The Knowledge Engineering Review, Vol. 36, e2, 1 of 14.
doi:10.1017/S0269888920000387

Human–agent transfer from observations

BIKRAMJIT BANERJEE and SNEHA RACHARLA

The University of Southern Mississippi, 118 College Drive #5106, Hattiesburg, MS 39406, USA
e-mails: Bikramjit.Banerjee@usm.edu, Sneha.Racharla@usm.edu

Abstract

Learning from human demonstration (LfD), among many speedup techniques for reinforcement learning (RL), has seen many successful applications. We consider one LfD technique called *human–agent transfer* (HAT), where a model of the human demonstrator's decision function is induced via supervised learning and used as an initial bias for RL. Some recent work in LfD has investigated learning from observations only, that is, when only the demonstrator's states (and not its actions) are available to the learner. Since the demonstrator's actions are treated as labels for HAT, supervised learning becomes untenable in their absence. We adapt the idea of learning an inverse dynamics model from the data acquired by the learner's interactions with the environment and deploy it to fill in the missing actions of the demonstrator. The resulting version of HAT—called *state-only HAT (SoHAT)*—is experimentally shown to preserve some advantages of HAT in benchmark domains with both discrete and continuous actions. This paper also establishes principled modifications of an existing baseline algorithm—called A3C—to create its HAT and SoHAT variants that are used in our experiments.

1 Introduction

Reinforcement learning (RL) (Sutton & Barto, 1998) is a well-known technique for intelligent agents to learn optimal behavior (policy) from mere interactions in unknown sequential task domains. There have been many notable applications of RL over the years, including the most recent conquest of Go (Silver *et al*., 2016). A significant limitation of RL techniques is their large sample complexity—typically a large number of environment interactions are needed for the learner to reach a (near-) optimal policy. One widely applied idea for improving the learning rate is to reframe the problem as one of learning behavior from an expert demonstrator— *learning from demonstration* (LfD). The learner typically learns an inductive model of the expert demonstrator's decision function via supervised learning. This literature mostly assumes that both the demonstrator's observations and its decisions (treated as labels for supervised learning) are available to the learner. For instance, in the Mario domain (Karakovskiy & Togelius, 2012), an expert demonstrator's observations and decisions are the states and actions of the character Mario, as the expert demonstrator navigates Mario through the game, and entire trajectories of successive states and actions are provided to the learner as demonstrations.

In some prior work in LfD, the induced model of the expert demonstrator's decision function has been leveraged as an initial bias in an RL agent's action choices. This approach—called *human–agent transfer* (HAT) (Taylor *et al*., 2011)—seeks to combine the sample efficient learning of LfD with the reward seeking behavior of RL. However, this technique, like much of LfD, assumes that the expert demonstrator's observations and decisions (i.e., states and actions) are both available to the learner.

Recently, there has been some interest in extending LfD to settings where only the demonstrator's state trajectory is observed, not its actions (Liu *et al*., 2018; Torabi *et al*., 2018). For instance, demonstration videos typically only show the demonstrator's state trajectory, while its actions are either narrated

via a separate signal (audio) or are left implicit. In this paper, we are interested in studying HAT in such settings. Notice that the unavailability of the expert demonstrator's actions—labels for supervised learning—precludes induction of the initial bias for HAT. In an approach called behavioral cloning from observations (BCO), Torabi *et al*. (2018) allow the RL agent to first learn an inverse dynamics model exploiting the learner's agent specific, domain independent action model. They then use the inverse dynamics model to predict the missing actions in the expert demonstration and then use the completed demonstration in the standard behavioral cloning (BC) way. By contrast, in this paper we extend this idea to HAT without needing a separate agent-specific action model. Instead, the agent uses ordinary domain interactions to infer the inverse dynamics model, albeit sacrificing the use of the expert demonstration during those interactions. The agent then keeps updating the inverse dynamics model at a chosen interval and updating the completion of the expert demonstration, inducing a new HAT action model every time. Via experiments in benchmark domains with discrete and continuous actions, we show that this version of HAT—called *state-only HAT* (SoHAT)—can still preserve some of the advantages of HAT compared to a baseline agent. Our experiments also reveal an interesting finding about HAT (as well as SoHAT) that even though a HAT agent's online performance is superior to a baseline, its policy actually improves slower than that of the baseline. This indicates a negative impact of biased exploration on an RL agent's rate of policy improvement, even if the bias is derived from an expert demonstrator. One final contribution of this paper is a principled modification of an existing baseline algorithm (asynchronous advantage actor-critic (A3C); Mnih *et al*., 2016) to create its HAT/SoHAT variants that we have used in our experiments. This modification may be of independent interest for future work on HAT.

The significance of LfD that is devoid of the demonstrator's actions stems from the fact that there is a vast amount of data of this type that cannot be exploited by methods that require the demonstrator's actions. Torabi *et al*. (2018) cite the example of the vast array of DIY videos on YouTube where the video stream merely shows the demonstrator's states. Methods such as BCO and SoHAT are, in principle, capable of learning from such demonstrations, whereas many competing methods that require the demonstrator's actions are not. Another motivation for SoHAT arises in the context of human-focused transfer. Da Silva and Reali Costa (2019) state the following about demonstrations '*. . . AI experts might be able to design informative reward shapings or rules, while a layperson would probably only be able to give simple instructions, often in a language hard to translate to a machine. Therefore, properly leveraging knowledge from a human is not trivial . . .*' Clearly, methods that do not rely on the demonstrator's ability to articulate his/her decisions, and can learn from action-less demonstrations, are better suited for human-focused transfer. Torabi *et al*. (2018) further argue that it is also the kind of data from which human beings naturally learn by imitation. The capability to learn from state-only demonstrations brings us closer to automating this form of human learning.

To summarize our main contributions:

- We propose a new method—an extension of HAT—for LfD that includes only the demonstrator's states but not its actions. Furthermore, our method does not require an agent-specific action model to be prespecified;
- We derive a principled modification of A3C to extend HAT to actor-critic learning for continuous action domains;
- We show empirically that our method preserves some of the advantages of HAT, while requiring less data in discrete and continuous action domains;
- We observe (empirically) that both HAT and our method have an undesirable property that may prompt future research—that although these agents show improved performance compared to baseline RL, their policies improve slower than baseline RL's policy.

2 Background and related work

Here, we describe the main background topics of this paper, RL and HAT, followed by an account of the related literature.

2.1 Reinforcement learning

RL problems are modeled as *Markov decision processes* or MDPs (Sutton & Barto, 1998). An MDP is given by the tuple $\langle S, A, R, P \rangle$, where S is the set of visible environmental states that an agent can occupy at any given time, A is the set of actions from which it can select one at a given state, $R : S \times A \mapsto \mathbb{R}$ is the reward function, that is, $R(s, a)$ specifies the reward from the environment that the agent gets for executing action $a \in A$ in state $s \in S$; $P : S \times A \times S \mapsto [0, 1]$ is the state transition probability function, that is, $P(s, a, s')$ specifies the probability of the next state in the Markov chain being s' following the agent's selection of action a in state s. The agent's goal is to learn a policy $\pi : S \mapsto A$ that maximizes the sum of current and future rewards from any state s, given by,

$$V^{\pi}(s^0) = \mathbb{E}_P \left[R(s^0, \pi(s^0)) + \gamma R(s^1, \pi(s^1)) + \gamma^2 \ldots \right]$$

where $s^0, s^1, \ldots$ are successive samplings from the distribution P following the Markov chain with policy π, and $\gamma \in (0, 1)$ is a discount factor.

2.1.1 Q-learning

RL algorithms often evaluate an action-quality value function Q given by

$$Q(s, a) = R(s, a) + \max_{\pi} \gamma \sum_{s'} P(s, a, s') V^{\pi}(s').$$

This quality value stands for the sum of rewards obtained when the agent starts from state s, executes action a, and follows the optimal policy thereafter. Model free RL methods, bereft of any prior knowledge of P or R, directly learn $Q(s, a)$ by online stochastic approximation, for example, *Q-learning* (Sutton & Barto, 1998):

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right],$$

where r and s' are sampled from the unknown distributions R and P, respectively, by interacting with the environment. The final policy is calculated as

$$\pi(s) = \arg\max_{a} Q(s, a). \tag{1}$$

For domains with continuous states, $Q(s, a)$ is represented by function approximation, for example, a (deep) neural network. Such networks typically minimize a loss function given by

$$L_{\omega} = \mathbb{E} \left(r_{sa} + \gamma \max_{a'} Q_{\omega}(s', a') - Q_{\omega}(s, a) \right)^2 \tag{2}$$

where $r_{sa} \sim R(s, a)$, $s' \sim P(s, a, .)$, ω is the parameter set of the network, and the expectation is taken over the learner's trajectories.

The algorithm deep Q network (DQN) (Mnih *et al.*, 2015), used in this paper as a baseline for discrete action domains, uses the above concept even for discrete states when the state space is enormous, for example, the space of images from computer games. For stability of the network, DQN uses the concept of *experience replay* whereby past experience trajectories are maintained in a replay memory and sampled randomly to update the Q network.

2.1.2 Policy search

Policy search methods (Sutton *et al.*, 2000) explicitly maintain a policy $\pi_{\theta}(a|s)$ denoting the probability of taking action a in state s, with the distribution being parametrized by θ. In this paper, we use a policy gradient method—belonging to the class of policy search methods—where $\pi_{\theta}(a|s)$ is differentiable w.r.t θ.

One popular policy gradient technique, called advantage actor-critic (A2C), uses two function approximations. One function approximation represents the *actor*, viz. $\pi_{\theta}(a|s)$ responsible for selecting an action given a state, as stated above. The other function approximation represents the *critic*, viz., $V_{\omega}(s)$ which

gives the value of the state where the actor is (in essence critiquing the actor's performance), and is parametrized by ω. Normally, θ is improved by policy gradient, optimizing

$$J(\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta} A(s, a)$$

where $d^{\pi_\theta}(s) = \sum_{t=0}^{\infty} \gamma^t Pr(s_t = s|s_o, \pi_\theta)$ is the discounted state distribution that results from following policy π_θ, and $A(s, a)$ is called the advantage function, representing

$$A(s, a) = Q_\omega(s, a) - V_\omega(s)$$

A simple yet good estimate of the advantage function is the temporal difference (TD) error (Sutton *et al.*, 2000) given by

$$\delta_{TD}(s, a) = r_{sa} + \gamma V_\omega(s') - V_\omega(s) \tag{3}$$

where $s' \sim P(s, a, .)$. This estimate only depends on the reward and states from the actual trajectories and the critic itself. While Equation (3) is used as the loss function for updating the critic network, the actor network is updated using the gradient (Williams, 1992)

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta} \nabla_\theta \log \pi_\theta(a|s)\delta_{TD}(s, a) \tag{4}$$

We use an asynchronous version of the A2C algorithm, called A3C ((Mnih *et al.*, 2016), for experiments with continuous actions as discussed later.

2.2 *Human–agent transfer*

HAT (Taylor *et al.*, 2011) reimagines demonstration-based learning from the perspective of transfer learning (Taylor & Stone, 2009), where knowledge gained from a prior task (often referred to as *source task*) is used as inductive bias to bootstrap RL in a related but different task (often referred to as *target task*). In lieu of a related source task, HAT views the expert demonstration itself as initial bias (coming from a *source* agent) that informs the RL's action choices but gradually weans RL off this bias as the RL's own value function improves over time. In particular, the idea of HAT engages the following steps:

1. Induce a supervised classifier with the expert demonstration, to approximate the source agent's policy $\pi^E : S \mapsto A$, where E stands for the expert/source agent.
2. Bootstrap RL with the trained classifier, that is, instead of acting randomly as an RL agent would initially, use the classifier for action selection, but only with a probability, Φ, that decreases over time. Thus, an ϵ-greedy RL agent would select action as follows: select an action randomly with probability ϵ, select the classifier's predicted action with probability Φ, or select action according to Equation (1) with probability $1 - \Phi - \epsilon$.

Even though the (typically limited amount of) expert demonstration may cover a small part of the state space, with a sufficiently rich representation of states the classifier's capability of generalization can be effectively leveraged to predict what the expert demonstrator's actions might have been in unseen states. Thus, the classifier can serve as a powerful initial bias for RL. Different types of classifiers have been used in the past, for example, decision trees, neural networks, and Gaussian processes (Wang & Taylor, 2017). Typically, an initial comparative analysis of different classifiers on the available demonstration data can inform the choice of the most appropriate classifier.

The above framework was originally introduced as *probabilistic policy reuse* (Fernandez *et al.*, 2010), where Φ is called the *policy reuse parameter*. More recently, Wang and Taylor (2017) leveraged the confidence in the classifier predictions to decide whether to rely on its prediction, or on RL's value function, in a refined framework called confidence-based HAT (CHAT). However, this framework still uses the policy reuse parameter Φ, which typically starts close to 1 but decays by a constant multiplicative factor ($\Phi_D < 1$) to enable the agent to rely more on its (increasingly better) learned action value function over time. The choice of the decay factor may pose a difficult tuning problem; too rapid decay may make

RL rely prematurely on its learned Q-values, resulting sometimes in a temporary drop in performance and an associated 'valley' in the learning curve. On the other hand, if the decay in Φ is too slow, then the performance of the learner, even when mature, can be skewed in the learning curve by the classifier's own performance. To address this dilemma, Wang and Taylor (2019) introduced a new variant of CHAT—called dynamic reuse of prior—that does not require explicit decay of the reuse parameter. Since these improvements are orthogonal to the issue addressed in this paper, we use the original version of HAT for the development of our approach, although it can easily be extended to the newer variants of HAT.

2.3 Related work

The thread of literature that is most closely related to this paper is *imitation learning* (Ho & Ermon, 2016), which along with HAT and its variants falls under the broader umbrella of LfD. BC and inverse reinforcement learning (IRL) are two of the main tracks of imitation learning.

BC (Ross *et al*., 2011; Daftry *et al*., 2016) assumes that both the demonstrator's states and actions are available to the learner. Using the expert demonstration, the learner induces a classifier or regressor that predicts actions for (possibly new) states. In this respect, BC is similar to HAT. The main difference is that the replicated policy is the end product of BC subject to direct evaluation, whereas HAT's objective is to use it as an initial bias for RL and eventually learn to outperform the demonstrator. BC has been deployed in several interesting applications, for example, training an autonomous vehicle to drive in normal traffic based on camera recording (states) and steering commands (actions) of a human demonstrator (Bojarski *et al*., 2016), training a manipulator robot complex tasks using kinesthetic demonstration (Niekum *et al*., 2015), and training a quadrotor to follow a forest trail using demonstration from a human-controlled quadrotor (Giusti *et al*., 2016). These techniques typically use convolutional neural networks to facilitate deep learning from image-based demonstrations.

IRL (Russell, 1998) assumes that an MDP without the reward function is supplied, in addition to the expert demonstration. It infers this unknown reward function, for instance, as $R^* = \arg\max_R P(R|demonstration)$. This optimization is performed in such a way that if the reward function is used to complete the MDP, then solving it will yield an optimal policy which is most likely to have generated the expert demonstration. Many techniques have been developed for IRL, including Bayesian inference (Ramachandran & Amir, 2007), entropy maximization (Ziebart *et al*., 2008), and maximum likelihood estimation (VRoman, 2014), among others. Some more recent techniques relax the need for a known transition function of the MDP, thereby yielding *model-free* IRL techniques (Uchibe, 2018; Jain *et al*., 2019). Similar to BC, IRL techniques have found application in robot LfD (Argall *et al*., 2009). IRL generates not only a policy but also the demonstrator's preferences (reward function); however, since RL continues beyond the (initial) policy inference in a HAT setting, rewards are in fact available to a learner, and the inferred reward function may not be especially useful to the learner. The inferred policy, however, is usable as an initial bias in HAT.

The broader category of LfD has a rich literature spanning well over two decades. One of the earliest works by Schaal (1997) demonstrates the benefit of building a task model from the demonstrations, whereby a greater speedup is achieved compared to a model-free alternative. Kolter *et al*. (2008) also construct models, but a hierarchical one containing low level and high level MDPs, that are tuned from the demonstrations. In a similar vein, our proposal in this paper is crucially reliant on model building and refinement. Chernova and Veloso (2007) not only build Gaussian mixture models from the demonstrations but also incorporate a confidence measure to decide when to ask the demonstrator for additional demonstrations. Others in the field also investigate the informatics of querying the demonstrator, for example, Judah *et al*. (2012) empirically show the need for fewer demonstrations; Subramanian *et al*. (2016) propose a confidence measure in the context of linear models for value functions that also engender fewer queries to the demonstrator; and Walsh *et al*. (2011) establish theoretical guarantees on the required number of demonstrations while bounding the performance of the learner as well. In contrast with these approaches, we assume no additional demonstrations are available, that is, the demonstrator cannot be queried in an online manner. Recently, Tamassia *et al*. (2017) propose a method to discover subgoals from the demonstrations. The agent then autonomously learns options (temporally extended

subplans) for these subgoals, which can have a significant impact on the speed of learning, compared to an agent that does not use this insight. This method requires the knowledge of the state transition model, which is related to the inverse dynamics model that is learned via exploration in our approach. Furthermore, compared to some of the work in this category, the basis of our work—viz., HAT—seeks to improve on the performance of the demonstrator, rather than replicating the behavior of the demonstrator assumed to be an expert.

3 Problem statement

HAT and its variants, similar to other work in LfD, assume that the expert demonstrator's state and actions have been captured, based on which a classifier is induced. We consider settings where only the expert demonstrator's state is available, but its actions are private signals that are not available to the learner. This setting has been recently studied in the context of BC as *BCO* (Torabi *et al.*, 2018). There, the authors argue that human imitation is unlike most of the prior work in LfD, where human imitators typically do not have access to the internal control signals that demonstrators use to guide their behavior. They cite the example of tutorial videos on YouTube which only allow the observers to glean the demonstrator's state trajectory. Consequently, many LfD techniques cannot utilize the vast amounts of real-world demonstration data devoid of action sequences, unlike human imitators. Their approach, BCO, is able to overcome this limitation.

We ask the same question in the context of HAT: how does a learner utilize state-only expert demonstrations within the HAT framework? Clearly, the learner cannot induce a classifier that predicts actions given state, if actions are absent from the training data.

4 Our approach

We assume that the learner is supplied with a state-only expert demonstration, notated $\mathcal{T}_s = \{s_\tau^0, s_\tau^1, \ldots, s_\tau^{|\tau|-1}\}$, where $|\tau|$ is the number of observed states. Although $\mathcal{T}_s$ is assumed to be a single episode of the task, it is straightforward to extend our formulation to multiple episodes/demonstrations. In fact, we have used multi-episode expert demonstrations in our experiments. Our approach is for the learner to begin exploration just as a reinforcement learner would, while additionally accumulating information about state transitions that occur as a consequence of its own actions (this is distinct from the expert demonstration) for T episodes. The learner can utilize these data to train a model to predict which actions cause the observed state transitions, that is, an *inverse dynamics model*. The learner can then impute the missing labels (actions) in the state-only expert demonstrations, $\mathcal{T}_s$, by applying this model on the transitions observed in the expert demonstrations. The completed state-action expert demonstrations, $\mathcal{T}_{s,\bar{a}} = \{(s_\tau^0, \bar{a}_\tau^0), (s_\tau^1, \bar{a}_\tau^1), \ldots, (s_\tau^{|\tau|-1}, \bar{a}_\tau^{|\tau|-1})\}$, can now be used for HAT from episode $T+1$ onward. Instead of doing this just once, the learner can also renew the expert demonstration completions after every T episodes using the most recent observations of state transitions. This repetition is useful since as the agent learns, it may explore newer parts of the state space, which may call for a refinement of the inverse dynamics model. Further, an updated model may yield a more accurate completion of $\mathcal{T}_s$ over time, improving the utility of the expert demonstration. We found this overall idea to be quite effective.

4.1 Algorithm: SoHAT

We divide the episodes of an agent's interactions into T-episode epochs, so that the t-th epoch contains episodes in the range $\rho_t = [tT, (t+1)T - 1]$. Let s_i^j, a_i^j be the state and action of the learner in step j of episode i during its learning interactions. Based on data collected (i.e., sequences of tuples ⟨state, action, next-state⟩) in the t-th epoch, we seek a maximum likelihood inverse dynamics model $\mathcal{M}_{\theta_t^*}$, where

$$\theta_t^* = \arg\max_\theta \sum_{i \in \rho_t} \prod_{j=k}^{|\tau_i|-2} P_\theta(a_i^j | s_i^{j-k}, \ldots, s_i^j, s_i^{j+1}) \tag{5}$$

Algorithm 1 SoHAT

Require: State-only demonstration trajectory $\mathcal{T}_s$, environment *env*, RL-agent *learner*.

$\Phi \leftarrow \Phi_0$
for $t \leftarrow 0, 1, 2, \ldots$ **do**
  for $i \leftarrow tT, tT+1, \ldots, (t+1)T-1$ **do**
    $s_i^0 \leftarrow env.reset()$
    $j \leftarrow 0$
    repeat
      if (rand()$< \Phi$) $\wedge$ $(t > 0)$ **then**
        $a_i^j \leftarrow \mathcal{H}_{\phi_t^*}(s_i^j)$
      else if rand()$< \epsilon$ **then**
        $a_i^j \leftarrow$ random action
      else
        $a_i^j \leftarrow learner.getAction(s_i^j)$
      end if
      $s_i^{j+1}, r, done \leftarrow env.step(a_i^j)$
      $learner.update(s_i^j, a_i^j, s_i^{j+1}, r)$
      $j \leftarrow j+1$
    until *done*
    $\Phi \leftarrow \Phi \times \Phi_D$
  end for
  Train $\mathcal{M}_{\theta_t^*}$ optimizing θ_t^* (Equation (5))
  for $j \leftarrow k, \ldots, |\tau| - 2$ **do**
    $\bar{a}_\tau^j \leftarrow \mathcal{M}_{\theta_t^*}(s_\tau^{j-k}, \ldots, s_\tau^j, s_\tau^{j+1})$
    $\mathcal{T}_{s,\bar{a}}^j \leftarrow (s_\tau^j, \bar{a}_\tau^j)$
  end for
  Train $\mathcal{H}_{\phi_{t+1}^*}$ on $\mathcal{T}_{s,\bar{a}}$, optimizing ϕ_{t+1}^* (eqn. 6)
end for

where θ is the parameter set (e.g., weights of a neural network) representing the posterior probabilities, $|\tau_i|$ is the length of the i-th episode, and k is the length of history utilized for an action prediction. Then using the learned model, we predict the missing actions in the expert demonstrations. For step j in the expert demonstration, the predicted action, $\bar{a}_t^j$, is the maximum likelihood action from the distribution $\mathcal{M}_{\theta_t^*}(s_\tau^{j-k}, \ldots, s_\tau^j, s_\tau^{j+1})$, where s_τ are the states in the expert demonstration. Finally, we induce the maximum likelihood model $\mathcal{H}_{\phi_{t+1}^*}$, where

$$\phi_{t+1}^* = \arg\max_{\phi} \prod_{j=k}^{|\tau|-2} P_\phi(\bar{a}_t^j | s_\tau^j), \tag{6}$$

where ϕ is the parameter set of the posterior probabilities. $\mathcal{H}_{\phi_{t+1}^*}$ is used as the HAT policy in epoch $t+1$. Note that there is no HAT policy to be used in epoch $t=0$; therefore, during this epoch the agent behaves as a standard reinforcement learner. Algorithm 1 shows our approach, titled state-only HAT, or SoHAT.

4.2 HAT/SoHAT variant of A3C

To serve as a baseline for continuous action domains, we use the A3C algorithm (Mnih *et al.*, 2016), specifically its implementation from https://github.com/stefanbo92/A3C-Continuous. This implementation accommodates continuous valued, multi-dimensional actions that are modeled as Gaussians.

However, unlike DQN for discrete action domains where the value network ($Q_\omega(s, a)$) updates can work with any agent policy, the critic and the actor updates in A3C/A2C are intricately interdependent. Therefore, some modifications are required to this algorithm for its HAT and SoHAT variants, given that the learner's action policy, $\pi^L(a|s)$, is not solely due to the actor, but is a mixture of the actor and the expert:

$$\pi^L(a|s) = \Phi\pi^E(a|s) + (1 - \Phi)\pi^A_\theta(a|s)$$

where superscript A is used to denote the actor network's output. In order to use π^L in Equation (4), we note that

$$\begin{aligned}\nabla_\theta \log \pi^L &= \nabla_\theta \log(\Phi\pi^E + (1 - \Phi)\pi^A_\theta) \\ &= \frac{(1 - \Phi)\nabla_\theta \pi^A_\theta}{\pi^L} \\ &= \frac{(1 - \Phi)\pi^A_\theta \nabla_\theta \log \pi^A_\theta}{\pi^L}\end{aligned}$$

Rather than using the true/known value of Φ in the above expression, it is convenient to substitute an unbiased estimate $\mathbb{I}(a \sim \pi^E)$, since $\mathbb{E}[\mathbb{I}(a \sim \pi^E)] = \Phi$. This substitution greatly simplifies the above expression of the gradient as

$$\nabla_\theta \log \pi^L(a|s) = \begin{cases} \nabla_\theta \log \pi^A_\theta(a|s) & \text{if } \mathbb{I}(a \not\sim \pi^E(.|s)) \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The above can be substituted in Equation (4) to give an intuitively sensible gradient update rule for the actor network: update the actor network parameters in the standard A2C way *only when* the learner's action is due to the actor, but make no update if the action is recommended by the expert instead. It may appear that this discards valuable information provided by the expert's policy, but note that the other factor in Equation (4), viz. $\delta_{TD}(s, a)$, is an estimate of long-term payoffs and includes steps where the learner indeed executed the expert's actions.

Consistent with the above update logic for the actor network, we also seek the critic network updates to credit the actor only when actor's actions are taken. Thus, instead of using $\delta_{TD}(s, a)$ in the loss function for the critic network (L_ω), we estimate the portion of it for which the actor is responsible, $(1 - \Phi)\pi^A_\theta(a|s)\delta_{TD}(s, a)$. Once again, this is estimated as

$$[1 - \mathbb{I}(a \sim \pi^E(.|s))]\pi^A_\theta(a|s)\delta_{TD}(s, a),$$

which gives the modified step loss function:

$$L_\omega(s, a) = \begin{cases} \left(\pi^A_\theta(a|s)\delta_{TD}(s, a)\right)^2 & \text{if } \mathbb{I}(a \not\sim \pi^E(.|s)) \\ 0 & \text{otherwise} \end{cases}. \tag{8}$$

From this, we get the loss function $L_\omega = \mathbb{E}[L_\omega(s, a)]$ with the expectation taken over the learner's trajectories as in Equation (2) before.

Equations (7) and (8) give the modifications to A3C required for its HAT variant. These changes can also be easily incorporated into step 15 of Algorithm 1 for the SoHAT variant of A3C.

4.3 Scalability

The time complexity of the proposed method is greater than HAT, due to the presence of additional lines 20–25 in Algorithm 1 that are executed $\frac{\text{total \#episodes}}{T}$ times, that is, the number of epochs. In comparison, HAT only executes line 25 *once* and does not need lines 20–24 because the demonstrator's actions are available to it. The added complexity of SoHAT is justified by the fact that it is solving a harder problem than HAT. Note, in particular, that the measures of complexity of the underlying MDP, viz., the sizes of the state and action spaces, do not directly impact the complexity of the additional lines. Rather, the

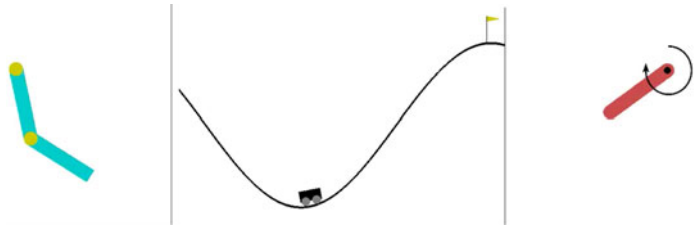

Figure 1 Screenshots from experimental domains. Left to right: Acrobot, MountainCar, and Pendulum. See description in Section 5

latter is dependent on the size of the demonstration, as well as the capacities of the learned models $\mathcal{M}_{\theta^*}$ and $\mathcal{H}_{\phi^*}$. We assume that these capacities are sufficient to accommodate the state and action spaces of the underlying MDP, as well as the chosen length of history (i.e., k). The above observations specifically apply to the HAT/SoHAT derivative of DQN, but the rough idea holds even in the context of the continuous action baseline (i.e., A3C) of HAT/SoHAT.

5 Experimental setting

We envision that the proposed methods can be applied to real-world data from a variety of projects, ranging from humans working in a manufacturing plant (for autonomous training of their robotic replacements) to academic course tutorials (for creating automated tutors). However, such complex tasks also pose engineering challenges, for instance, processing video data requires the engineering of appropriate deep pipelines, to discover salient characteristics of the data without human intervention. Since this aspect is orthogonal to the current study, in this paper we focus on a comparative evaluation of our proposed techniques in much simpler tasks, in order to elicit their key strengths and limitations in controlled settings, to inform the future evolution of related techniques.

We conducted experiments in domains taken from OpenAI's gym, with both discrete actions and continuous actions. The discrete action domains are Acrobot and MountainCar. In Acrobot (see screenshot in Figure 1 left), the task is for an Acrobot—formed by two linked pieces suspended at the top—to swing the bottom end up to a specific height. In MountainCar (see screenshot in Figure 1 middle), the task is for a car at the bottom of a valley to acquire enough momentum via back and forth motion to reach the target location at the top right (shown as a flag). The domains with continuous actions are Pendulum and MountainCar-continuous. In Pendulum (see screenshot in Figure 1 right), the task is to swing up a Pendulum and make it stay upright as long as possible. The MountainCar-continuous domain is essentially the same problem as the discrete MountainCar, except that the action of the car is now continuous (forward/backward throttle velocity), instead of binary forward/backward throttle direction. We modified MountainCar-continuous to use a constant penalty of -0.02 per step instead of the gym-implementation's variable penalty that depends on the action's magnitude. This modification simplifies the learning problem, allowing experiments with a short horizon. Below we discuss the implementation details for the two kinds of environments, followed by the experimental results. Table 1 summarizes the size and types of domain features. In all of our experiments, we set the value of k to 0 since the chosen domains are Markovian with respect to the state observations. However, for partially observable domains, $k > 0$ normally yields higher quality policies. Also, in all of our experiments we set the Φ decay parameter (Φ_D in Algorithm 1) to be such that the value of Φ decays to 0.05 by the end of the learning trial. This makes the learned policy the dominant factor (responsible for 95% of action choices) behind the agent's performance at the end, thereby allowing us to evaluate its quality.

5.1 Discrete actions

For discrete action domains, we chose DQN (Mnih *et al.*, 2015) from OpenAI baselines with default parameters as our baseline algorithm. While the neural network for MountainCar had one hidden layer

Table 1 Summary of domain sizes

Domain	States	Actions	# Demo episodes
MountainCar	Box (2)	Discrete (3)	10
Acrobot	Box (6)	Discrete (3)	10
Pendulum	Box (3)	Box (1)	10
MountainCar-cont	Box (2)	Box (1)	10

Table 2 Summary of F-measure values, averaged over all classes

Domain	Decision tree	Neural network	Random forest
MountainCar	0.971	0.973	0.975
Acrobot	0.930	0.932	0.941

with 64 units, the network for Acrobot had two hidden layers with 64 units each. These architectures stayed consistent across the HAT and SoHAT variants as well. To create the HAT version of DQN, we paired it with a *random forest* classifier from Weka 3.8.1, for $\mathcal{H}_{\phi^*}$. Cruz *et al*. (2017) discuss a more direct way to implement HAT version of DQN by bootstrapping the network weights from a prior trained neural network. By contrast, we chose to keep the classifier separate from DQN's value function network, to facilitate a fair comparison with SoHAT which requires repeated retraining on completed state-only expert demonstrations. This choice helps to avoid repeated bootstrapping of the network weights at regular intervals, which might destabilize learning. In order to train the inverse dynamics model, that is, $\mathcal{M}_{\theta^*}$, we again used a random forest classifier from Weka. The choice of random forest was informed by preliminary experiments with complete expert demonstrations, due to its superior F-measure score over 10-fold cross-validation (using default settings in Weka), as shown in Table 2.

5.2 Continuous actions

For continuous action domains, as mentioned before, we used the A3C-continuous algorithm as our baseline, with its modifications for HAT and SoHAT as described in Section 4.2. We trained a neural network for the inverse dynamics model, $\mathcal{M}_{\theta^*}$, with two hidden layers, 600 and 300 nodes, and rectified linear activation. We used the Adam optimizer (Kingma & Ba, 2015) (with learning rate 2^{-12}) to minimize the mean square error. Similarly, for the HAT action selection, again we trained a neural network model for $\mathcal{H}_{\phi^*}$, with three hidden layers, containing 600, 300, and 60 nodes, and rectified linear activation. Adam optimizer was used to minimize the mean square error in this network as well.

6 Experimental results

We performed two separate sets of experiments for both discrete and continuous action settings. In the first experiment, we compared the *online* performances of the baseline, HAT and SoHAT learners. The results are shown in Figures 2 and 3. Furthermore, we investigated the errors in action predictions from the inverse dynamics model of SoHAT. For discrete action domains, the error plots show the variation of $\sum_j \mathbb{I}(\mathcal{T}^j_{s,\bar{a}} \neq \mathcal{T}^j_{s,a*})/|\tau|$, where $a*$ are the true expert demonstrator actions available to HAT, with $\mathcal{M}_{\theta^*_t}$ for various epochs t. For continuous action domains, the error is based on relative absolute differences instead of the indicator function. The results are shown in Figures 4 and 5.

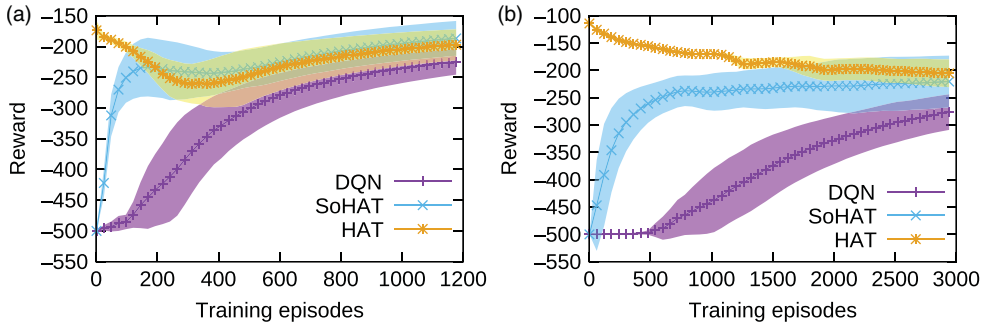

Figure 2 Comparative learning plots for baseline (DQN), HAT, and SoHAT in discrete action domains. (a) Acrobot. (b) MountainCar

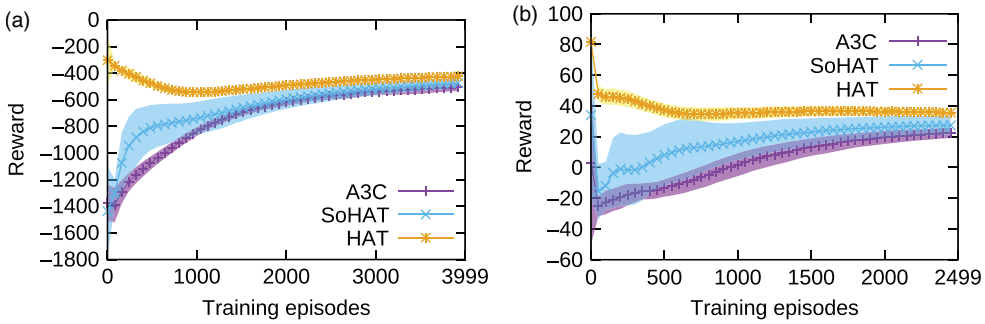

Figure 3 Comparative learning plots for baseline (A3C), HAT, and SoHAT in continuous action domains. (a) Pendulum. (b) MountainCar-continuous

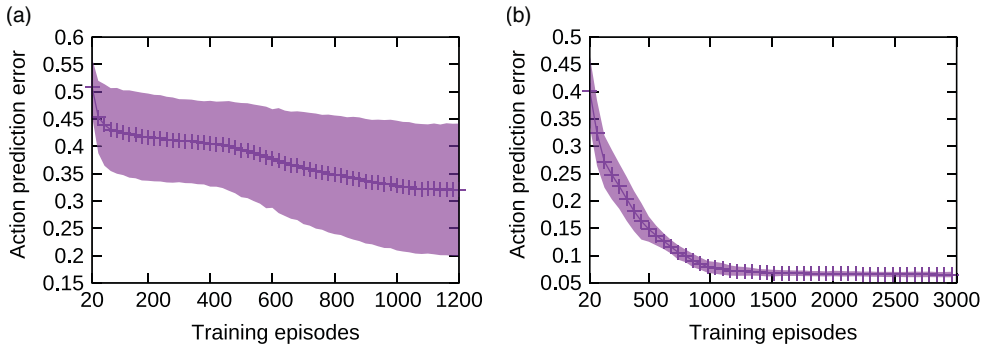

Figure 4 Error plots for predicted completions of state-only demonstrations for SoHAT in discrete action domains. (a) Acrobot. (b) MountainCar

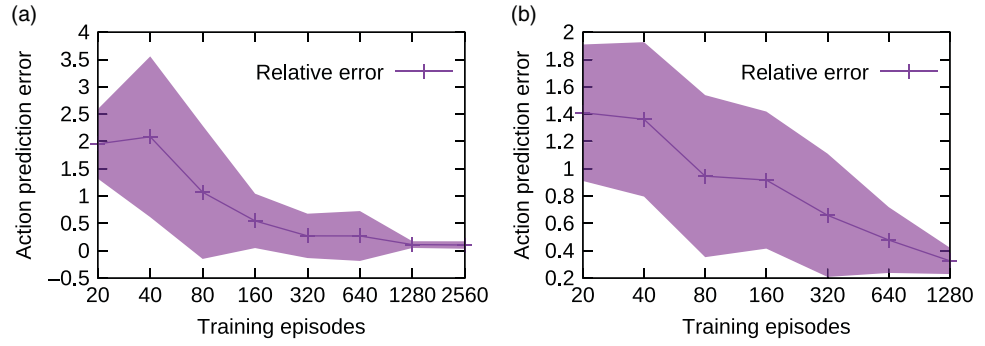

Figure 5 Error plots for predicted completions of state-only demonstrations for SoHAT in continuous action domains. (a) Pendulum. (b) MountainCar-continuous

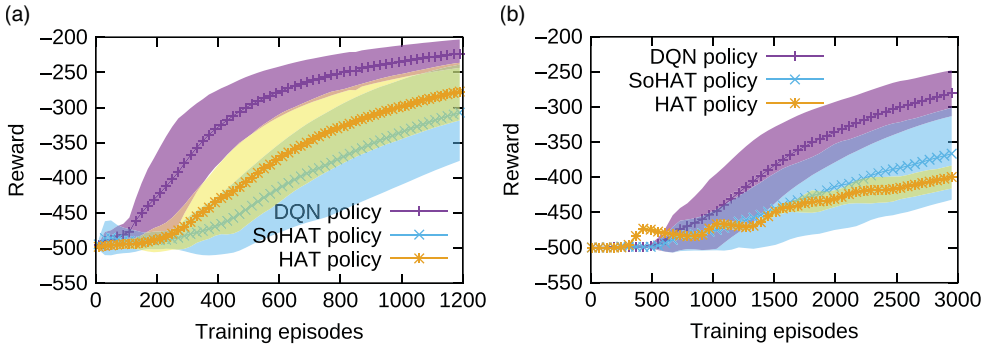

Figure 6 Performance of learned policies for baseline (DQN), HAT, and SoHAT in discrete action domains. (a) Acrobot. (b) MountainCar

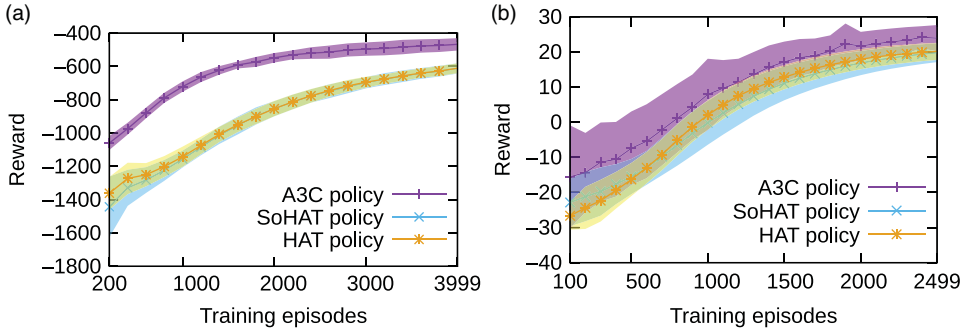

Figure 7 Performance of learned policies for baseline (A3C), HAT, and SoHAT in continuous action domains. (a) Pendulum. (b) MountainCar-continuous

In the second experiment, we investigated the consistency of learning from expert demonstration. We saved the intermediate learned policies of the baseline, HAT, and SoHAT agents at some regular interval. Later, we evaluated those policies in the respective domains, each over 1000 episode simulations. The results are shown in Figures 6 and 7.

Figures 2 and 3 are mostly consistent with previous findings regarding HAT that it affords a jumpstart (high initial performance) and often leads to higher convergence value for the learner as well, compared to the baseline and compared to the human demonstrator. In our case, the human demonstrator's performance was the same as the baseline's final performance, because the expert demonstrations were generated by the baseline's final learned policy. Our main finding is that SoHAT indeed improves over the baseline, despite not observing the expert demonstrator's actions (unlike HAT). Further, the decreasing errors in Figures 4 and 5 show the benefit of regenerating $\mathcal{H}_{\phi^*}$ at some intervals using the most recent experience of the learner, instead of generating it just once in epoch 1. Since a neural network model of $\mathcal{H}_{\phi^*}$ (used for continuous action domains) is slow to train, we reduced the frequency of $\mathcal{H}_{\phi^*}$ regeneration for the continuous action domains by doubling T after line 25 of Algorithm 1, note the log scale of x-axis in Figure 5.

Figures 6 and 7 reveal an interesting finding about HAT. Although the HAT learner has a superior online performance compared to baseline, as seen in Figures 2 and 3, its policy improves *slower* than the baseline. This is true in both discrete and continuous action domains, including a significant difference in Pendulum. However, note that SoHAT's policy improvement rate is still comparable to, and not significantly different from, HAT.

7 Conclusion and future work

In this paper, we have investigated the application of HAT to scenarios where only the expert demonstrator's states, and not its actions, are available to the learner. We have proposed a new

algorithm—SoHAT—that extends some of the advantages of HAT to such scenarios. In particular, experiments in both discrete and continuous action domains have shown that a SoHAT learner can exhibit faster learning rates, and in some cases improved asymptotic performance, compared to the baseline. A HAT learner shows superior jumpstart, but that is only due to the fact that it has access to the expert demonstrator's true actions, which SoHAT does not have.

We have also proposed a modification of the well-known A3C algorithm for its HAT and SoHAT variants and used it for experimental studies. This modification is an original contribution in itself and can play an important role in future studies of HAT where the learner and the expert's separate performances might be of interest, precluding an existing alternative where the learner's weights are pre-trained from the expert network (de la Cruz *et al*., 2017).

Our experiments have also revealed a previously unknown, yet perhaps unsurprising, limitation of HAT (and SoHAT as well). Although a HAT learner's online performance appears superior to a baseline's, its policy actually improves slower than the baseline's. This is not surprising because the frequent (initial) usage of the expert's policy only improves the learner's values for a small set of states while the learner's policy may remain oblivious of what to do in the other states that are not visited due to the expert-biased lack of exploration. In other words, the deployment of the expert's knowledge negatively impacts the learner's exploration, at least initially. While this finding may spur additional future work on HAT to strike an improved balance of exploitation versus exploration, the main lesson that we draw in this paper in the context of our proposed algorithm SoHAT is that this impact is not significantly different than what we see in case of HAT. Thus, we posit SoHAT as a more practical alternative to HAT in real-world scenarios where a learner must rely on mere observations—and not the internal signals—of the expert demonstrator.

Acknowledgement

We are grateful to the anonymous reviewers for helpful comments and suggestions. This work was supported in part by a National Science Foundation grant IIS-1526813.

References

Argall, B. D., Chernova, S., Veloso, M. & Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems* **57**(5), 469–483.

Bojarski, M., Testa, D., *et al.* 2016. End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

Chernova, S. & Veloso, M. 2007. Confidence-based policy learning from demonstration using Gaussian mixture models. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, **233**, 1–8. ACM.

Daftry, S., Bagnell, J. & Hebert, M. 2016. Learning transferable policies for monocular reactive MAV control. In *Proceedings of the International Symposium on Experimental Robotics*, 3–11.

Da Silva, F. L. & Reali Costa, A. H. 2019. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research* **64**, 645–703.

de la Cruz Jr, G. V., Du, Y. & Taylor, M. E. 2017. Pre-training Neural Networks with Human Demonstrations for Deep Reinforcement Learning. arXiv preprint arXiv:1709.04083.

Fernandez, F., Garcia, J. & Veloso, M. 2010. Probabilistic policy reuse for inter-task transfer learning. *Robotics and Autonomous Systems* **58**(7), 866–871.

Giusti, A., Guzzi, J., *et al.* 2016. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters* **1**(2), 661–667.

Ho, J. & Ermon, S. 2016. Generative adversarial imitation learning. *In Advances in Neural Information Processing Systems*, 4565–4573.

Jain, V., Doshi, P. & Banerjee, B. 2019. Model-free IRL using maximum likelihood estimation. *In Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI-19)*, Honolulu, HI, 3951–3958.

Judah, K., Fern, A. & Dietterich, T. G. 2012. Active imitation learning via reduction to I.I.D. active learning. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence (UAI)*, 428–437.

Karakovskiy, S. & Togelius, J. 2012. The Mario AI benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games* **4**(1), 55–67.

Kingma, D. P. & Ba, J. 2015. Adam: a method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*.

Kolter, J. Z., Abbeel, P. & Ng, A. Y. 2008. Hierarchical apprenticeship learning with application to quadruped locomotion. In *Advances in Neural Information Processing Systems (NIPS)*, 769–776.

Liu, Y., Gupta, A., Abbeel, P. & Levine, S. 2018. Imitation from observation: learning to imitate behaviors from raw video via context translation. In *Proceedings of the International Conference on Robotics and Automation (ICRA-18)*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. & Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, Proceedings of Machine Learning Research **48**, PMLR, New York, New York, USA, Balcan, M. F. and Weinberger, K. Q. (eds), 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. & Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533.

Niekum, S., Osentoski, S., Konidaris, G., Chitta, S., Marthi, B. & Barto, A. G. 2015. Learning grounded finite-state representations from unstructured demonstrations. *International Journal of Robotics Research* **34**(2), 131–157.

Ramachandran, D. & Amir, E. 2007. Bayesian inverse reinforcement learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2586–2591.

Ross, S., Gordon, G. & Bagnell, J. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTAT)*, 627–635.

Russell, S. 1998. Learning agents for uncertain environments (extended abstract). In *Eleventh Annual Conference on Computational Learning Theory*, 101–103.

Schaal, S. 1997. Learning from demonstration. In *Advances in Neural Information Processing Systems (NIPS)*, 1040–1046.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489.

Subramanian, K., Isbell Jr, C. L. & Thomaz, A. L. 2016. Exploration from demonstration for interactive reinforcement learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 447–456.

Sutton, R. & Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., McAllester, D., Singh, S. & Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*. MIT Press, 1057–1063.

Tamassia, M., Zambetta, F., Raffe, W., Mueller, F. & Li, X. 2017. Learning options from demonstrations: A Pac-Man case study. *IEEE Transactions on Computational Intelligence and AI in Games* **10**(1), 91–96.

Taylor, M. E. & 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. & 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)*.

Torabi, F., Warnell, G. & Stone, P. 2018. Behavioral cloning from observation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, 4950–4957.

Uchibe, E. 2018. Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters* **47**(3), 891–905.

VRoman, M. C. 2014. *Maximum Likelihood Inverse Reinforcement Learning*. PhD thesis, Rutgers University.

Walsh, T. J., Hewlett, D. K. & Morrison, C. T. 2011. Blending autonomous exploration and apprenticeship learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2258–2266.

Wang, Z. & Taylor, M. E. 2017. Improving reinforcement learning with confidence-based demonstrations. In *Proceedings of the 26th International Conference on Artificial Intelligence (IJCAI)*.

Wang, Z. & Taylor, M. E. 2019. Interactive reinforcement learning with dynamic reuse of prior knowledge from human and agent demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI-19)*, 3820–3827.

Williams, R. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* **8**(3–4), 229–256.

Ziebart, B. D., Maas, A., Bagnell, J. A. & Dey, A. K. 2008. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 1433–1438.