian model is typically 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, we have the prediction by a classifier:

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

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

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 (for class label i):

$$C_i(x) = \exp\{-d(x, \mu_i) - \alpha_i\} \quad (1)$$

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

Algorithm 1: GPHAT: Bootstrap target learning

Input: Confidence model GP, confidence threshold T, PPR initial probability Φ_0, PPR decay $\Phi_D (= \Phi_0)$

```
Φ ← Φ_0
for each episode do
    Initialize state s to start state
    for each step of an episode do
        a ← ∅
        if rand() ≤ Φ then
            Use GP to compute C_i as shown in (1) for each action
            if max C_i ≥ T then
                a ← corresponding a_i
            else
                a ← default a_0
        else
            if rand() ≤ ε then
                a ← random action
            else
                a ← action that maximizes Q
    Execute action a
    Observe new state s' and reward r
    Update Q (SARSA, Q-Learning, etc.)
    Φ ← Φ × Φ_D
```

Neural Network (NNHAT) We use a 2-hidden-layer neural network as our confidence model. To calculate the uncertainty of demonstration, we apply softmax regression [Bishop, 2006, pp. 206–209] at output layer:

$$C_i(x) = \frac{1}{\sum_i exp(\theta_i^T \cdot x)} \begin{bmatrix} exp(\theta_1^T \cdot x)) \\ exp(\theta_2^T \cdot x)) \\ \cdots \\ exp(\theta_i^T \cdot x)) \end{bmatrix}$$

$C_i(x)$ is then used as confidence.

Decision Tree (DTHAT) We use the accuracy of each leaf node as an estimate of the confidence. Assuming the training and test data have the same distributions, we use the heuristic that the more data a node correctly classifies, the less uncertainty we expect in the node's decision. The percentage of correctly classified data of each leaf node is used as the classification confidence.

4 Experimental Setting

This section discusses the two experimental domains and our methodology.

4.1 Mario

Mario is a benchmark domain [Karakovskiy and Togelius, 2012], based on Mario Brothers. In this simulation, Mario (the learning agent) is trained to score as many points as possible. The game state is represented in a 27-tuple vector space, indicating the state and position information of Mario and his enemies [Suay *et al.*, 2016]. This vector space allows for 3.65×10^{10} different states, indicating the complexity of its learning problem. The action space for Mario is generated from these three groups: {no direction, left, right}, {don't jump/jump}, and {run/fire, don't run/ fire}. By selecting one sub-action from each of the three groups simultaneously, Mario has a total of 12 $(3 \times 2 \times 2)$ different actions.

4.2 Keepaway Simulation

Keepaway is a simulated robot soccer game. We use version 9.4.5 of the Robocup Soccer Server [Noda *et al.*, 1998], and version 0.9 of the Keepaway player framework [Stone *et al.*, 2006]. There are 3 keepers and 2 takers, playing within a bounded square. Keepers learn to keep control of the ball while takers follow hard-coded rules to chase after the ball. 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 player. We use a continuous 13-tuple vector to represent the states (e.g., position information like distances and angles). Once a keeper gets the ball, it must make a decision among three actions: *Hold*: hold the ball, *Pass*$_1$: pass the ball to the closer teammate, or *Pass*$_2$: pass the ball to the further teammate. The two keepers without the ball follow a fixed policy to try to get open for a pass. The reward is $+1$ per time step for every keeper.

4.3 Methodology

Demonstrations (state-action trajectories) are collected from a human participant via a visualizer or directly from an agent.

We first evaluate CHAT with 3 confidence models in Mario. For GPHAT, we train one-vs.-all Gaussian classifiers for each action. For NNHAT, we build a 2-hidden-layer network (50 nodes of each hidden layer).

For DTHAT, we train J48 tree with the default parameters of Weka 3.6. Second, we also evaluate GPHAT in Keepaway. We train Gaussian classifiers only on actions *Pass*$_1$ and *Pass*$_2$, as action *Hold* is executed roughly 70% of the time, making the data unbalanced. Notice that the Gaussian classifiers for *Pass*$_1$ and $Pass_2$ are one-vs.-all since it is a multiclass problem. Similarly, two one-vs.-all decision trees are trained exclusively for *Pass*$_1$ and *Pass*$_2$.

To achieve better classification accuracy, we will first use clustering to help determine the number of Gaussian classifiers in GPHAT (which can be greater than or equal to the number of actions). We cluster these data using the

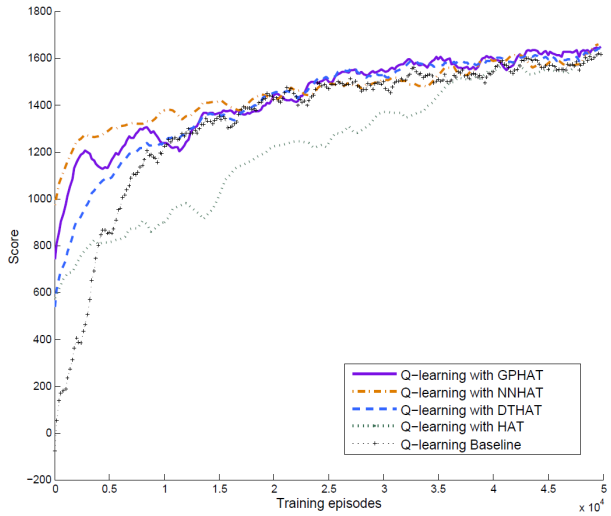

Figure 1: This figure compares the learning curves of Confidence-HAT with HAT and RL without any bootstrapping in Mario.

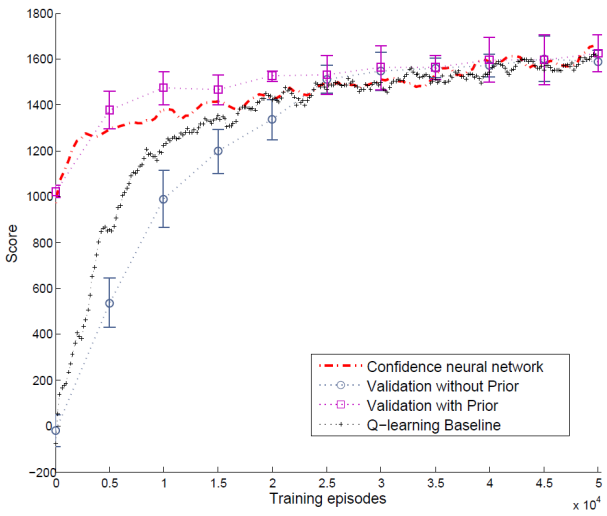

Figure 2: This figure compares validation with or without prior knowledge using confidence neural network.

Expectation-Maximization (EM) algorithm [Celeux and Govaert, 1992], with default parameter settings in Weka 3.6 [Witten and Frank, 2005], into N groups and then train N Gaussian classifiers for this class. We determine N by comparing the average performance of the first few episodes. By having several smaller data clusters, the precision of the classifier can be increased. We only use clustering for the Gaussian model.

We use SARSA in Keepaway and Q-learning in Mario to be consistent with previous work. SARSA uses: $\alpha = 0.05$, $\epsilon = 0.1$, and $\gamma = 1$. Q-learning uses: $\alpha = \frac{1}{10\times32}$, $\epsilon = 0.1$, and $\gamma = 0.9$. Notice that these parameters are consistent with previous research in these domains. The parameter Φ determines when the agent listens to prior knowledge. Φ is 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 for Keepaway and $\Phi_D = 0.9999$ to be the best for Mario (explored further in Section 5.2).

We evaluate learning performance in terms of jumpstart and total reward. Jumpstart is the average initial performance before learning. A higher jumpstart indicates that prior knowledge is more useful. The overall performance is measured by the area under a learning curve.

5 Mario Results

In this section, we show our results of learning performance by leveraging confidence in Mario domain. We also discuss and evaluate techniques that help improve CHAT. Simulation results are all averaged over 10 trials.

5.1 CHAT Outperforms HAT

In Mario, we collect demonstration data of 20 episodes (roughly 15 minutes) from a human player with an average score of 1876 points. For the benchmark of CHAT in Mario domain, we compare our algorithm with HAT and learning without any prior. Figure 1 shows the learning curves — CHAT can successfully outperform HAT and RL with no prior. In particular, the jumpstart of GPHAT, relative to HAT,

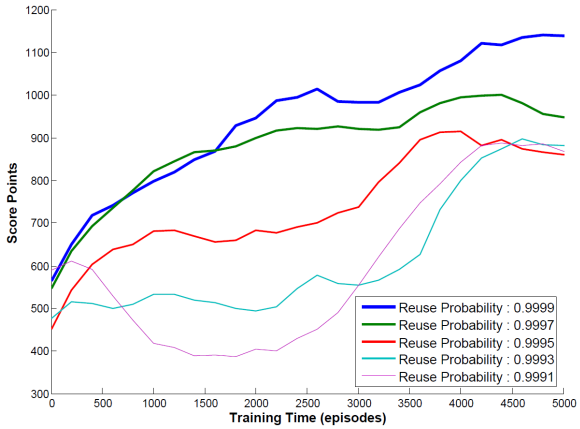

Figure 3: Different $\Phi_0 = \Phi_D$ settings with a Gaussian

was statistically significant ($p < 10^{-4}$ via t-tests). Here GPHAT uses four (on average) clusters for each action, PPR $\Phi_0 = \Phi_D = 0.9999$, and a confidence threshold of 0.8. Note that the learned performance is less than the average demonstrator for the training times considered due to the domain complexity. We next evaluate the two other confidence models (see Figure 1). The confidence threshold of neural network is 0.6 while that of the decision tree is 0.85. We again see improvement relative to HAT.

To highlight the contribution of confidence demonstration, we perform an additional validation to see how performance changes if the agent selects actions based only on its learned experience rather than prior knowledge during learning process ("without Prior" in Figure 2). This is averaged over 1000 episodes with and without prior knowledge every 5000 episodes. In Figure 2, the difference shows performance improves significantly when leveraging confidence measures.

5.2 Tuning CHAT's Reuse Probability

Taking Gaussian model (GPHAT) as an example, it uses prior knowledge with a decaying probability as mentioned before. In order to see the effect of this parameter, comparison results

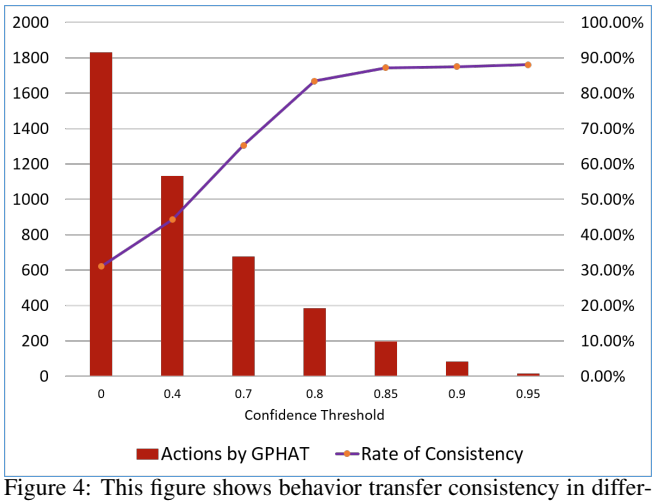

Figure 4: This figure shows behavior transfer consistency in different confidence intervals.

are plotted in Figure 3. A lower reuse factor (e.g., 0.9991) would lead to a decrease in performance shortly after the start, while a higher factor (e.g., 0.9999) would not. Notice that this does not indicate that the reuse probability should be as close to 1 as possible — once the reuse probability becomes too high, exploration will be decreased to the point that is difficult for the agent to learn to outperform the source demonstration.

5.3 Tuning CHAT's Confidence Threshold

In Mario, the Gaussian's confidence threshold T is 0.8, determined through initial parameter tuning. We now discuss how policy consistency interacts with this parameter, where the behaviors of two agents are defined as "consistent" when they select the same action for the same state.

First, we let a trained agent play Mario using its fixed policy (following its fixed Q-values) to generate 20 demonstration episodes. Second, we train GPHAT (with the same settings as above) on that demonstration. Third, we compare the actions suggested by the GPHAT classifier with actual actions made by the fixed-policy agent to see how often they are the same. Figure 4 shows how the GPHAT act with different confidence thresholds. For each confidence threshold, we show the number of actions made by GPHAT and the rate of consistency with respect to the fixed-policy agent. When the confidence threshold is too low, actions made by GPHAT are less likely to be the same as the source task agent's actions. When the confidence threshold is too high, the actions are now consistent, but very few actions will be selected.

6 Keepaway Results

This section evaluates our methods in a continuous domain, showing the benefits of our methods and investigating different types of demonstrations. Simulation results are all averaged over 10 trials.

6.1 Improvement Over Baselines

To make comparisons between different human demonstrations, we consider four different demonstrations (each with 20 episodes), their source and performance (the average episode duration and standard deviation), as summarized in Table 1.[2] The human player demonstrate three qualitatively different policies:

Table 1: This table summarizes the Keepaway demonstration datasets.

Demonstration	Source	Average Duration
Simple-Response	Human	10.5s $\pm$ 3.5s
Complex-Strategy	Human	10.1s $\pm$ 3.8s
Novice	Human	7.45s $\pm$ 2.2s
Learned-Policy	Learned Agent	10.1s

1. Simple-Response: The player holds the ball until the takers are very close to the keeper with the ball. The player only passes the ball when necessary.
2. Complex-Strategy: The player is more flexible and active in this setting. The player tries to pass the ball more often, requiring the keepers to move more. 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.
3. Novice: Consider an even worse case where we have Novice demonstration, which is only slightly better than a random policy, where many sub-optimal actions are demonstrated.

We show learning curves using the first two demonstrations in Figure 5. Calculated total rewards are in Table 3. Notice that HAT with double DTs works better than that with single DT. We therefore focus on comparing GPHAT with double-DT HAT in the remainder of this section. As expected, both sets of demonstrations allow HAT and GPHAT to outperform learning without any prior and GPHAT improves more than HAT. However, there is a significant difference in the two datasets. The Complex-Strategy data is harder to train classifiers on. This is supported by Table 2: the J48 pruned tree needed to be deeper but still had lower accuracy, indicating that the Complex-Strategy demonstration needs a decision tree with more complexity relative to the Simple-Response demonstration. Besides, we compare the robustness of GPHAT and HAT on "Novice" demonstration in Figure 5 and Table 2. GPHAT still improves learning performance upon such worse data because it can put less weight on actions that have lower confidence (i.e., instances when the classifier is less certain in the source agent's policy).

These results allow us to conclude that 1) CHAT can outperform RL agent with no bias and HAT agent for a variety of demonstration data qualities and 2) CHAT agents able to successfully outperform demonstrated policies.

6.2 Ensembles of Confidence Thresholds

Rather than tuning CHAT's confidence threshold, we also consider ensemble methods [Dietterich, 2000] made of learners with different Φ_0's. The ensemble uses majority voting, weighted by the confidence threshold of each prediction.

Voting weights used in this paper are confidence threshold of each classifier. If we considered a confidence threshold

[2]Our code and demonstration datasets are available at the first author's website: irll.eecs.wsu.edu/lab-members/zhaodong-wang/.

Table 2: This table shows comparisons among different methods. For double DTs, depth and accuracy are averaged over the two trees. For Gaussian processes, the confidence threshold is 0.9.

Demonstration	HAT (single DT)			HAT (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-Policy	+3.18	4	88.67%	+3.26	4	91.22%	**+4.55**	2	86.12%

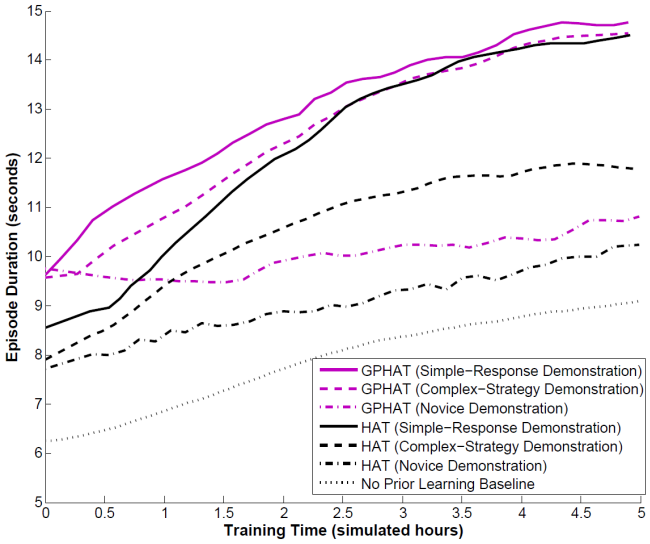

Figure 5: This figure compares the learning curves of GPHAT with HAT and RL without any bootstrapping in Keepaway.

Table 3: Total rewards of different methods in Keepaway

Method	Total Reward (5 hours)	Total Reward (20 hours)
GPHAT (Simple-Response)	76.9	290.4
GPHAT (Complex-Strategy)	75.1	283.6
HAT (Simple-Response)	72.8	270.3
HAT (Complex-Strategy)	62.7	251.6
No-Prior	47.2	219.8

of 0.7, classifiers in GPHAT with confidence higher than 0.7 vote for the final action selection, but with a scaled weight (multiplied by its threshold, 0.7 in this case). An intuitive way of understanding is that we would like those predictions with lower confidence to be considered in the final action selection, but with less significance. By doing this at different confidence thresholds, we select the final action with highest votes. Figure 6 shows the result of an ensemble of 5 confidence thresholds (from 0.5 to 0.9). GPHAT with an ensemble can outperform the best single confidence threshold (0.9), from the previous section, at the expense of additional computation (linear in the number of ensemble members).

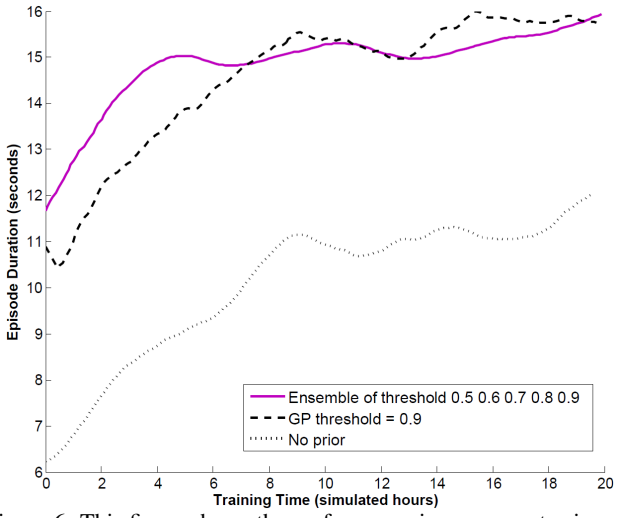

Figure 6: This figure shows the performance improvement using an ensemble of different confidence thresholds.

Results have shown that by applying different confidence models we get different learning performance and this could depend on the type/amount of human demonstrated data. In our domain, CHAT with Gaussian model could converge to the best performance, even when there is little demonstrated data. Additional results investigated how parameters in CHAT affect performance.

Having shown the potential of CHAT, future work will consider a number of extensions. First, we will investigate whether the confidence factor could be used to target where additional human demonstrations are needed. Second, we will use CHAT to learn from multiple agents — we expect that the confidence of a classifier and the demonstrated ensemble method will both be useful when the target agent is deciding which source agent to follow. Third, we have also shown how, in the Keepaway domain, the actions executed are unbalanced and have unequal importance. To make transfer more efficient, the demonstration data could be modified to focus on the most important data, eliminating redundant data. Fourth, we will investigate adaptive methods that could take advantage of judging the significance of demonstration data, further improving learning performance.

7 Conclusion and Future Work

This paper has introduced and evaluated CHAT, showing successful transfer from a human to an RL in two complex domains. CHAT outperformed both an existing method and the original demonstrations. Such improvements are most important when learning is slow or initial performance is critical.

Acknowledgements

We thank Tim Brys for sharing code for Mario. This research has taken place in part at the Intelligent Robot Learning (IRL) Lab, which is supported in part by NASA NNX16CD07C, NSF IIS-1149917, NSF IIS-1643614, and USDA 2014-67021-22174.

References

[Albus, 1981] JS Albus. Brains, behavior. *& Robotics. Peterboro, NH: Byte Books*, 1981.

[Argall *et al.*, 2009] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[Bishop, 2006] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

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

[Chernova and Veloso, 2009] Sonia Chernova and Manuela Veloso. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1):1, 2009.

[Dietterich, 2000] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

[Karakovskiy and Togelius, 2012] Sergey Karakovskiy and Julian Togelius. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.

[Lin, 1992] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[Noda *et al.*, 1998] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer server: A tool for research on multiagent systems. *Applied Artificial Intelligence*, 12(2-3):233–250, 1998.

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

[Rummery and Niranjan, 1994] Gavin A Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. 1994.

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

[Stone *et al.*, 2006] Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway soccer: From machine learning testbed to benchmark. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 93–105. Springer, 2006.

[Suay *et al.*, 2016] Halit Bener Suay, Tim Brys, Matthew E Taylor, and Sonia Chernova. Learning from demonstration for shaping through inverse reinforcement learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 429–437. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

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

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

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

[Taylor *et al.*, 2011] Matthew E. Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2011.

[Watkins and Dayan, 1992] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[Witten and Frank, 2005] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.