A MODEL BASED APPROACH TO EXPLORATION OF CONTINUOUS-STATE MDPS USING DIVERGENCE-TO-GO

Matthew Emigh, Evan Kriminger, José C. Príncipe

University of Florida
Department of Electrical and Computer Engineering
Gainesville, Florida 32611

ABSTRACT

In reinforcement learning, exploration is typically conducted by taking occasional random actions. The literature lacks an exploration method driven by uncertainty, in which exploratory actions explicitly seek to improve the learning process in a sequential decision problem. In this paper, we propose a framework called Divergence-to-Go, which is a model-based method that uses recursion similarly to dynamic programming to quantify the uncertainty associated with each state-action pair. Information-theoretic estimators of uncertainty allow our method to function even in large, continuous spaces. The performance is demonstrated on a maze and mountain car task.

1. INTRODUCTION

The problem of exploration in single state Markov Decision Processes (i.e. n-armed bandits) has largely been solved [1]. Traditional statistical techniques such as Gittins index [2] or upper confidence bounds [3] are essentially unimprovable [4]. However, exploitation of multi-state Markov Decision Processes (MDPs) remains an open problem currently limited to special cases, such as deterministic MDPs [5]. We propose a model-based reinforcement learning framework to guide exploration for the class of continuous-state MDPs in which it is sensible to assign a metric to its state space. We further discuss a method to exploit our learned model using a variant of Monte Carlo tree search (MCTS).

The most popular exploration strategy is to select a random action with a small probability, \( \epsilon \), which is decreased over time. This epsilon-greedy approach is simple, but is not ideal in practical problems where only a limited amount of interaction with the environment is possible. In such problems, more directed approaches are needed, which very efficiently explore, in a directed manner, and stop exploring when the task becomes solvable. A system which intelligently explores the environment requires more information about the progress of learning. In particular, we need to quantify the uncertainty that is present in the quantities from which decisions are made (e.g. value functions or state transition models). When these quantities are known perfectly, no exploration is needed. It is a difficult problem to model uncertainty in a reinforcement learning (RL) problem because not only does the amount of uncertainty constantly change, but it is a local and temporal attribute. Knowledge about a state necessarily requires knowledge about future states. However, this problem mirrors the temporal credit assignment problem which is solved by RL, by using recursion to assign value to states which are part of a sequence. There is a parallel between this management of rewards, and how we can deal with uncertainty.

2. BACKGROUND

2.1. Markov Decision Process

An MDP is a model for a certain class of stochastic control problems. A (discounted) MDP can be represented mathematically as a 5-tuple \((\mathcal{X}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)\), where \(\mathcal{X}\) is the set of all possible states (state space), \(\mathcal{A}\) is the set of all possible actions (action space), \(\mathcal{P}\) is the set of transition probabilities, \(\mathcal{R}\) is the set of rewards, and \(\gamma\) is the discount factor. Since the available actions are typically state-dependent, we define \(\mathcal{A}(x) = \{a \in \mathcal{A}|a\text{ is allowed in } x\}\). A state or action occurring at time \(t\) is denoted by \(X_t\) and \(A_t\), where \(X_t \in \mathcal{X}\) and \(A_t \in \mathcal{A}\). When an agent takes an action \(a\) in a certain state \(x\), this leads to a new state \(x'\). This new state is a random variable, and a probability is assigned to the transition as follows: 

\[ P(x, a, x') = P(X_{t+1} = x'|X_t = x, A_t = a) \]

Note that the next state probability is conditioned only on the current state and action. This is known as the Markov property, and is the primary assumption for modeling with an MDP. Rewards can be associated with states \((R(x'))\) or state transitions \((R(x, x'), R(x, a, x'))\). We write \(R_{t+1}\) to denote the reward received when the state transitions from \(X_t\) to \(X_{t+1}\) and \(R(\cdot)\) to denote rewards associated with specific states or state transitions. Since state transitions are random, rewards \(R_{t+1}\) are as well. Furthermore, rewards \(R(x, a, x')\) may be random in their own right.

Since typically only a small fraction of possible state transitions receive a non-zero reward, credit must be assigned to actions that indirectly led to the rewarded state. This is achieved by evaluating actions with the rewards that are received at current and future times, which is formalized in a criterion known as the Q-value, or value function. The value function is defined as the sum of expected future

\[ V(x) = \sum_{x'} P(x, a, x') R(x', a, x') \]

This work was supported by ONR grant N00014-14-1-0542
Model-based RL methods model the underlying MDP by learning or other reinforcement learning framework known as model-based RL. Actions that maximize said Q-function. In contrast, there exists an exclusively by learning a Q-function and choose a policy by selecting \( \pi \). The policy may be stochastic or deterministic. If the policy is stochastic, then it is a mapping from the states \( X \) to a distribution over \( A \). The MDP is considered to be solved when the policy maximizes the expected future sum of rewards from any state \( X \in X \). We call \( \pi^* \) the optimal policy for the MDP. Every MDP has at least one optimal policy \( \pi^* \).

### 2.2. Model-Based Learning

Popular RL algorithms such as Q-learning and SARSA, which are known as model-free RL algorithms, learn a policy without ever learning transition probabilities \( P \) and rewards \( R \). They learn exclusively by learning a Q-function and choose a policy by selecting actions that maximize said Q-function. In contrast, there exists another reinforcement learning framework known as model-based RL.

Model-based RL methods model the underlying MDP by learning or approximating \( P \) and \( R \). Once a model of the MDP is known, a policy \( \pi \) can be formulated offline using methods such as value iteration and policy iteration. Naturally, model-based methods require more memory and computational resources, but have the ability to learn a policy much more quickly in terms of number of actions that must be taken and number of times states must be visited in \( X \). For example, in temporal-difference based model-free learning, values must be propagated back through the space as state transitions are experienced.

For small finite-state MDPs, the simplest modeling approach is to approximate the rewards and transition probabilities using a simple maximum likelihood estimate. For example, the reward associated with state \( x' \) can be computed as \( \frac{1}{N_{x'}} \sum_i R(x_i) \), where \( N_{x'} \) is the number of times state \( x' \) has been visited and \( R(x_i) \) is the reward received the \( i \)th visit. Furthermore, \( P(x, a, x') \) can be computed as \( \frac{N_{x,a,x'}}{N_{x,a}} \), where \( N_{x,a} \) is the number of times action \( a \) has been taken in state \( x \) and \( N_{x,a,x'} \) is the number of times \( x' \) has been reached after taking action \( a \) in state \( x \). Clearly, this approach becomes impractical as the number of states and actions increase.

For larger state and action spaces, models which are able to generalize are required. Schaal and Atkeson [8] used locally weighted regression to train a “deterministic” inverse \( \hat{a}_k = f^{-1}(x_k, x_{k+1}) \) and forward \( \tilde{x}_{k+1} = f(x_k, \hat{a}_k) \) transition model. A linear-quadratic controller was then computed for each local linear model. Using this framework, they were able to teach a robot to juggle “devil sticks.” Hester and Stone [9] use decision trees to generalize a transition and reward model. However, this framework requires discrete MDPs whose state spaces can be factored [10]. Deisenroth [11] used Gaussian process regression to learn a model, but the computational requirements are extremely high. Jong and Stone [12] created a transition and reward model similar to the work in this paper using averaging kernels. However, their work was not an information-theoretic point of view, and is unable to exploit the catalog of algorithms and techniques that have been developed within the information-theoretic learning [13] framework.

### 2.3. Uncertainty in RL

Model-based RL approaches have been shown to be well-suited for the estimation of uncertainty in a RL setting [11], since meaningful quantities can be obtained from probabilities. Such quantities can be used to improve sample efficiency. That is, while online, the model can help direct exploration of the space in such a way as to improve itself. This is especially important in the case of infinite state spaces since it is impossible to visit every state and difficult to even revisit previously experienced states. In the case of the simple empirical modeling approach described above, exploration mechanisms, such as ES3 [14] and Rmax [15], have been proposed. Unfortunately, such approaches require visiting every state exhaustively.

In [16], a model-based RL approach is presented, in which a parametric distribution over the Q-value is estimated as Bayesian inference. The distribution captures uncertainty in the estimate of the Q-value, and actions are chosen with a quantity known as the Value of Perfect Information. This criterion explores an action only if enough uncertainty remains such that it is possibly the optimal action. After sufficient exploration, the variances in the Q-value estimates are small enough that it is unlikely any action other than the highest Q-value action is optimal. Which this method provides a measure of uncertainty that is local in the state-action space, and includes temporal propagation of uncertainty, the Bayesian Q-learning framework is only practical for small, discrete RL problems. It is our goal to provide a local and temporal measure of uncertainty that can be easily estimated even in large and continuous state-action spaces.

### 3. DIVERGENCE-TO-GO

We seek a quantity that measures the uncertainty of a state-action pair. In model-based RL, the agent interacts with the environment and observes state transitions and estimates a model of the state transition probabilities under the given action. That is, for any point \( x \in X \) and \( a \in A \), we have a belief distribution which estimates the transition distribution \( P(y|x, a) \), where \( y \in X \). Each time we observe a transition \( x \to y \), we can update our belief distribution \( P(y|x, a) \). Following the observation of a state transition, we can compute the divergence \( D([P_{new}(y|x, a)] || P_{old}(y|x, a)) \) between the two belief distributions, before and after incorporating the new knowledge. When a state transition is observed that greatly changes the model, this means that the agent was quite uncertain about the outcome of the action taken in that particular state.

The divergence between state transition estimates is useful for tracking the learning process because it measures uncertainty, not inherent randomness in the environment. Only when observations greatly change our beliefs, do we assign high uncertainty to a state-action pair. However, so far, this measure is local in time, while a proper uncertainty criterion for RL should include the uncertainty of not only the current state-action, but future states as well. Dynamic programming is a perfect mechanism for such temporal credit assignment. Rather than a reward for each state-action, consider using the local measure of uncertainty provided by the divergences between state transition model estimates. Thus, dynamic programming used in conjunction with the transition model divergences, provides the basis for our divergence-to-Go framework.

We define the divergence-to-go by the expected discounted sum of divergences for the transition probability estimates over time, 

\[
dtg(x, a) = E \left[ \sum_{t=0}^{\infty} \gamma^t D(x_t) \right],
\]
where the expectation is taken over the ensemble of possible paths taken through the state space and the stochasticity is provided by non-deterministic state transitions. Just as the value function in RL characterizes the rewards that will be accumulated during a sequence of actions, the divergence-to-go captures the uncertainty of a sequence of actions. In a Markov setting, the sum of divergence values naturally arises because the divergence for the joint transition probability over time becomes a sum of divergences of the marginal transition probabilities. In fact, entropy was used in such a recursive fashion in [17].

Unlike a value function, the divergence-to-go is not a stationary quantity that we estimate for each state. Instead, as the models improve, the divergence values will decrease to zero. Choosing actions based on the maximum divergence-to-go has the effect of maximizing exploration in the sense of visiting the states (or experiencing the state transitions) that most effectively learn the state transition model. Just as with choosing the action with the largest Q-function, the divergence-to-go accounts for future uncertainty as well.

3.1. Transition and Divergence Estimators

Estimating divergence values requires an easy to estimate transition model in continuous spaces. A kernel density estimator allows for the estimation of continuous probability models, but must be modified for estimating transition probabilities. Consider Figure 1. The black points \((x_1, x_2, x_3)\) and arrows represent state-transitions that have been observed. We wish to estimate the transition model for the red point \((x_4)\). Using the intuitive idea that points close in space have similar transition probability distributions, we choose to weight the contribution of each observation to our model by a similarity measure. In Figure 2, the contributions made by \(x_1\) and \(x_2\) are much larger than that made by \(x_3\), which is illustrated by the smaller Gaussian “bump” placed over \(x_3\).

Suppose we wish to estimate the transition \((x \rightarrow y)\) pdf starting from point \(x\) under action \(a\). Then,

\[
\hat{p}(y|x, a) = \sum_{i} s(x_i, x) k_x(y_i - x_i, y - x)
\]

where

\[
s(x_i, x) = \frac{s'(x_i, x)}{\sum_i s'(x_i, x)}
\]

because \(\sum_i s(x_i, x)\) must sum to 1 so that \(\hat{p}(y|x, a)\) will integrate to 1.

We can compute \(s'\) using a product kernel composed of the state and action kernels so that \(s' = k_x(x_1, x_2)k_a(a_1, a_2)\). Rather than maintaining separate quantities for each action, using a product kernel over the joint state-action space allows for even continuous action spaces. The product kernel makes intuitive sense because DTG values can only be expected to be similar when both states and actions are similar. Using the Gaussian kernel,

\[
s^2(x_1, x_2) = \exp\left\{ -\frac{d(x_1, x_2)^2}{\sigma}\right\}
\]

\[
s^a(a_1, a_2) = \exp\left\{ -\frac{d(a_1, a_2)^2}{\sigma}\right\}
\]

We make the assumption that points close in space have similar transition probability distributions. We also assume that transition distributions are centered at the initial point of transition. That is, throughout the state space, transition distributions vary smoothly about the initial point of transition. Without this assumption, or some similar weaker one, our model would never be able to generalize. Using this assumption, we center each of our observations around the initial point of the distribution we wish to estimate, weighting them according to the discussion above. Finally, for action spaces that allow it, we may assume that similar actions have similar transition distributions.
Consider Figure 3. An agent was allowed to randomly wander a 2-dimensional 15 \times 15 open space. The agent was permitted to select any action in the interval \( a \in [0, 2\pi) \). For each selected action, the agent moved one unit in the direction \( a + \theta \), where \( \theta \) is randomly sampled from \( \mathcal{N}(0, (\pi/6)^2) \). The agent randomly wandered the space in this way for 10000 steps. Using the discussion above, and equations (3)-(6), we computed the probability distribution \( p(y|x, a) \) at 8 positions and angles. The contours conform to what we would expect from the test set-up described.

With this transition model, we can compute the divergence between successive models \( f \) and \( g \). Information-Theoretic Learning (ITL) [13] provides divergence quantities with simple closed-form estimators, when used with kernel density pdf estimation. The Euclidean distance between pdfs \( D_{euc}(f,g) = \int (f(x) - g(x))^2\,dx \) can be estimated as

\[
\hat{D}_{euc} = \hat{V}_f + \hat{V}_g - 2\hat{V}_c, 
\]

where

\[
\hat{V}_f = \sum_{i=1}^{N} \sum_{j=1}^{N} s_f(x_i, x) s_f(x_j, x) G_{\sigma \sqrt{2}}(y_i - x_i, y_j - x_j) 
\] (8)

\[
\hat{V}_g = \sum_{i=1}^{N-1} \sum_{j=1}^{N-1} s_g(x_i, x) s_g(x_j, x) G_{\sigma \sqrt{2}}(y_i - x_i, y_j - x_j) 
\] (9)

\[
\hat{V}_c = \sum_{i=1}^{N} \sum_{j=1}^{N-1} s_f(x_i, x) s_g(x_j, x) G_{\sigma \sqrt{2}}(y_i - x_i, y_j - x_j). 
\] (10)

Since dtg is a dynamic programming framework, we must solve it with analogous methods to those in RL. In continuous state-action spaces, we must map state-action pairs to dtg values. We use the Kernel Temporal Difference (KTD) algorithm [18], which can be considered as a kernelized version of Q-learning. Let \( \bar{x} := (x, a) \) be the state-action pair. We compute divergence-to-go with KTD using the product kernel over \( \bar{x} \).

\[
\delta_t = D + \gamma \max_{a} \left\{ dtg_{t+1}(\bar{x}) - dtg_t(\bar{x}) \right\} 
\] (11)

\[
dtg_t(\bar{x}) = \alpha \sum_{j=1}^{t} \delta_j k(\bar{x}, \bar{x}_j) 
\] (12)

, where \( D \) is a any divergence measure.

### 3.2. Tree Building

One of the advantages of model-based learning is the ability to use the model offline to simulate the effects of sequences of actions. Using the Markov assumption, we are able to estimate any n-step action sequence by repeatedly resampling that sequence until a suitable transition distribution is learned. If we build a tree in which each action is represented by a branch and each sequence of actions is represented by a node, the data stored at each node can be used to represent that action sequence’s transition distribution.

Unfortunately, the number of possible sequences exponentially increases with sequence length. Even for small action spaces, e.g., the simulation of every possible action sequence quickly becomes intractable. We use a variation of Monte Carlo tree search (MCTS) [19] to select which action sequences to resample.

### 4. RESULTS

#### 4.1. Two-dimensional Maze

We now present results for a simple two-dimensional maze. For this simulation, the agent starts at the point (1,1). Similarly to the agent in Figure 3, at each time step, it must select an action \( a \in [0, 2\pi) \). Each of these actions moves the agent one unit in a direction \( a + \theta \) where \( \theta \) is drawn from \( \mathcal{N}(0, (\pi/6)^2) \). In order to learn a model of the entire space, divergence-to-go will attempt to explore the entire maze in a relatively even fashion. We first compare the number of iterations required to reach within \( \sqrt{2} \) of the right corner (10,0). Figure 4 shows each point visited in the maze for two representative simulations of random exploration and dtg-based exploration. Table 1 gives the results.

<table>
<thead>
<tr>
<th>( n_{\text{steps}} )</th>
<th>mean</th>
<th>max</th>
<th>min</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>dtg</td>
<td>487.77</td>
<td>1651</td>
<td>145</td>
<td>311.47</td>
</tr>
<tr>
<td>Random</td>
<td>2486.1</td>
<td>8887</td>
<td>327</td>
<td>2019.9</td>
</tr>
</tbody>
</table>

Furthermore, as a measure of evenness of exploration, we compute the mean distance of 100 randomly selected first nearest neighbors, over 100 Monte Carlo trials. Table 2 gives the results. The results indicate that exploration by divergence-to-go results in a more
uniformly spread sample selection.

Table 2. Divergence-to-go vs random exploration in a 2-d maze (mean 1-nn distance)

<table>
<thead>
<tr>
<th>dtg</th>
<th>mean</th>
<th>max</th>
<th>min</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-1.1672</td>
<td>-2.372</td>
<td>0.1162</td>
<td>0.263</td>
</tr>
<tr>
<td>Random</td>
<td>-1.213</td>
<td>-2.443</td>
<td>0.0651</td>
<td>0.0374</td>
</tr>
</tbody>
</table>

4.2. Mountain Car

4.2.1. Set up

The mountain car [20] is a commonly used RL test bed in which the agent attempts to drive an underpowered car up a 2-dimensional hill. Depending on the set up, the action set consists of \( \mathcal{A}_{disc} = \{-1, 0, 2\} \) or \( \mathcal{A}_{cont} = [-1, 1] \). Each time step, the agent selects an action from \( \mathcal{A} \) based on the 2-dimensional state space consisting of the cart’s position and velocity. Negative actions drive the cart to the left, while positive actions drive it to the right. To reach the goal, the agent must climb the hill to the left and then allow the combination of gravity and positive actions to drive the cart to the top of the hill on the right.

Table 3. Number of steps from cart at rest at bottom of hill to goal for various dtg kernel sizes

<table>
<thead>
<tr>
<th>dtg ks</th>
<th>0.01</th>
<th>0.1</th>
<th>0.25</th>
<th>5</th>
<th>rand</th>
<th>SARSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>#steps</td>
<td>9657</td>
<td>857</td>
<td>511</td>
<td>231</td>
<td>41994</td>
<td>3792</td>
</tr>
<tr>
<td>#steps (MCTS)</td>
<td>1026</td>
<td>488</td>
<td>235</td>
<td>151</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

4.2.2. Results

Divergence-to-go was applied to the mountain car problem. The state space and action spaces were normalized so that the range of each dimension is one. The similarity kernels were set to .05, and the state kernel size was set to .001. Furthermore, a divergence-to-go learning rate \( \alpha \) of 0.75 and a discount factor \( \gamma \) of 0.9 were used. The action space \( \mathcal{A}_{cont} \) was used. Figure 5 shows each point visited in the trajectory of the mountain car state space from at rest at the bottom of the hill to the goal state for various divergence-to-go kernel sizes. Smaller kernel sizes lead to a better model as the state/action space is more thoroughly sampled while larger sizes cause the space to be explored more quickly.

Table 3 shows the number of steps until exploration by divergence-to-go reaches the goal state for the four kernel sizes shown in figure 5. Furthermore, the average number of steps required to reach the goal over 1000 Monte Carlo trials are shown for both a random policy and (tabular) SARSA [21] over a discretized state space. The following parameters were used for SARSA: a learning rate \( \alpha \) of 0.7, an eligibility trace parameter \( \lambda \) of 0.3, an \( \epsilon \)-greedy exploration parameter \( \epsilon \) of 0.1, and a discount parameter \( \gamma \) of 1. Furthermore, the action space \( \mathcal{A}_{disc} \) was used. Table 3 also shows the number of steps required to reach the goal after the policy learned by divergence-to-go is improved using 250 episodes of Monte Carlo tree search (MCTS). The results show that pure exploration is able to reach the goal more quickly than a traditional reinforcement learning method on a single-episode basis.

5. CONCLUSION

The Divergence-to-go framework has been developed to guide exploration such that the agent seeks uncertain paths and covers the state space. The divergence between estimated transition probabilities provides a local measure of uncertainty for each state-action pair. This measure is used similarly to a reward in dynamic programming to provide a quantification of uncertainty that includes future states. Kernel methods allow the dtg to be estimated directly from the state-action-reward information of the RL task. A dtg policy prevents agents from wasting environmental interaction by returning to well-known states. We show that on some tasks, divergence-to-go can actually solve the task much faster than a pure RL approach. Since transition probabilities are maintained to estimate the dtg, model-based RL methods can be used to further exploit experience offline and improve the solution. One drawback to the divergence-to-go framework includes the difficulty of kernel density estimation in high dimensional spaces due to the curse of dimensionality. As the dimensionality of the state space grows, the data becomes more sparse, and density estimators take longer to converge to their true distributions. Future work will focus on coupling the dtg framework with dimensionality reducers which learn lower-dimensional representations of the underlying state space.

6. REFERENCES


