Exploration from Demonstration for Interactive Reinforcement Learning

Kaushik Subramanian
College of Computing
Georgia Tech
Atlanta, GA 30332
ksubrama@cc.gatech.edu

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

Andrea L. Thomaz
Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78701
athomaz@ece.utexas.edu

ABSTRACT

Reinforcement Learning (RL) has been effectively used to solve complex problems given careful design of the problem and algorithm parameters. However standard RL approaches do not scale particularly well with the size of the problem and often require extensive engineering on the part of the designer to minimize the search space. To alleviate this problem, we present a model-free policy-based approach called Exploration from Demonstration (EfD) that uses human demonstrations to guide search space exploration. We use statistical measures of RL algorithms to provide feedback to the user about the agent's uncertainty and use this to solicit targeted demonstrations useful from the agent's perspective. The demonstrations are used to learn an exploration policy that actively guides the agent towards important aspects of the problem. We instantiate our approach in a gridworld and a popular arcade game and validate its performance under different experimental conditions. We show how EfD scales to large problems and provides convergence speed-ups over traditional exploration and interactive learning methods.

1. INTRODUCTION

Reinforcement Learning (RL) [36] is the field of research focused on solving sequential decision-making tasks modeled as Markov Decision Processes. Researchers have shown RL to be successful at solving a variety of problems like computer games (Backgammon [39], Atari games [26]), robot tasks (soccer [34], helicopter control [1]) and system operations (inventory management [16]); however, in general, standard RL approaches do not scale well with the size of the problem and often require extensive engineering on the part of the designer to minimize the search space. The reason this problem arises is that RL approaches rely on obtaining samples useful for learning the underlying structure without always using smart methods to explore the problem. In RL literature this is related to balancing the exploration-exploitation tradeoff - a central problem in Reinforcement Learning. Existing methods either use a fixed (uniformly random) policy or value-based metrics [6, 35] that either result in redundant exploration and/or have prohibitively expensive sample complexity. In this work we tackle the problem of smart exploration in RL, by using human interaction. We present a model-free policy-based approach called Exploration from Demonstration (EfD) that serves to *bias an RL agent's exploration to cover the search space effectively.*

In designing an exploration policy for sequential decision-making problems, it is important to help the agent reach parts of the search space that are necessary to model in order to solve the problem. These include guiding the agent towards stochastic regions of the problem or regions of high reward for example. To facilitate exploration in these parts of the domain, we utilize statistical measures of the learning algorithm. We pose the optimization function as a linear regression problem and use relevant measures [27] to help identify *influential* regions. Such an approach has the advantage of learning to explore the problem from the agent's perspective while taking into account the underlying representation, the RL algorithm as well as the problem definition. We solicit interaction from an oracle (human or simulated) to acquire demonstrations that lead the agent towards these *influential* regions. We instantiate our approach on a gridworld and a popular arcade game (Frogger) and show how our policy-based approach is able to solve long horizon problems using exploratory demonstrations while outperforming traditional exploration and interactive learning methods.

2. RELATED WORK

There has been extensive work related to exploration in RL. With regard to the approach presented in this paper, we broadly categorize the existing work into two aspects - automatic exploration in RL and interactive machine learning. We present details on the related work in these fields to provide context to our approach.

2.1 Automatic Exploration in RL

Rmax [31] is an automatic approach introduced to perform exploration using the idea of optimism in the face of uncertainty. The approach performs well in practice, however R-max scales exponentially in the number of state variables which makes it intractable for sufficiently large problems. There are several value-based like UCB [6] and others [24, 7, 11] and model-based methods [35, 29, 18, 19], studied in the context of multi-arm bandits, that perform effective exploration by maintaining statistics about changes in the value function and the number of times state-action pairs have been visited. While successful in smaller domains, these

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

approaches run into sample complexity issues when dealing with the high-dimensional long horizon domains we aim to solve.

A smart exploration method was proposed by Gehring and Precup [14] which uses the residual (TD error) as a reward for a Q-function, whose implied policy is then used for exploration. The intuition behind this approach being that state-action pairs that have high residuals should be visited and tried more often. This method relies on stable estimates of the residual. We adapt a version of this approach in our experiments. Using an internal reward function [9] is an interesting approach to exploration in RL. In this method, a user is required to design a reward function for skill learning which is often non-trivial. While the authors provide empirical results on small-sized domains, the idea presented is promising and warrants further exploration. An approach closely related to work presented in this paper is that of Active RL [13] where the authors model the problem as a POMDP. They model the sensitivities of the policy to the unknown transition and reward function and build exploration strategies focusing on these aspects of the problem. This method relies on using Newtons method to solve the problem till convergence (which cannot be guaranteed) and they need to solve the MDP numerous times to test the sensitivity. The work most closely related to our work is one authored by Akiyama et al. [3] where they leverage concepts of least squares approaches to guide exploration in policy iteration methods. The main difference is that the approach they design requires the problems to have certain properties with respect to the reward distribution and as such it is not directly comparable.

2.2 Interactive Machine Learning

There exists is a wide variety of work in the field of Interactive Machine Learning, namely Learning by Demonstration [4], Imitation Learning [30], Policy Shaping [17] and TAMER [22]. Work by Knox and Stone [22, 23] has shown that a general improvement to learning from human feedback is possible if it is used to directly modify the action selection mechanism of the Reinforcement Learning algorithm. Although both approaches use human feedback to modify an agent's exploration policy, they still treat human feedback as either a reward or a value. In our work, we use human interaction to directly learn a policy.

Active Reward Learning [10] has been used to learn a reward function from human feedback and use that in an RL algorithm. They use the human to provide input on task executions - a score to the execution - that they then smooth using Gaussian Processes and Bayesian Optimization. Reward function design in general is known to be a hard problem as there are always possibilities of loops in the learned policy. A paper similar in theme to the work in this paper is one on active imitation learning by state queries [20]. The authors present an approach where the human interacts with the agent by giving a optimal action in a specific state or by saying that the state is bad. The query-states are chosen by a query-by-committee approach based on Bayesian Active Learning. In their approach, they assume the learner has access to a simulator of the MDP and also do not explicitly handle the case where humans provide a bad state response.

In other works, rather than have the human input be a reward shaping input, the human provides demonstrations of the optimal policy. Several papers have shown how the policy information in human demonstrations can be used for inverse optimal control [28, 2], for teaching [8], to seed an agent's exploration [5, 38], and in some cases be used entirely in place of exploration [21, 33]. DAgger [32] is no-regret online learning approach used for supervised learning. The method learns from training data and then executes the learned model. For every mistake made, the human demonstrator provides more examples in that space. These examples are appended to the training set and the learner is retrained. While it is not strictly an RL approach, it is mainly providing examples useful for a supervised learner. We note here that all of the methods presented here require/assume the human provides optimal information which is not necessary in our approach.

3. BACKGROUND AND PRELIMINARIES

Reinforcement Learning (RL) defines a class of algorithms for solving problems modeled as a Markov Decision Process (MDP). An MDP can be represented as a tuple $M = \langle S, A, \mathcal{T}, R, \gamma \rangle$ with states S, actions A, transition function $\mathcal{T} : S \times A \mapsto \Pr[S]$, reward function $R : S \times A \mapsto [R_{\min}, R_{\max}]$, and discount factor $\gamma \mapsto [0, 1]$. A policy $\pi : S \mapsto \Pr[A]$ defines the probability of selecting an action in a state. For a given MDP, a Q-function $Q^\pi(s, a)$ represents the *expected long-term reward* of taking action a in state s, and following policy π thereafter. Mathematically, the Q-function is computed as

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s') \sum_{a'} \pi(s',a') Q^\pi(s',a') \tag{1}$$

The optimal action-value function is $Q^*(s,a) = \max_\pi Q^\pi(s,a)$, and the solution to an MDP is any optimal policy π^*, which maximizes the expected reward for every state.

In this paper we use Q-learning with linear function approximation [37]. The Q-function is represented as a linear function of state-action features, $Q(s,a) = \phi(s,a)^T\theta$. Assuming $\phi(s)$ are the features used to represent the states in an MDP, we obtain $\phi(s,a)$ by duplicating the features $\phi(s)$ for all actions and only activate the ones for the action under consideration. For example if $|A| = 4$, $\phi(s,3) = [\mathbf{0}, \mathbf{0}, \phi(s), \mathbf{0}]^T$. A set of transitions made by the RL agent from a starting state to any terminal state comprises a single episode. For every transition made by the agent in the form $\langle s, a, r, s' \rangle$, where r is the reward obtained for taking action a in state s and transitioning to s', the Q-learning algorithm updates its weight vector, θ using first-order gradient methods in the following manner:

$$\begin{aligned} \delta &= r + \gamma \max_{a' \in A} [\phi(s',a')^T\theta] - \phi(s,a)^T\theta \\ \theta &= \theta + \alpha\, \delta\, \phi(s,a) \end{aligned} \tag{2}$$

Here $\alpha \mapsto [0, 1]$ is the learning rate. The error, δ is the Temporal Difference (TD) error (also loosely referred to as Bellman error). The algorithm, while not guaranteed to converge in the general case, is found to perform well in practice.

Exploration in RL is commonly achieved using two methods: ϵ-greedy and softmax action selection. In ϵ-greedy action selection, exploration is performed by uniformly sampling a random action with probability ϵ and the current best action (greedy action selection) with probability $1 - \epsilon$. The choice of ϵ is left to the designer. In some cases, a de-

cay schedule is used where the value of ϵ is decayed over time as the agent gains more domain experience. Softmax action selection takes a more informed approach to exploration. Instead of sampling a random action, the actions are weighted according to their respective Q-value estimates and sampled from the resulting distribution. The most common implementations use a Boltzmann distribution in the following manner: $\pi^B = \frac{e^{Q(s,a)/\tau}}{\sum_{a=1}^{|A|} Q(s,a)/\tau}$ where τ is a positive temperature parameter. A higher temperature value results in a more uniform distribution while a lower value results in greedy action selection.

4. APPROACH

In this section we describe an interactive approach to exploration in reinforcement learning using statistical properties relevant to the underlying learning algorithm. We outline properties of exploration policies, statistical measures useful for this purpose and how these measures can be used to solicit interaction that drives the agent's exploration.

4.1 Exploration Policies

For sequential decision-making problems, exploration policies are used to guide the agent to different parts of the search space so as to obtain good estimates of the value function while covering as much of the state-action space as possible. There are a number of factors affecting this exploration such as the dynamics of the domain, the sparsity of rewards, the size of the state-action space and the problem horizon (steps to goal). In addition to these properties, a subtle but important aspect of exploration that is implicit in existing methods is that exploration policies are not strictly stationary. As the agent gathers more information about the world, the exploration policy changes to accommodate the learned model.

Traditional methods, as explained earlier, explore using a variety of methods ranging from a uniformly random policy to value-based heuristics. These methods are prone to redundant exploration and (in some cases) expensive sample requirements. The idea of inefficient exploration also applies to methods involving human interaction, as human data is often limited to specific regions of the search space while relying on the learning algorithm to generalize effectively. To account for the characteristics of exploration policies while overcoming the limitations of existing methods, we present a policy-based approach to exploration. We solicit demonstrations based on the agent's uncertainty about its model to guide the agent to cover the search space more efficiently. For a given MDP, model uncertainty arises from a combination of stochastic elements in the domain (transition and reward function) and insufficiently explored states and actions. Keeping this in mind, we investigate properties of relevant RL algorithms and select measures that serve to characterize the model uncertainty with the goal of designing effective exploration policies.

In our work we use Q-learning with linear function approximation which uses gradient methods to perform optimization. The loss function used is akin to to minimizing the squared loss of the Bellman error [15]. Using this information, we use statistical properties of analogous methods that have been well-studied, like linear regression or least squares, to understand the impact of each data point or observation on the learning agent's model.

4.2 Statistical Measures

In order to understand the agent's model uncertainty, we review statistical analysis of linear regression methods [27] to find measures useful for our purposes. In linear regression problems, input observations can be scored by their *influence* to measure the effect they have on the learned model. A high influence score points towards observations that merit further investigation. Influence is computed as a combination of two measures: Leverage and Discrepancy. Leverage is a measure of how far a specific observation is from the convex hull of known observations. It helps recognize outliers. Discrepancy is related to how much an observation contributes towards model error. From the perspective of exploration in RL, these measures help to identify parts of the state-action space that have not been explored (using Leverage) and observations that lead to high model error (using Discrepancy). A key insight into using these type of measures for RL is that the data the agent trains on does not contain any outliers as every observation made by the agent, by interacting with the domain, is relevant to solving the MDP. This indicates that there is essentially no data to be discarded. We hypothesize that an RL agent that actively explores the observations that have high leverage and high discrepancy, i.e. overall high influence, will lead to more efficient exploration to solve the MDP.

In order to utilize these statistical measures we explicitly set up the problem as a linear set of equations that correspond to the standard form, $X\beta = y$. An RL agent is solving the MDP to optimize the function, $Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$ where $Q(s,a) = \phi(s,a)^T\theta$. It is straightforward to see that the left-hand side of the optimization function, $Q(s,a)$ or $\phi(s,a)^T\theta$ takes the place of $X\beta$ and the right-hand side forms y. The input data for our approach comes from transitions of the RL agent as it is attempting to solve the problem. The state-action features, $\phi(s,a)$ observed by the agent during transitions are used to populate the rows of the data matrix, X ($n \times k$ for n observations and k features). Given this formulation we define the statistical measures useful for exploration.

4.2.1 Leverage

Leverage is a measure useful to determine how well the state-action space has been covered as it detects outliers in the data. Given independent variables, X, we compute leverage, h using the hat matrix, H as follows [27]:

$$H = X(X^TX)^{-1}X^T \tag{3}$$

The hat matrix maps the vector of dependent variables (y) to the vector of fitted values, $\hat{y} = Hy$. The diagonal elements of the hat matrix, h_{ii} are the leverages, which describe the influence each dependent variable value has on the fitted value for observation i. Leverage values are in the range $[0, 1]$. A high value indicates that the observation is an outlier and vice versa. A fixed threshold parameter is used to detect the presence of outliers. We use 0.5 as the cut-off to indicate if an observation is an outlier. For RL problems, a high leverage indicates that the respective state-action pair is an outlier, i.e. is novel and has not been visited often.

Typically RL algorithms require large amounts of data to solve the MDP which makes it infeasible to store all the transitions in a batch. In addition to that, computing leverage can pose computational issues as it requires taking the

inverse of a matrix of size $k \times k$ (for k features) which can be very large for high-dimensional problems. To address the memory and computational concerns, we use the Sherman-Morrison formula to incrementally compute the inverse of $X^T X$. The formula is stated as follows:

$$(A + x^T x)^{-1} = A^{-1} - \frac{A^{-1} x^T x\, A^{-1}}{1 + x A^{-1} x^T} \tag{4}$$

where A^{-1} is initially set to $\frac{1}{\delta} I$ (identity matrix of size $k \times k$) and δ is a small positive number, say $1e^{-4}$. Using this computation, the leverage of an instance x of data matrix X can be computed as $x A^{-1} x^T$.

4.2.2 Discrepancy

This measure captures the observations that the learning algorithm is unable to model thus leading to large errors. Discrepancy is computed using the externally studentized residual [27]. These residuals are obtained by computing the residual for an observation and dividing it by the standard error (or standard deviation). This is done to reduce the effect of the variance in the errors and allow residuals to be compared. An externally studentized residual is one that computes the residual by taking into account the difference in the learned model with and without the observation in question. For observation i, the discrepancy, t_i can computed as:

$$\begin{aligned} t_i &= \frac{e_i}{\sqrt{MSE_{(i)}(1 - h_{ii})}} \\ MSE_{(i)} &= \frac{(n-p)MSE - \frac{e_i^2}{(1-h_{ii})}}{n - p - 1} \end{aligned} \tag{5}$$

Here MSE represents the mean-squared error, n is the number of samples, p the number of independent variables, h_{ii} is the leverage for observation i and e_i is the TD error for sample i. $MSE_{(i)}$ is the mean squared error for the model based on all observations excluding sample i. We note that MSE is typically computed using batch data which is computationally infeasible to store in large scale RL domains. It does not lend itself to incremental computations due to the max operator in the RL optimization function (when computing MSE and e_i). We circumvent this problem by storing a batch data matrix, update it with new observations using a (FIFO) sliding window and compute the required parameters [12] online. In analysis of linear systems, when the absolute value of the externally studentized residual, $|t_i|$ is greater than 2, the corresponding observation is considered an outlier that needs further investigation. Henceforth we use the term discrepancy to represent the externally studentized residual.

Using these measures the RL agent can identify observations in the MDP that require further exploration. We now describe how we use them to solicit demonstrations for exploration.

4.3 Demonstration Query

For every transition made by the RL agent, it computes the leverage and discrepancy and compares it to the respective thresholds. If either threshold is exceeded, we identify the corresponding observation as influential. To learn more about that observation and reduce its influence, we learn a policy using guidance from a person or a simulated oracle that drives the agent towards these observations.

Consider the state associated with an influential observation the agent transitioned into as s^+. To encourage exploration to s^+, it is important to bridge the gap between regions of low influence to those of high influence. Intuitively this can be explained by the idea that state-action pairs that have low influence are likely to have been frequently visited and sufficiently explored. Therefore designing a policy from parts of the state-action space that the agent knows well and visits often to those with high influence is most likely to encourage exploration to and around s^+. As explained before, leverage provides a way to identify data points that have been visited often. To acquire the necessary low influence data point that is to be connected to s^+, we review every observation i, in the current episode and compute the corresponding leverage: $h_i = \phi(s_i, a_i) A^{-1} \phi(s_i, a_i)^T$. We then compute the mean leverage, μ_h from these observations and find the state, s_i that corresponds to the data point with the closest leverage,

$$\underset{i}{\operatorname{argmin}} |h_i - \mu_h| \tag{6}$$

Once the low and high influence observations have been identified, we collect exploratory demonstrations either from a person or a simulated oracle using a Graphical User Interface (GUI) for the domain. When the algorithm queries for demonstrations, the GUI highlights the states that need to be connected by demonstrations. Using the GUI, the user can a) provide demonstration(s), b) choose to ignore the query and c) stop interacting with the algorithm altogether. The simulated oracle provides demonstrations by following the shortest distance path between the queried states. There are no inherent assumptions made about quality or quantity of demonstrations. The only requirement is for the user to be knowledgeable about the MDP dynamics to help the agent navigate in the domain. For every demonstration provided, we learn an oracle exploration policy π^O using standard supervised learning algorithms and sample an action from this policy when the agent decides to explore.

We note here that when soliciting demonstrations, the final state in the demonstration may be different from the query state requested by the agent. This is likely to be observed in domains with stochastic elements and/or non-playable characters. While there is no straightforward way to ensure a certain state is visited in an MDP, our experimental results show that the policy learned using our approach is effective at driving the agent towards influential parts of the MDP as it continues to actively request user demonstrations.

4.4 Action Selection

The oracle exploration policy, π^O defines a policy that when followed is likely to guide the agent from regions of low influence to those of high influence. However using such a policy alone to explore in and of itself can be insufficient for the purposes of RL where the goal is to arrive at the optimal policy as soon as possible. Additionally the nature of exploration is non-stationary and as such if there are limited demonstrations, the agent is less likely to explore the set of influential regions in the MDP. To account for these properties, we design our exploration policy as follows:

$$\pi^E \propto (\pi^O + \pi^L) \cdot \pi^B \tag{7}$$

where π^O is the oracle exploration policy, π^L is the leverage

Algorithm 1 Exploration from Demonstration (EfD)

repeat(for each episode):
  Initialize s
  repeat(for each step of episode):
    Compute π^E (Eqn. 7)
    Choose a (ϵ-greedy action selection using π^E)
    Take action a, observe r, s'
    Store transitions $\langle s, a, r, s' \rangle$
    Update θ and A^{-1} (Eqn. 2 & 4)
    Compute leverage and discrepancy (Eqn. 3 & 5)
    if high influence at s **then**
      $s^+ \leftarrow s$
      Compute starting state, s_i (Eqn. 6)
      Query demonstrations from s_i to s^+
      Update θ and A^{-1}
      Self-play for T_s (includes parameter updates)
    end if
    $s \leftarrow s'$
  until s is terminal
  Decay ϵ
until end of learning

value $\forall a \in A$ for state s and π^B represents the Boltzmann exploration policy. We note that the leverage values lie in the range $[0, 1]$ and for our purposes can be used as probabilities. A leverage value closer to 1 will have the effect of sampling the corresponding action more often. Intuitively π^E represents the exploration policy that chooses between the oracle demonstration or the leverage values, weighted by the softmax Q-values of the actions in the state. This exploration policy allows the agent to reach regions of high influence using human demonstrations or select actions with high leverage while actively seeking the goal. We use π^E in an ϵ-greedy fashion to facilitate exploration in our approach.

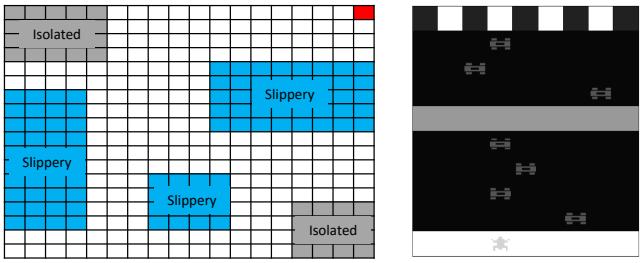
4.5 Exploration from Demonstration

We now outline our approach with all the pieces defined using Algorithm Block 1. The objective of Exploration from Demonstration (EfD) is to learn the optimal policy using RL while ensuring the agent actively explores regions of the state-action space that have a potentially large influence on the learned model.

EfD as described in Algorithm Block 1 has a tendency to query the user demonstrations repeatedly as high influence regions are often in close proximity to others. This results in the leverage and discrepancy thresholds being crossed very often within the same episode. In order to make EfD more user-friendly we include a predefined fixed time period, T_s where the agent executes self-play without any user queries. During self-play Q-learning and leverage parameters (θ & A^{-1}) continue to be updated. We note that EfD does not modify any theoretical guarantees of the methods used as Q-learning is an off-policy algorithm. This completes the description of our approach.

5. EXPERIMENTAL SETUP

To validate the performance of EfD we conduct experiments on a gridworld and popular arcade game domain and compare our method to several baselines. In this section we describe the domains used in our experiments and the relevant baselines.



(a) Gridworld (b) Frogger1x

Figure 1: A snapshot of the two domains used in our experiments.

5.1 Domains

We use two domains to empirically highlight the performance and properties of EfD. We represent these domains as MDPs in the following manner:

Gridworld. This domain is designed by adapting the specifications outlined in [14]. We implement an 18×18 discrete grid (Figure 1a) with four deterministic actions that move the agent up, down, left and right. The goal is to reach the top right corner of the grid. For every step taken, the agent accrues a step cost of -1 and a reward of 0 at the goal state. The blue shaded regions represent slippery squares. If the agent transitions out of a slippery square (both into an unshaded square or another slippery square), the reward is uniformly distributed in the interval $[-12, +10]$. The gray shaded regions represent isolated squares. Any transition that leads the agent into this region from an empty square has a 0.1 probability of success. Once inside, movements within the isolated region as well as those leading out are not restricted. Transitions leading into the bounding walls keeps the agent's state unchanged. An episode starts with the agent randomly placed in the grid and stops when the agent reaches the goal. We used identity features to represent the state space.

Frogger. In the game of Frogger (Figure 1b), the agent's goal is to navigate from the bottom to the top of the grid while avoiding the cars and water pits (shown as dark squares in the top row). The cars move in straight lines in their individual rows either going left or right. The direction is randomly chosen at the start of an episode and stays fixed until the episode ends. The intermediate gray row(s) represent safe zones for the agent, i.e. no cars and water pits. The agent has 5 actions: 4 directional actions and a no-op action. Within an episode, the transitions are deterministic. The reward function is $+1000$ for reaching the goal, -100 for dying (directly hitting a car, crossing over a car, falling into water) and 0 everywhere else. An episode starts with the agent in a random position in the bottom row and stops when the agent dies or reaches the goal. The state space of the domain consists of the agent's position along with the position and direction of travel of cars in the grid and represented using binary features. The Frogger domain is particularly interesting as it lends itself to easy scalability. The domain can be made more complex, i.e. have a longer solution horizon, by increasing the number of intermediate rows between the start and goal positions. We use this property to show how our method scales with the size of

the domain. We refer to the domain configuration in Figure 1b as Frogger1x and use Frogger2x to indicate doubling the number of rows used in Frogger1x and Frogger4x to indicate a quadruple version of the same.

5.2 Baselines

We implement five baselines in our experiments and compare their performance to EfD. Uniform random exploration and softmax exploration comprise two of the baselines. The remaining three are defined as follows:

Learning from Demonstration + RL. In this method we acquire demonstrations of optimal behavior from people or a simulated oracle. These demonstrations are used to learn a policy using supervised learning methods (in this case logistic regression). We use this policy as the seed policy to initiate RL. We execute the algorithm on the domain and report the results.

Exploration by TD error. This approach draws from insights highlighted in this paper [14] and learns an exploration policy based on TD error. In our implementation we use the current estimate of TD error (the absolute value) as the reward for a given state-action pair and learn a Q-function using this information. A policy is extracted from the Q-function using softmax action selection. The Q-function learned in this process plays the role of driving the agent towards parts of the state-action space that have high TD error in order to gather more information in those regions. We note that this approach is not strictly consistent with standard MDP assumptions as the reward function is non-stationary (TD error is constantly changing). While we counteract this effect to a certain degree by using a small learning rate and a decaying exploration parameter, our experiments show that performance was not greatly affected by this characteristic.

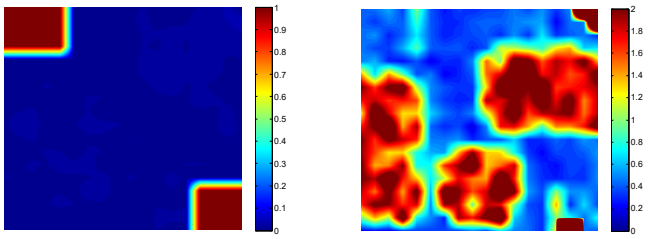
Exploration by Leverage. We derive an exploration policy by computing the leverage on data consisting of visited state-action pairs. For any given state, we compute the leverage for all actions, normalize the results and use it as a distribution from which we sample exploratory actions. As explained earlier leverage captures outliers in the data, which in this case would represent actions, for a given state, that have not been tried often. This way by sampling from normalized leverage values for all actions in a state, the agent is more likely to sample new actions. This baseline is useful to signify the importance of exploratory demonstrations.

6. EXPERIMENTS AND RESULTS

We implement EfD for the chosen domains and highlight the results achieved along with several tests that provide insight into EfD's performance under different experimental conditions.

6.1 Using Leverage and Discrepancy for Exploration

In this experiment we use the gridworld (Figure 1a) to show the utility of leverage and discrepancy as useful measures to guide exploration. The gridworld is suitable for this purpose due to several design choices made. Firstly the isolated (gray) regions in the grid represent parts of the state-space that are hard to reach and thus unlikely to be explored by the agent. Secondly the non-deterministic reward function (represented by slippery blue patches on the grid) pose some difficulty to the learning algorithm in accurately modeling the underlying value-function. We conduct two experiments to show the individual contribution of the chosen exploration measures.



(a) Leverage heat map (b) Discrepancy heat map

Figure 2: Leverage and discrepancy heat maps generated from random exploration of the gridworld domain.

To highlight the utility of leverage, we perform a random walk in the gridworld for 50 episodes. In each episode, the agent starts in a random position and moves randomly until the goal is reached. For every step taken, features of visited state-action pairs are used to form the data matrix X. Using X, we compute the hat matrix H (see Equation 3) and use that to derive the leverage $h(s,a)$ for every state-action pair.

In Figure 2a, we plot a heat map using $\max_a h(s,a)\ \forall s \in S$. The heat map shows the correspondence between the regions of high leverage ($h \geq 0.5$) and hard to explore regions of the gridworld (isolated gray regions). Leverage, as explained earlier, is used to identify outliers in the data. In this case the heat map presents the gray region as outliers which necessitates the need for further exploration. Analogously this effect can be seen in other more complex high-dimensional domains where it is hard to entirely cover the state-action space. Leverage, as shown here, captures regions that the learning agent is unable to reach often.

While leverage captures how often state-action pairs have been visited, it does not capture details about the transition function, reward function and RL algorithm's learned model. Here we show how discrepancy is useful for this task. We perform Q-learning with linear function approximation on the gridworld domain for 50 episodes. We set $\gamma = 0.99$, $\alpha = 0.1$ and $\epsilon = 1.0$. We store all the transitions $\langle s, a, r, s' \rangle$ from the sampled episodes and compute the mean squared error using the learned weight vector θ (refer Equation 5). The mean squared error is used to derive the discrepancy $t(s,a)$ for every state-action pair.

In Figure 2b, we plot a heat map using $\max_a |t(s,a)|\ \forall s \in S$. The heat map shows the correspondence between the regions of high discrepancy ($|t| > 2$) and the slippery regions in the gridworld. Intuitively this is to be expected as the TD error in these areas is likely to have large magnitude and high variance and that warrants further investigation. We note that the bright red patch in the top right corner of the heat map signifies the high residual obtained at the goal and its adjoining states. Using this heat map as a threshold for exploration would draw the agent towards the slippery patches and the goal until the learning algorithm captures the underlying model and the residual decreases.

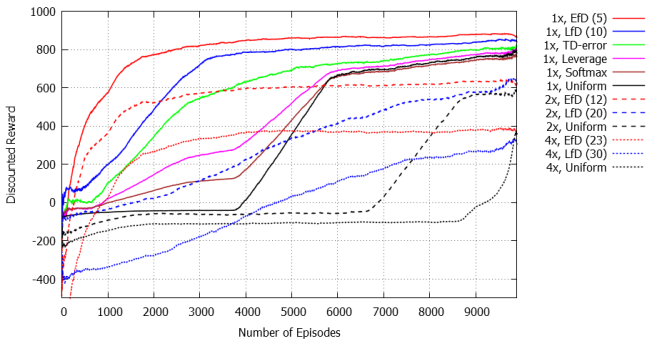
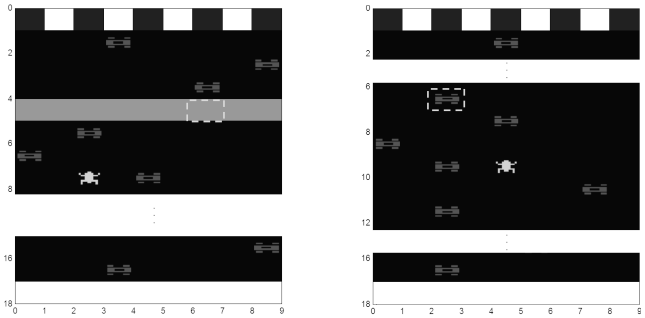


Figure 3: Performance of EfD and baselines on the Frogger game domain of varying sizes (1x, 2x and 4x) averaged over 10 trials. The numbers in parenthesis are the average number of input user demonstrations.



(a) Agent queries the user for demonstrations to the highlighted position based on the leverage threshold.

(b) Agent queries the user for demonstrations to the highlighted position based on the discrepancy threshold.

Figure 4: Examples of the types of states queried by the agent during EfD when applied to Frogger2x (18 rows).

The experiment serves to highlight the roles played by leverage and discrepancy and how they guide the agent's exploration. Leverage guides the agent towards state-action pairs that have not been visited often during learning and the discrepancy guides the agent towards regions of the domain which the learning algorithm has difficulty modeling the value-function.

6.2 EfD for Frogger

In this experiment we instantiate the EfD algorithm in the Frogger domain for different sizes of the problem, Frogger 1x (10 rows), Frogger 2x (18 rows) and Frogger 4x (34 rows). We perform Q-learning with linear function approximation with $\gamma = 0.99$, $\alpha = 0.0006$ and $\epsilon_{start} = 0.8$. We use the following decay schedule for the exploration parameter: $\epsilon = \frac{\epsilon_{start} \times N_0}{(N_0 + Ep\#)}$ where N_0 is the decay rate and $Ep\#$ is the current episode number. We set N_0 as 1500, 2500 and 5000 respectively for the three versions of Frogger. The threshold parameters for EfD were fixed with the leverage threshold set at 0.5 and discrepancy threshold at 2. The self-play time period for EfD was set to $T_s = 500$, $T_s = 2000$ and $T_s = 5000$ steps respectively for the three versions of Frogger. The temperature for softmax Boltzmann exploration was set to 50. We acquired demonstrations from two users who were both familiar with the dynamics of the game. We received anywhere from 5 to 30 demonstrations depending on the size of the domain that was being tested. Demonstration time in total was no more than 10 to 15 mins. The human policy was learned using logistic regression with a learning rate of 0.01. We plot the results of this experiment along with comparative baselines in Figure 3. We see that EfD converges to the optimal policy faster than the baselines using a small number of demonstrations. To ease readability we plot only a subset of the baseline methods in Figure 3 as the performance of the baselines (TD-error, Leverage and Softmax) were consistent across the three sizes. The performance is further improved over the baselines as the size of the domain is increased. This is explained by highlighting how EfD queries and utilizes user demonstrations. In the initial stages of learning, the user is queried with demonstrations leading to states in the rows closer to the bottom row. As the agent gains experience, demonstrations are requested for states further up. In this process, the agent incrementally explores the rows until it finally reaches the goal in the top row. Such an incremental learning approach makes it easier for agent to reach the goal as well as easier for the user to provide demonstrations. This also explains why LfD (+ RL) does not perform as well as EfD for larger grids. The size of the domain limits the search space covered by the human demonstrations as well as prohibits optimal demonstrations from start to the goal. EfD performs better by using the RL algorithm's inductive bias as well as the underlying representation to acquire incremental demonstrations that are most useful to the agent. These results were consistent across both users. The TD-error and leverage baseline methods while more informed do not perform as well due to their redundant exploration. An interesting observation of EfD from our experiments is that, by using thresholds for the statistical measures, with sufficient experience the agent automatically ceases to request demonstrations. In which case, we observe that the agent has enough information to model the Q-function and solve the MDP.

6.3 Types of States Queried

Here we take a closer look at the types of states queried by the agent during EfD. We present results from the Frogger2x domain which consists of 18 rows from start to goal. Figure 4a is an example of an agent query, based on the leverage measure, where a demonstration is required between the frog near row 8 to the highlighted grid position near row 4. An observation that exceeds the leverage threshold indicates that the agent has not visited that state-action pair often and therefore requires input demonstrations. Such a query points towards how EfD gathers information about the state-action - decomposing the domain in smaller regions. Demonstrations are requested from known regions to unknown regions and often they are in close proximity to each other. Providing a demonstration for such a query would be easier than providing optimal demonstrations from start to end in this domain. Figure 4b is an example of an agent query based on the discrepancy threshold. We note that the highlighted position is around a car near row 6 which is a terminal state with reward -100. The discrepancy here exceeded the threshold as the agent's current model was unable to make an accurate prediction of the Q-value of an action in this state and thus requested a demonstration.

We would also like to highlight a few uncommon queries

that provide interesting insights into the method. In some cases the agent requests demonstrations from a state where the frog is closer to the goal to states where the frog is further away. From the perspective of solving the MDP, using such a policy would encode suboptimal information, however for policy-based exploration, it only serves to get a better estimate of the Q-function. Note that exploration is carried out by combining the human policy with softmax policy (Section 4.4) which therefore ensures the agent select actions that are more likely to lead it towards the goal. For some queries we observe that the agent's position on the grid remains the same for both start and final states, while the position of cars is different. With respect to EfD, these are different states and therefore it is a valid demonstration query. This shows how EfD makes demonstration queries by taking into account the underlying representation.

6.4 Effect of Input Demonstrations and Threshold Parameters

In this experiment, we test the sensitivity of EfD to the quality of demonstrations used to learn the exploration policy. While we do not place any assumptions on the quality of demonstrations, we analyze the degrees to which performance is affected as the quality of demonstration is varied. We use the simulated oracle for this experiment under different demonstration noise conditions: Oracle0.1, Oracle0.3, Oracle0.5. An oracle with noise 0.1 (Oracle0.1) will provide the required demonstration 90% of the time and 10% of the time, take random actions. The results of this experiment are summarized in Table 1. As evidenced by the

	Frogger1x (5)	Frogger2x (12)	Frogger4x (23)
Oracle0.0	**2560 ± 150**	**3194 ± 230**	**4430 ± 410**
Oracle0.1	2752 ± 321	3470 ± 527	4893 ± 564
Oracle0.3	2648 ± 469	3304 ± 699	5218 ± 866
Oracle0.5	5102 ± 932	6329 ± 875	7688 ± 1043
ϵ-greedy	6570 ± 120	8555 ± 212	10000 ± 405

Table 1: EfD performance as a function of the quality of input demonstrations from a simulated oracle in the Frogger domain. The values represent the number of episodes taken by each method to converge to the optimal policy. The numbers in parenthesis are the number of input demonstrations the simulated oracles are limited to. We include results from the ϵ-greedy baseline for comparison. The results are averaged over 10 trials.

table, the performance of EfD varies based on the quality of input demonstrations. Relative to Oracle0.0 (optimal oracle demonstrations), Oracle0.1 and Oracle0.3 achieve similar performance. This is explained by the fact that for most query demonstrations there exist multiple ways to reach the desired state and neither path is any more optimal than the other from the perspective of exploration. By introducing noise in the oracle demonstrations, they can potentially explore more states than Oracle0.0 which can include both low and high influence observations. However this has the effect of increased variance in performance for noisy simulated oracles. Oracle0.5 (with 50% random action selection) has large variance in its performance but still outperforms ϵ-greedy uniform random exploration.

In our experiments, we set the leverage threshold at 0.5 and the discrepancy threshold at 2. Changes to these parameters directly affect the number of demonstrations queried by the agent which affects the amount of exploration carried out by the agent. Higher values results in fewer demonstration queries and as a result most of the exploration is carried out by the agent autonomously. On the other hand lower thresholds result in frequent queries which has the effect of learning an exploration policy close to a uniform policy. In general, from tests in our domain, we find that setting leverage threshold to 0.5 and discrepancy threshold anywhere in the range $[2, 6]$ provides the best results.

7. DISCUSSION AND CONCLUSION

Here we highlight the **benefits of using a policy-based approach** for exploration in RL in the context of EfD. When faced with large domains with sparse rewards and long horizons, a policy-based approach is less vulnerable to the large sample requirements of value-based methods as the information acquired from a single demonstration allows the agent to extends its range of exploration over multiple timesteps. Additionally such a method does not concern itself solely with reward information. The statistical measures used in EfD (leverage and discrepancy) focus on different aspects of the MDP which allow the algorithm to function well across a wider class of problems. This is in contrast to value-based methods which rely on large samples of reward information to estimate the uncertainty in the value function often made complicated in sparse reward and long horizon domains. Also EfD does not require optimal demonstrations to learn but instead demonstrations that serve to connect two regions of the agent's choice. As these demonstrations are used for exploration, they can be potentially noisy (which may in some cases help the agent).

In this paper we presented a model-free policy-based approach called Exploration from Demonstration (EfD) that performs interactive exploration for RL algorithms. Our method adapts statistical measures of linear regression to capture aspects of an MDP that are important to explore and model in order to learn the optimal Q-function. We highlight the properties of these measures in an instructional gridworld MDP and empirically test our approach on a popular arcade game under different experimental conditions. We show how EfD scales to larger problems and outperforms baselines using only exploratory demonstrations while placing very few requirements on the quality and quantity of input data. Our method is particularly suited to problems which have a long horizon and sparse rewards as well as those domains where optimal demonstrations are hard to acquire. In the future we would like to extend EfD to learn a model of the MDP, thus allowing the algorithm to request examples from arbitrary states rather that waiting to transition to those areas. Another interesting avenue for future work is the idea of extending EfD to work with more data efficient RL algorithms like Least Squares Policy Iteration [25].

Acknowledgments

We thank the reviewers for their helpful comments in improving the paper. This work is supported by ONR grant No. N000141410003.

REFERENCES

[1] P. Abbeel, A. Coates, and A. Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] P. Abbeel and A. Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proc. of the 21st ICML*, 2004.

[3] T. Akiyama, H. Hachiya, and M. Sugiyama. Efficient exploration through active learning for value function approximation in reinforcement learning. *Neural Networks*, 23(5):639 – 648, 2010.

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

[5] C. Atkeson and S. Schaal. Learning tasks from a single demonstration. In *Proc. of the IEEE ICRA*, pages 1706–1712, 1997.

[6] P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397–422, Mar. 2003.

[7] S. Bubeck, R. Munos, and G. Stoltz. Pure exploration in multi-armed bandits problems. In *ALT*, volume 5809 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 2009.

[8] M. Cakmak and M. Lopes. Algorithmic and human teaching of sequential decision tasks. In *AAAI Conference on Artificial Intelligence*, 2012.

[9] O. Şimşek and A. G. Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 833–840. ACM, 2006.

[10] C. Daniel, M. Viering, J. Metz, O. Kroemer, and J. Peters. Active reward learning. In *Proceedings of Robotics: Science & Systems (R:SS)*, 2014.

[11] R. Dearden, N. Friedman, and S. Russell. Bayesian Q-learning. In *Proc. of the 15th AAAI*, pages 761–768, 1998.

[12] T. G. Dietterich. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer-Verlag, 2002.

[13] A. Epshteyn, A. Vogel, and G. DeJong. Active reinforcement learning. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 296–303. ACM, 2008.

[14] C. Gehring and D. Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS*, pages 1037–1044, 2013.

[15] M. Geist and O. Pietquin. Algorithmic survey of parametric value function approximation. *IEEE Trans. Neural Netw. Learning Syst.*, 24(6):845–867, 2013.

[16] I. Giannoccaro and P. Pontrandolfo. Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161, 2002.

[17] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In *Neural Information Processing Systems 26*, pages 2625–2633, 2013.

[18] T. Hester, M. Lopes, and P. Stone. Learning exploration strategies in model-based reinforcement learning. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1069–1076, 2013.

[19] T. Hester and P. Stone. TEXPLORE: Real-time sample-efficient reinforcement learning for robots. *Machine Learning*, 90(3), 2013.

[20] K. Judah, A. P. Fern, T. G. Dietterich, and P. Tadepalli. Active imitation learning: Formal and practical reductions to i.i.d. learning. *Journal of Machine Learning Research*, 15:4105–4143, 2014.

[21] L. P. Kaelbling, M. L. Littmann, and A. W. Moore. Reinforcement learning: A survey. *JAIR*, 4:237–285, 1996.

[22] W. B. Knox and P. Stone. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *Proc. of the 9th Intl. Conf. on AAMAS*, pages 5–12, 2010.

[23] W. B. Knox and P. Stone. Reinforcement learning from simultaneous human and MDP reward. In *Proc. of the 11th Intl. Conf. on AAMAS*, pages 475–482, 2012.

[24] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006.

[25] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. In *Journal of Machine Learning Research*, volume 4, pages 1107–1149, 2003.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015.

[27] D. C. Montgomery, E. A. Peck, and G. G. Vining. *Introduction to Linear Regression Analysis (4th ed.)*. Wiley & Sons, Hoboken, July 2006.

[28] A. Y. Ng and S. Russell. Algorithms for inverse reinforcement learning. In *Proc. of the 17th ICML*, 2000.

[29] J. Pazis and R. Parr. PAC optimal exploration in continuous space markov decision processes. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[30] B. Price and C. Boutilier. Accelerating reinforcement learning through implicit imitation. *Journal of Artificial Intelligence Research (JAIR)*, 19:569–629, 2003.

[31] M. T. Ronen I. Brafman. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, pages 213–231, 2002.

[32] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011.

[33] W. D. Smart and L. P. Kaelbling. Effective

reinforcement learning for mobile robots. In *ICRA*, 2002.

[34] P. Stone, R. S. Sutton, and G. Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behaviour*, 13(3):165–188, 2005.

[35] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[36] R. Sutton and A. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[37] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[38] M. Taylor, H. B. Suay, and S. Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *Proc. of the Intl. Conf. on AAMAS*, pages 617–624, 2011.

[39] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, Mar. 1995.