Automatic State Abstraction from Demonstration*

Luis C. Cobo
luisca@gatech.edu
College of Engineering
Georgia Tech
Atlanta, GA 30332

Peng Zang
pengzang@cc.gatech.edu
College of Computing
Georgia Tech
Atlanta, GA 30332

Charles L. Isbell Jr.
isbell@cc.gatech.edu
College of Computing
Georgia Tech
Atlanta, GA 30332

Andrea L. Thomaz
athomaz@cc.gatech.edu
College of Computing
Georgia Tech
Atlanta, GA 30332

Abstract

Learning from Demonstration (LfD) is a popular technique for building decision-making agents from human help. Traditional LfD methods use demonstrations as training examples for supervised learning, but complex tasks can require more examples than is practical to obtain. We present Abstraction from Demonstration (AfD), a novel form of LfD that uses demonstrations to infer state abstractions and reinforcement learning (RL) methods in those abstract state spaces to build a policy. Empirical results show that AfD is greater than an order of magnitude more sample efficient than just using demonstrations as training examples, and exponentially faster than RL alone.

1 Introduction

Our research goals center around deploying agents in real-world human environments, *e.g.*, robotic assistants in homes and hospitals, or non-player characters in video games. As it is impractical to pre-program an agent with every skill it might need in these domains, we aim for flexible agents that can learn new tasks and skills from *non-expert* end-users.

Learning from Demonstration (LfD) is a natural choice for achieving such flexibility, having been applied successfully in domains ranging from helicopter flying [Ng *et al.*, 2004] to simulated robosoccer [Aler *et al.*, 2005]. LfD is a form of supervised learning that uses demonstrations to approximate a mapping from states to actions [Argall *et al.*, 2009]. Unfortunately, the possibly large number of examples needed to learn a mapping may be a burden on the human. Further, training and test distributions may not match when the agent has not perfectly mimicked the demonstrator: a policy only partially replicating human behavior may lead to unknown parts of the state space. Thus, small errors can produce large differences in the distribution of states that the agent will encounter.

We present an alternative approach, Abstraction from Demonstration (AfD), a combination of demonstration and traditional reinforcement learning (RL) approaches. AfD avoids these problem and dramatically reduces the number of demonstrations needed to obtain good performance. In AfD, we use demonstration data not to learn the underlying mapping, but to identify relevant task features. We then apply RL methods with this abstract state space to learn a policy.

AfD lies between LfD approaches that rely primarily on demonstrations to build a good policy, and RL methods that rely primarily on a reward signal. Compared to LfD, AfD supplements human examples with information from the reward to build policies better than those demonstrated. Compared to RL, AfD supplies abstraction information from human data to reduce the complexity of the problem.

AfD exploits the fact that the set of features necessary to learn and represent a reasonable policy may be much smaller than the full set of features. It might seem more straightforward to ask human subjects directly for features; however, such an approach requires interfaces for presenting and querying domain features and requires users to understand the features and their ramifications. Demonstrations on the other hand, let us extract information about what features are relevant implicitly through users' actions.

In this paper we present our algorithm for using non-expert human demonstrations to identify relevant features. We present experiments showing that resulting policies are significantly better than what can be learned directly from the demonstrations or via RL in the full state space. Finally, we discuss the implications and applicability of our approach.

1.1 Related work

There are numerous extensions to LfD [Argall *et al.*, 2009]. A significant branch of LfD focuses on combining demonstrations with traditional RL methods, including using demonstration to guide exploration [Smart and Kaelbling, 2002], learn a reward function [Abbeel and Ng, 2004], and induce task decompositions [N. Mehta *et al.*, 2007; Zang *et al.*, 2009]. AfD uses demonstrations to learn state abstractions, making the subsequent application of RL dramatically faster. This is most similar to inducing task decompositions but through a different mechanism. As such, when a domain contains both potential abstractions and hierarchical decompositions, these techniques are complementary.

Another branch of LfD uses demonstration data for learning plans, *e.g.*, learning pre and post conditions; however, this work typically assumes additional information such as annotations. We limit our attention to cases where only demonstra-

*This work is supported by the National Science Foundation under Grant No. 0812116 and Obra Social "la Caixa".

tions are available, reflecting our focus on learning from non-experts. Requiring additional information introduces complexity and moves us away from our target teacher pool.

Prior work has also focused on automated methods of performing feature selection in RL such as the use of L_1 regularization [Kolter and Ng, 2009], L_2 regularization [Farahmand *et al.*, 2009], and "forward-selection" style feature selection from features generated based on Bellman error analysis [Keller *et al.*, 2006; Parr *et al.*, 2007]. Our work differs in that we are selecting features for representing the human policy, as opposed to an optimal or near-optimal Q or value function. The features we select from human data are typically inadequate for representing an accurate Q or value function for any given (*e.g.,* linear) model. We use an exact (tabular) Q-representation in the abstract state space.

Finally, much prior work in LfD learns from researchers. In our work we focus on using everyday people.

2 Preliminaries: Reinforcement Learning

A Markov Decision Process (MDP) is a tuple $M = (S, A, P^a_{ss'}, R^a_s, \gamma)$ where $S = \{F_1 \times \cdots \times F_n\}$ is a finite set of states with n features, A a finite set of actions, $P^a_{ss'} = Pr(s'|s,a)$ the transition model that specifies the probability of reaching state s' when taking action a in state s, $R^a_s = r(s,a)$ the immediate reward when taking action a on state s, and $0 \leq \gamma \leq 1$ the discount factor. We define a state $s \in S$ as an n-tuple of features $s = (f_1, f_2, \ldots, f_n)$.

A *policy* $\pi : S \rightarrow A$ defines which action an agent takes in a particular state s. $V^\pi(s) = R^{\pi(s)}_s + \gamma \sum_{s' \in S} P^{\pi(s)}_{ss'} V^\pi(s')$ is the state value of s when following policy π, *i.e.,* , the expected sum of discounted rewards that the agent obtains when starting on state s and following policy π. $Q^\pi(s,a) = R^a_s + \gamma \sum_{s' \in S} P^a_{ss'} V^\pi(s')$ is the Q-value of (s,a), or the discounted reward received when choosing action a in state s, and then following policy π. π^* is the optimal policy maximizing the value of each state. $V(s) = V^{\pi^*}(s)$, $Q(a,s) = Q^{\pi^*}(a,s)$ are the optimal state and Q-values, and sometimes are referred to simply as *the* values of the state.

A human demonstration, or *user trace*, is a set of k pairs of tuples $\{(s_1, a_1), \cdots, (s_k, a_k)\}$. Each pair is a *sample* that describe which action the human teacher took in state s_i.

3 Abstraction from Demonstration

3.1 Algorithm

AfD learns a policy for an MDP by building an abstract space S^α and using reinforcement learning to find an optimal policy that can be represented in S^α. AfD obtains S^α by selecting a subset of features from the original state space S with which it can predict the action that a human teacher has taken in the set of demonstrations. Learning in S^α can be significantly more efficient because a linear reduction in the features leads to an exponential reduction in the size of the state space.

We present our algorithm for AfD in Listing 1. It is composed of two sequential steps. First, a feature selection algorithm is applied on human demonstrations to choose the subset of features we will use. In the second step, that corresponds with the loop in Listing 1, the algorithm learns a policy for M using a modified version of a Monte Carlo learning algorithm with exploring starts. Instead of learning the Q-values of states in the original state space $Q(s,a), s \in S$, it learns the Q-values of states in the transformed state space $Q(s^\alpha, a)$ where $s^\alpha \in S^\alpha$. We stop when policy performance converges, *i.e.,* , when the expected return remains stable.

Algorithm 1 Generic AfD algorithm with $\gamma = 1$, the general case is a simple extension.

Require: MDP $M = (S, A, P^a_{ss^\alpha}, R^a_s, \gamma)$, $S = \{F_1 \times \ldots \times F_n\}$, feature selector F, human demonstrations $H = \{\{(s_1, a_1), (s_2, a_2), \ldots\}, \ldots\}, s \in S, a \in A$.
  chosenFeatures ← F(H)
  $\pi \leftarrow$ arbitrary policy
  Initialize all $Q(s^\alpha, a)$ to arbitrary values
  Intialize all Rewards[(s^α, a)] to []
  while π performance has not converged **do**
    $visited \leftarrow []$
    Start episode with random state-action, then follow π
    for all Step (state s, action a, reward r) **do**
      for all (s^α, a) in $visited$ **do**
        EpisodeReward[(s^α, a)] += r
      end for
      $s^\alpha \leftarrow$ getFeatureSubset(s, chosenFeatures)
      if (s^α, a) not in $visited$ **then**
        Add (s^α, a) to $visited$
        EpisodeReward[(s^α, a)] $\leftarrow 0$
      end if
    end for
    for all (s^α, a) in $visited$ **do**
      Add EpisodeReward[(s^α, a)] to Rewards[(s^α, a)]
      Q(s^α, a)=average(Rewards[(s^α, a)])
    end for
    Update greedy policy π
  end while

We have used two different feature selection algorithms for AfD. The first, which we call C4.5-greedy, is a simple greedy backward selection algorithm. Starting with the full feature set, it iteratively removes features, one at a time, that have little impact on the performance. In each iteration, the feature whose absence affects accuracy the least is removed. If the best feature to drop affects accuracy by more than 2% with respect to the current feature set, we stop. We also stop if dropping a feature results in an accuracy drop greater than 10% with respect to the original feature set. These stopping parameters are not sensitive. In experiments we have tested parameter values of 1% and 5% with no significant difference in the feature set selected. Note that we use *relative accuracy* for these stopping criterion, *i.e.,* , the amount of accuracy gained with respect to the majority classifier.

The second approach, Cfs+voting, uses Correlation-based Feature Subset Selection (Cfs) [Hall, 1999]. Cfs searches for features with high individual predictive ability but low mutual redundancy. The Cfs algorithm is used separately on the demonstrations of each individual. A feature is chosen if it is included by at least half of the individuals.

3.2 Policy Invariance

State abstractions that, like ours, collapse states with the same optimal action are called policy-invariant. These present the

problem that while they are enough to represent a policy, they might not be good enough to learn it.

In the transformed state space, a single state $s^\alpha \in S^\alpha$ represents several states of the original space $s_1, \ldots, s_k \in S$. Assuming the subset of features selected is sufficient for predicting the demonstrator's action, every original state corresponding to s^α should share the same action. Because only values with the same associated action are collapsed, the teacher policy can be represented in S^α. The original state value function $V(s)$ may not be representable in S^α because the collapsed states $s_1, \ldots, s_k$ may have different true state values in S. The same holds for the Q-values $Q(s, a)$.

For example, Fig. 1 shows a deterministic MDP with states defined by features $\{F_1, \cdots, F_n\}$. F_1 can have values U or D depending whether the state is up or down in the figure, and the rest of the features are policy-irrelevant. The available actions are a_1 and a_2 and the only reward is obtained by taking action a_1 in state s_2. With $\gamma < 1$, the value of s_1 and s_2 is different; however, because their optimal action is the same (a_1), AfD collapses both into a single state $s^\alpha_{a_1} \in S^\alpha$.

Conversion to the transformed space also breaks the Markovian property. In Fig. 1, we can see that *wrt.* the transformed space, actions a_1 and action a_2 in collapsed state $s^\alpha_{a_2}$ are identical. However, we know that in reality the two actions are very different. The value of taking action a_1 from collapsed state $s^\alpha_{a_1}$ depends upon how we got to $s^\alpha_{a_1}$; specifically, what action was taken in state $s^\alpha_{a_2}$ to get there. This means reinforcement learning algorithms that make use of bootstrapping will not work. Bootstrapping algorithms such as Q-learning compute Q-values considering the immediate reward and the value of the next state. In our example, with the immediate reward and next state identical for both actions, Q-learning would be unable to learn the best action, even though one leads to higher discounted rewards.

Previous work on policy-invariant abstractions, such as [Jong and Stone, 2005], encapsulate the state abstraction as an option that will not be used if it degrades performance. We work around problems of policy-invariant abstractions by using non-bootstrapping methods like Monte Carlo control.

3.3 Properties

AfD's feature elimination process is benign: in the limit of infinite data, the feature subset it yields will not negatively affect the accuracy of the learner. To see this, consider that AfD only removes features it judges will not reduce the accuracy of the learner. These judgements are based on some held out portion of the data.[1] In the limit of infinite data, these judgements will be accurate, and the elimination process will never remove a feature if it negatively impacts accuracy. Thus the final feature subset cannot negatively impact accuracy.

AfD's policy solver is sound. As shown in Sec. 3.2, in a policy-invariant abstraction like ours, bootstrapping algorithms like Q-learning may not find the best stationary policy; however, the abstracted space creates a POMDP, where Monte Carlo control is sound [Singh *et al.*, 1994]. Thus, applying the policy solver will not lower policy performance.

[1] We actually use cross validation for higher sample efficiency, but the same principle holds.

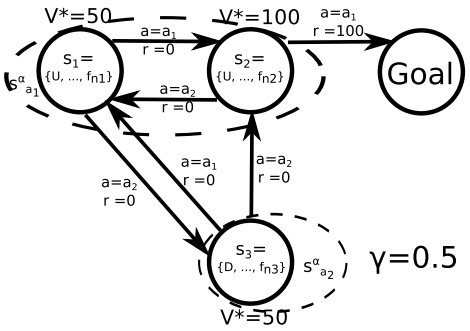

Figure 1: Example of a MDP that loses the Markov property on state abstraction. Circles represent states in S and arrows transitions from taking a_1, providing an immediate reward r. Dashed circles represent states in the transformed space S^α.

From these properties, we can show that in the limit, the worst case policy performance of AfD is that same as direct LfD. In the limit, LfD will yield a policy with the best prediction accuracy. Similarly, AfD's policy solver is sound: it will not lower policy performance. Further, AfD's feature elimination will not lower prediction accuracy. So AfD is working with policies at the same accuracy level as LfD. Thus, the worst case of AfD can be no worse than that of LfD.

4 Experimental results

We compare the performance of AfD with that of using demonstrations alone and that of using RL alone (using Sarsa(λ)). For direct LfD we use a C4.5 decision tree classifier to learn a direct mapping. The classifier uses all the features of the state space as input attributes and the actions of the particular domain as labels. We use two domains. *Pong* serves as a proving ground for demonstrating the correctness of AfD, using just 23 episodes as training data. The more complex *Frogger* gauges generalization and real world applicability. For Frogger, we recruited non-expert human subjects—six males and eight females—to provide demonstrations. Each had 3 minutes to familiarize themselves with the controls of the game; they were then asked to provide demonstrations by playing the game for 10 minutes.

4.1 Domains

Pong is a form of tennis where two paddles move to keep a ball in play. Our agent use one paddle while another uses a fixed policy to move in the direction to best match the ball's Y position when the ball is approaching, moving randomly otherwise. There are five features: `paddle-Y`, `ball-X`, `ball-Y`, `ball-angle`, and `opponent-Y`. `Y` coordinates and `ball-angle` have 24 possible values while `ball-X` has 18. There are two possible actions: `Up` or `Down`. Reward is 0 except when successfully returning a ball, yielding +10. The game terminates when a player loses or after 400 steps, implying a maximum policy return of 60.

The second domain is a version of the classic Frogger game (Fig. 2). In the game, the player must lead the frog from the lower part of the screen to the top, without being run over by a car or falling in the water.

Figure 2: Capture of the Frogger domain.

Player	Average Return	Episodes
Human	56.5	–
Sarsa	60.0	2554
Direct LfD	15.7	–
AfD - C4.5-greedy	60.0	59

Table 1: Performance on Pong with good demonstrations. Time is measured in seconds

At each time step, the cars advance one position in their direction of movement, and the player can leave the frog in its current position or move it up, down, left or right. The positions and directions of the cars are randomly chosen for each game, and the frog can be initially placed in any position in the lower row. The game was played at 7 steps per second, chosen empirically as a challenging enough speed to be fun.

The screen is divided in a grid, and the state features are the contents of each cell relative to the current position of the frog. For example, the feature `3u2l` is the cell three rows up and two columns to the left of the current frog position, and the feature `X1r` the cell just to the right of the frog. The possible values are `empty`, if the cell falls out of the screen; `good`, if the cell is safe; and `water`, `carR`, `carL` for cells containing water, or a car moving to the right/left.

There are 8x9 cells, so 306 features are needed to include the screen in the state representation. With 5 possible actions and 5 possible values per each cell, a table-based Q-learning algorithm might need to store up to $5^{307} \approx 10^{215}$ Q-values.

4.2 Results

Table 1 compares the performance of various learned policies in Pong. Human results are provided for reference. As we can see from the results, AfD is able to learn an optimal policy – outperforming direct LfD; moreover, while Q-learning also learns an optimal policy, AfD is significantly faster. AfD's speedup corresponds directly to the smaller abstract state space. In particular, AfD ignored the feature `opponent-Y`, which did not influence the teacher's actions.

With Frogger, we focus on how well AfD works with non-expert humans. The 14 human teachers obtained a success rate (percentage of times they lead the frog to the goal) between 31% and 55%. For learning in AfD, we filtered the demonstrations to keep only successful games and to remove redundant samples caused by the player not pressing keys while thinking or taking a small break. Each user provided on average 33.1 demonstrations ($\sigma = 9.3$), or 1230.1 samples ($\sigma = 273.6$). Note that a demonstration is a complete episode in the game and a sample is a single state-action pair.

Player	Success rate
Human	31%-55%
Direct LfD	43.9%
AfD - Cfs+voting	97.0%
AfD - C4.5-greedy	97.4%

Table 2: Frogger domain, all demos (17221 samples).

Player	Success rate
Human	55%
Direct LfD	17.1%
AfD - C4.5-greedy	88.3%

Table 3: Frogger domain, best player demos (1252).

We compared the algorithms using two sets of the demonstrations: the aggregated samples of all users, 464 demonstrations (17221 samples) in total; and the 24 demonstrations (1252 samples) from the best player.

For feature selection in AfD, we used the Cfs+voting and C4.5-greedy algorithms (Sec. 3). Before using C4.5-greedy in this domain, we reduced the number of variables using the Cfs algorithm, because iterative removal on 306 variables was too time consuming and our tests showed that the chosen features were the same as when using only C4.5-greedy. Cfs+voting was not used on the second data set, as it is designed to work with a set of demonstrations from different teachers. For reinforcement learning, we used $\gamma = 1$; and the rewards were $r = 100$ for reaching the goal, $r = -100$ for death, and $r = -1$ for any other transition.

In Table 2 we see that, using all demonstrations, AfD achieves a significantly higher success rate than direct LfD. AfD gets close to 100% success regardless of feature selection algorithm, while direct LfD does not reach 44%. Note also that the AfD policies perform better than the best human.

Table 3 shows results using only demonstrations from the best player. Even with only 7% the number of demonstrations of the previous experiment, AfD performance decreases only slightly. By contrast, direct LfD is much worse. Again, AfD performs better than the teacher. Comparing both tables we can appreciate that AfD is much more sample efficient than LfD, performing better with 20x fewer demonstrations.

By inspection, we see that AfD identified "key features" of the domain. The five key features for this domain are the cells at both sides of the frog and the three closest cells in the row immediately above. Of the original 306 features, the algorithms selected 9 to 12, and the five key features were included in these sets. Only when using just the best player demonstrations did AfD fail to include one of these key features, to which we attribute the slight decrease in performance. The other features selected were also useful: cells in the three rows contiguous to the frog and others that allowed the frog to line up with a goal cell when in a safe row.

We also compared the performance of AfD with that of applying RL directly in the raw feature space. Fig. 3 shows

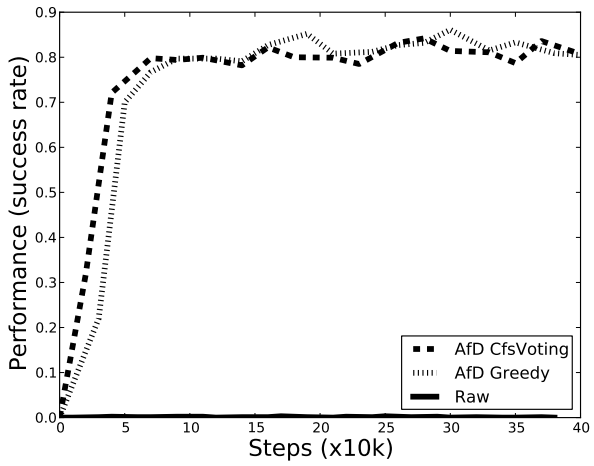

Figure 3: Performance (% of games won) of AfD and learning in the raw state space in Frogger domain.

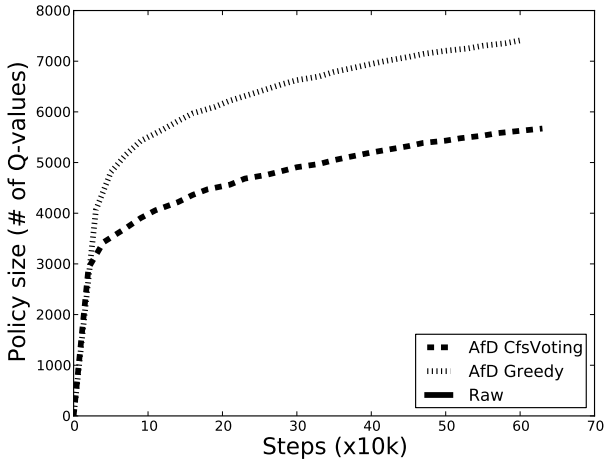

Figure 4: Policy size to number of step of AfD and learning in the raw state space in Frogger domain. Raw line is not visible because it matches the vertical axis.

that comparatively, working in the large raw state space did not achieve significant learning: states are rarely visited for a second time, retarding learning. Additionally, memory consumption grows linearly as a function of the number of steps taken. In our experiments, the raw method had consumed 19GB of memory before had to be killed. At that point, it had taken almost 1.7 million steps even though the success rate was still below 0.2%. By contrast, AfD was performing better within 1k steps (the success rate is lower than in Table 2 because exploration was not disabled). Fig. 4 illustrates the difference in state space size by showing the growth of policy sizes (the raw method is aligned with the y-axis). Policy sizes for the AfD methods begin to level off immediately.

5 Discussion

AfD presents several advantages over direct LfD. Direct LfD seeks to approximate the mapping from states to actions; however, in reality, the demonstrator's policy is not necessarily deterministic or stationary. Different demonstrators may also teach different but equally valid policies. Thus, for any given state, there may be a set of equally appropriate actions; hence we are in a multi-label supervised learning setting [Zhang and Zhou, 2007], but the data is not multi-label. For any single action, we see examples of when it should be used, but never examples of when it should not be. This problem stems from the fundamental fact that when a demonstrator shows an appropriate action for a given state, they are not necessarily indicating that other actions are inappropriate. This produces a situation commonly referred to as "positive and unlabeled data" [Letouzey *et al.*, 2009]. One of the approaches to deal with positive and unlabeled data is to assume that observed positive examples are chosen randomly from the set of all positive examples [Elkan and Noto, 2008]. This is not true for human demonstrations. AfD avoids this issue by only using the data to identify relevant features. It does not matter how well we can approximate the demonstrated policy, only that a feature has a positive contribution toward the approximation.

AfD performs better than policies built using direct LfD even when LfD is using an order of magnitude more demonstrations than AfD. The sample efficiency of AfD is one of the key advantages of the algorithm given that good human demonstrations is often expensive and time-consuming.

Given the cost of acquiring human samples, it might be desirable to avoid it altogether and use direct RL algorithms without using demonstrations; however, AfD achieves significant speedup by taking advantage of the exponential savings of feature reduction, and converges to a high performance policy in minutes while learning without demonstrations did not show improvement over the initial random policy after hours. This speedup suggests that, even including the time and cost required to acquire the human demonstrations, AfD will be more cost and time-effective than learning without using human demonstrations in many domains.

Another advantage of AfD is that its performance is not limited to that of the teacher. AfD uses the reward signal to obtain the best policy that can be expressed in its reduced feature space. This policy, as our results show, is significantly better than that of the teacher. There is no guarantee however of optimality. While it is possible to obtain optimality from the reward signal through use of RL methods, such methods could not take advantage of the abstraction we derive from human teachers. They would have to work in the full feature set. AfD focuses on providing a reasonable policy quickly by combining the ability of humans to discern relevant features, with the optimization capabilities of computers.

While it is possible that the set of task-relevant features is the full set, we believe that more often the relevant set is much smaller, making AfD widely applicable. We make three observations in support of this belief.

First, AfD uses only those features needed to represent policies in demonstrations. This set is often smaller than that needed to represent an optimal policy because humans often prefer simpler satisficing policies over optimal ones. Additionally, the focus on features for representing policies instead of Q-value functions helps to create small feature sets because

policies are often much simpler to represent than estimates of the expected sum of discounted rewards-to-go.

Second, autonomous agents to be deployed in human environments are often multi-purpose, equipped with many sensors that generate an array of useful features, but for any given task, only some of these sensors are needed.

Finally, in hierarchal settings, subtasks typically require only a small set of the available features. Often, the same is true for the root task when given solutions for the subtasks. We can imagine, for example, an agent with robotic limbs and fingers that is learning to play a piece of music on a piano. The agent must learn a subtask for playing every note and chord. For each of these subtasks, the agent only needs to consider the initial relative position of its fingers with respect to the piano. For the root task of deciding which note to execute for a specific piece, the only feature needed is how many notes of the piece have been played so far. By considering only the necessary features at each level of the hierarchy, AfD can boost the efficiency of LfD in hierarchical tasks.

6 Conclusions

We have presented Abstraction from Demonstration, a new method of Learning from Demonstration. AfD uses demonstrations from a human teacher to infer a state space abstraction before using RL algorithms to learn a policy. We found that AfD yielded better policies than direct LfD. Compared to learning in the raw state space without demonstrations, AfD provides an exponential increase in performance due to an exponential reduction in the state space size.

AfD offers a middle way between traditional LfD and learning without demonstrations, achieving better performance than both options and requiring only a small number of demonstrations. In future work, we plan to further study the sample efficiency of AfD and apply it to a wider set of complex domains, including hierarchical learning domains. We will also evaluate if we can achieve faster convergence by using demonstrations both to build an abstract feature space and to build a policy prior to guide the early exploration of the RL algorithm in the abstracted space.

References

[Abbeel and Ng, 2004] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, page 1. ACM, 2004.

[Aler *et al.*, 2005] R. Aler, O. Garcia, and JM Valls. Correcting and improving imitation models of humans for robosoccer agents. In *IEEE Congress on Evolutionary Computation*, volume 3, 2005.

[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, May 2009.

[Elkan and Noto, 2008] C. Elkan and K. Noto. Learning classifiers from only positive and unlabeled data. In *International Conference on Knowledge Discovery and Data Mining*, pages 213–220. ACM, 2008.

[Farahmand *et al.*, 2009] A.M. Farahmand, M. Ghavamzadeh, C. Szepesvari, and S. Mannor. Regularized policy iteration. *Advances in Neural Information Processing Systems*, 21:441–448, 2009.

[Hall, 1999] M.A. Hall. *Correlation-based feature selection for machine learning*. PhD thesis, 1999.

[Jong and Stone, 2005] Nicholas K Jong and Peter Stone. State Abstraction Discovery from Irrelevant State Variables. *International Joint Conference on Artificial Intelligence*, pages 752–757, 2005.

[Keller *et al.*, 2006] P.W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *International Conference on Machine learning*, pages 449–456. ACM, 2006.

[Kolter and Ng, 2009] J.Z. Kolter and A.Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning*, pages 521–528. ACM, 2009.

[Letouzey *et al.*, 2009] F. Letouzey, F. Denis, and R. Gilleron. Learning from positive and unlabeled examples. In *Algorithmic Learning Theory*, pages 71–85. Springer, 2009.

[N. Mehta *et al.*, 2007] S. Ray N. Mehta, M. Wynkoop, P. Tadepalli, and T. Dietterich. Automatic induction of maxq hierarchies. In *Proceedings of the Hierarchical Organization of Behavior Workshop*. 21st Conference on Neural Information Processing Systems, 2007.

[Ng *et al.*, 2004] A.Y. Ng, H.J. Kim, M.I. Jordan, S. Sastry, and S. Ballianda. Autonomous helicopter flight via reinforcement learning. *Advances in Neural Information Processing Systems*, 16, 2004.

[Parr *et al.*, 2007] R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *International Conference on Machine learning*, pages 737–744. ACM, 2007.

[Singh *et al.*, 1994] S.P. Singh, Tommi Jaakkola, and M.I. Jordan. Learning without state-estimation in partially observable Markovian decision processes. In *International Conference on Machine Learning*, pages 284–292, 1994.

[Smart and Kaelbling, 2002] W.D. Smart and L.P. Kaelbling. Effective reinforcement learning for mobile robots. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3404–3410, 2002.

[Zang *et al.*, 2009] P. Zang, P. Zhou, D. Minnen, and C.L. Isbell. Discovering options from example trajectories. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM New York, NY, USA, 2009.

[Zhang and Zhou, 2007] M.L. Zhang and Z.H. Zhou. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.