Proceedings of the Twenty-Ninth International
Florida Artificial Intelligence Research Society Conference

Learning Continuous State/Action Models for Humanoid Robots

Astrid Jackson and **Gita Sukthankar**
Department of Computer Science
University of Central Florida
Orlando, FL, U.S.A.
{ajackson, gitars}@eecs.ucf.edu

Abstract

Reinforcement learning (RL) is a popular choice for solving robotic control problems. However, applying RL techniques to controlling humanoid robots with high degrees of freedom remains problematic due to the difficulty of acquiring sufficient training data. The problem is compounded by the fact that most real-world problems involve continuous states and actions. In order for RL to be scalable to these situations it is crucial that the algorithm be sample efficient. Model-based methods tend to be more data efficient than model-free approaches and have the added advantage that a single model can generalize to multiple control problems. This paper proposes a model approximation algorithm for continuous states and actions that integrates case-based reasoning (CBR) and Hidden Markov Models (HMM) to generalize from a small set of state instances. The paper demonstrates that the performance of the learned model is close to that of the system dynamics it approximates, where performance is measured in terms of sampling error.

1 Introduction

In recent years, reinforcement learning (RL) has emerged as a popular choice for solving complex sequential decision making problems in stochastic environments. Value learning approaches attempt to compute the optimal value function, which represents the cumulative reward achievable from a given state when following the optimal policy (Sutton and Barto 1998). In many real-world robotics domains acquiring the experiences necessary for learning can be costly and time intensive. Sample complexity, which is defined as the number of actions it takes to learn an optimal policy, is a major concern. Kuvayev and Sutton (1996) have shown significant performance improvements in cases where a model was learned online and used to supplement experiences gathered during the experiment. Once the robot has an accurate model of the system dynamics, value iteration can be performed on the learned model, without requiring further training examples. Hester and Stone (2012) also demonstrated how the model can be used to plan multi-step exploration trajectories where the agent is guided to explore high uncertainty regions of the model.

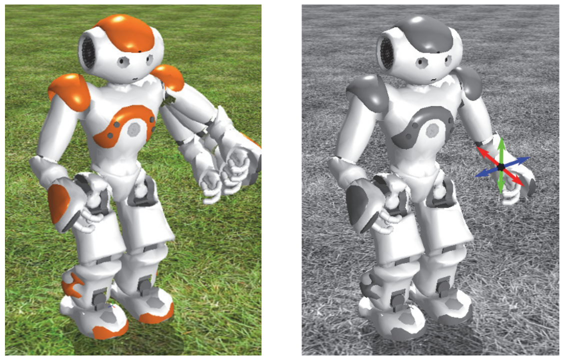

(a) (b)

Figure 1: Approximation of the transition probability density function for the joint motion of a humanoid robot. (a) The model is learned from the trajectory of the robot's left wrist. (b) The continuous state space is represented by the x-y-z position of the robot's end effectors; the continuous action space is composed of the forces on these effectors in each direction.

Many RL techniques also implicitly model the system as a Markov Decision Process (MDP). A standard MDP assumes that the set of states S and actions A are finite and known; in motion planning problems, the one-step transition probabilities $P(s_i \in S | s_{i-1} \in S, a_{i-1} \in A)$ are known, whereas in learning problems the transition probabilities are unknown. Many practical problems, however, involve continuous valued states and actions. In this scenario it is impossible to enumerate all states and actions, and equally difficult to identify the full set of transition probabilities. Discretization has been a viable option for dealing with this issue (Hester and Stone 2009; Diuk, Li, and Leffler 2009). Nonetheless, fine discretization leads to an intractably large state space which cannot be accurately learned without a large number of training examples, whereas coarse discretization runs the risk of losing information.

In this paper we develop a model learner suitable for continuous problems. It integrates case-based reasoning (CBR) and Hidden Markov Models (HMM) to approximate the successor state distribution from a finite set of experienced instances. It thus accounts for the infinitely many successor states ensuing from actions drawn from an infinite set. We apply the method to a humanoid robot, the Aldebaran Nao, to approximate the transition probability density function for a sequence of joint motions (Figure 1). The results show that the model learner is capable of closely emulating the environment it represents.

In the next section we provide formal definitions relevant to the paper. In Section 3 we give a detailed description of our Continuous Action and State Model Learner (CASML) algorithm. CASML is an extension of CASSL (Continuous Action and State Space Learner) (Molineaux, Aha, and Moore 2008) designed for learning the system dynamics of a humanoid robot. CASML distinguishes itself from CASSL in the way actions are selected. While CASSL selects actions based on quadratic regression methods, CASML attempts to estimate the state transition model utilizing an Hidden Markov Model (HMM). Section 4 describes our experiments, and in Section 5 we analyze the performance of our model approximation algorithm. Section 6 summarizes how our method relates to other work in the literature. We conclude in Section 7 with a summary of our findings and future work.

2 Background

2.1 Markov Decision Process

A Markov Decision Process (MDP) is a mathematical framework modeling decision making in stochastic environments (Kolobov 2012). Typically it is represented as a tuple (S, A, T, γ, D, R), where

- S is a finite set of states.
- A is a finite set of actions.
- $T = \{P_{sa}\}$ is the set of transition probabilities, where P_{sa} is the transition distribution; i.e. the distribution over probabilities among possible successor states, of taking action $a \in A$ in state $s \in S$.
- $\gamma \in (0, 1]$ is the discount factor which models the importance of future rewards.
- $D : S \mapsto [0, 1]$ is an initial-state distribution, from which the initial state s_0 is drawn.
- $R : S \times A \mapsto \mathbb{R}$ is a reward function specifying the immediate numeric reward value for executing $a \in A$ in $s \in S$.

2.2 Case Based Reasoning

Case-based reasoning (CBR) is a problem solving technique that leverages solutions from previously experienced problems (cases). By remembering previously solved cases, solutions can be suggested for novel but similar problems (Aamodt and Plaza 1994). A CBR cycle consists of four stages:

- *Retrieve* the most similar case(s); i.e. the cases most relevant to the current problem.
- *Reuse* the case(s) to adapt the solutions to the current problem.

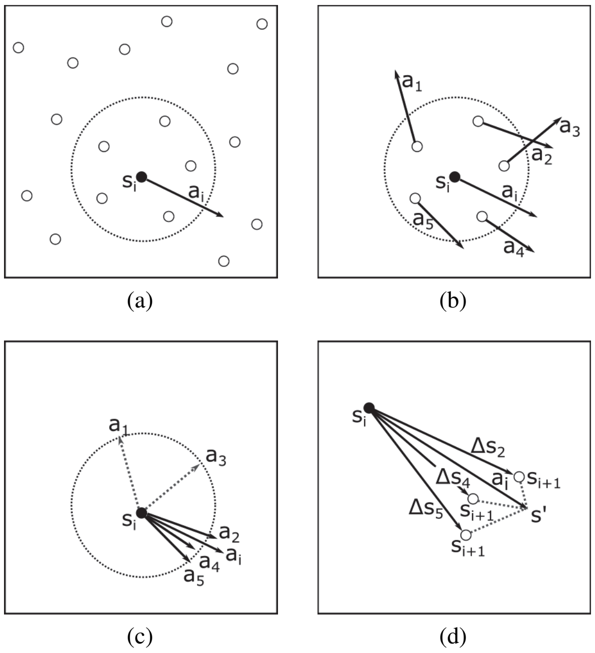


Figure 2: Process of extracting the successor states by utilizing the transition case base. (a) Retrieve similar states $(s_1, \ldots, s_n)$ of kNN. (b) Determine the actions previously performed in these states. (c) Reuse by identifying similar actions $(a_1, \ldots, a_k)$ using cosine similarity. (d) Select s_{i+1} as possible successor states.

- *Revise* the proposed solution if necessary (this step is similar to supervised learning in which the solution is examined by an oracle).
- *Retain* the new solution as part of a new case.

A distinctive aspect of CBR systems is that no attempt is made to generalize the cases they learn. Generalization is only performed during the *reuse* stage.

3 Method

This section describes Continuous Action and State Model Learner (CASML)[1], an instance-based model approximation algorithm for continuous domains. CASML integrates a case-based reasoner and a Gaussian HMM and returns the successor state probability distribution. At each time step, the model learner receives as input an experience of the form $\langle s_{i-1}, a_{s_i-1}, s_i \rangle$, where $s_{i-1} \in S$ is the prior state, $a_{i-1} \in A$ is the action that was taken in state s_{i-1} and $s_i \in S$ is the successor state. Both the states in S and the actions in A are real-valued feature vectors.

Much like in Molineaux, Aha, and Moore (2008) the case base models the effects of applying actions by maintaining the experienced transitions in the case base $C_T : S \times A \times \Delta S$. To reason about possible successor states, C_T retains the observations in the form:

$$c = \langle s, a, \Delta s \rangle,$$

[1]Code available at https://github.com/evenmarbles/mlpy

where Δs represents the change from the prior state to the current state.

The case base supports a case-based reasoning cycle as described in (Aamodt and Plaza 1994) consisting of retrieval, reuse, and retention. Since the transition model is fit to the experiences acquired from the environment directly, C_T is never revised.

In addition to the case base, a Gaussian HMM is trained with the state s_i of each received experience. While the case base allows for the identification of successor states, the HMM models the transition probability distribution for state sequences of the form $\langle s_{i-1}, s_i \rangle$.

At the beginning of each trial, the transition case base is initialized to the empty set. The HMM's initial state distribution and the state transition matrix are determined using the posterior of a Gaussian Mixture Model (GMM) fit to a subset of observations, and the emission probabilities are set to the mean and covariance of the GMM.

New cases are retrieved, retained and reused according to the CBR policies regarding each method. A new case $c^{(i)} = \langle s_{i-1}, a_{i-1}, \Delta s \rangle$, where $\Delta s = s_i - s_{i-1}$ is retained only if it is not correctly predicted by C_T, which is the case if one of the following conditions holds:

1. The distance between $c^{(i)}$ and its nearest neighbor in C_T is above the threshold:
$$d(c^{(i)}, 1\text{NN}(C_T, c_i)) > \tau.$$
2. The distance between the actual and the estimated transitions is greater than the error permitted:
$$d(c^{(i)}_{\Delta s}, C_T(s_{i-1}, a_{i-1})) > \sigma.$$

The similarity measure for case retention in C_T is the Euclidean distance of the state features of the case's state.

Algorithm 1 Approximation of the successor state probability distribution for continuous states and actions

```
 1: $C_T$: Transition case base $\langle S \times A \times \Delta S \rangle$
 2: $H$: HMM trained with state features $s$
 3: $T$: Transition probabilities $T : S \times A \times S \mapsto [0, 1]$
 4: ———————————————————————
 5: procedure P($s_i$, $a_i$)
 6:     $C_S \leftarrow$ retrieve($C_T, s_i$)
 7:     $C_A \leftarrow$ reuse($C_S, a_i$)
 8:     $T(a_i, s_i) \leftarrow \varnothing$
 9:     $\forall c \in C_A : T(s_i, a_i) \leftarrow$
            $T(s_i, a_i) \cup \langle c_s, \text{predict}(H, \langle s_i, s_i + c_{\Delta s} \rangle) \rangle$
10:     return normalize($T(s_i, a_i)$)
11: end procedure
```

Given a set of k episodes, the initial state distribution is defined as $D = \{s_0^{(0)}, s_0^{(1)}, \dots, s_0^{(k)}\}$. Therefore the initial state is added to D at the beginning of each episode, such that $D = D \cup s_0^{(i)}$. Initial states are drawn uniformly from D.

To infer the successor state probability distribution for an action a_i taken in state s_i, CASML consults both the transition case base and the HMM. The process is detailed in Algorithm 1. First the potential successor states must be identified. This is accomplished by performing case *retrieval* on the case base, which collects the states similar to s_i into the set C_S (see Figure 2(a)). Depending on the domain, the retrieval method is either a k-nearest neighbor or a radius-based neighbor algorithm using the Euclidean distance as the metric. The retrieved cases are then updated into the set C_A by the *reuse* stage (see Figure 2(b) and 2(c)). C_A consists of all those cases $c_k = \langle s_k, a_k, \Delta s_k \rangle \in C_S$ whose action a_k are cosine similar to a_i, which is the case if $d_{\cos}(a_k, a_i) \geq \rho$. At this point all successor states s_{i+1} can be predicted by the calculation of the vector addition $s_i + \Delta s_k$ (see Figure 2(d)).

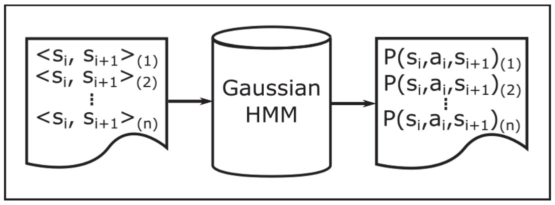

$<s_i, s_{i+1}>_{(1)}$
$<s_i, s_{i+1}>_{(2)}$
$<s_i, s_{i+1}>_{(n)}$
Gaussian HMM
$P(s_i, a_i, s_{i+1})_{(1)}$
$P(s_i, a_i, s_{i+1})_{(2)}$
$P(s_i, a_i, s_{i+1})_{(n)}$

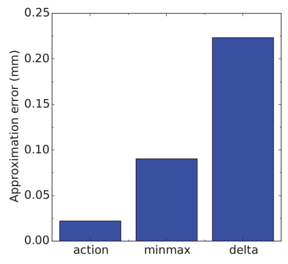
Figure 3: Estimation of the state transition probabilities for a given state and a given action. The forward algorithm of the HMM is used to calculate the probability for each one-step sequence $\langle s_i, s_{i+1} \rangle$, that was identified utilizing the case base (Figure 2(d)).

The next step is to ascertain the probabilities for transitioning into each of these successor states (Figure 3). The log likelihood of the evidence; i.e. the one-step sequence $\langle s_i, s_{i+1} \rangle$, for each of the potential state transitions can be induced by performing the forward algorithm of the HMM. The successor state probability distribution is then obtained by normalizing over the exponential of the log likelihood of all observation sequences.

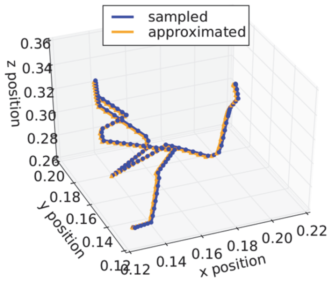
4 Experiments

Our goal was for a humanoid robot, the Aldebaran Nao, to learn the system dynamics involved in performing a joint motion. For our experiment we first captured a motion using an Xbox controller to manipulate the robot's end effectors . The x-, y-, and z-positions of the end effectors serve as data points for the joint trajectory and the continuous actions are the forces applied to the end effectors in each of the coordinate directions. The force is determined by the positional states of the controller's left and right thumb sticks which produce a continuous range in the interval [-1, +1] and generate a displacement that does not exceed 4 millimeters. We record the actions applied at each time step as the policy that will be sampled from the model. All experiments were performed in the Webots simulator from Cyberbotics[2], however they are easily adaptable to work on the physical robot. Henceforth, the trajectories which are acquired from the simulator will be referred to as the *sampled* trajectories and the trajectories sampled from the model will be referred to as *approximated* trajectories.

[2]http://www.cyberbotics.com

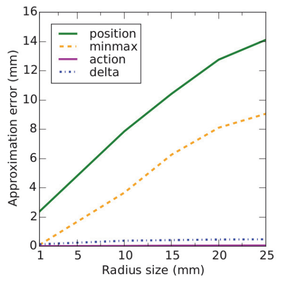


(a)

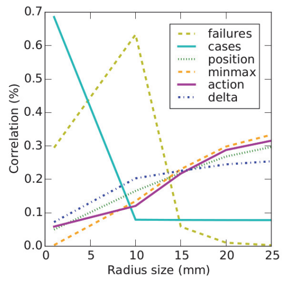


(b)

Figure 4: Performance of the model as quantified by the approximation error. (a) The accuracy of the model identified by various error metrics. (b) Comparison of the estimated trajectory and the actual trajectory obtained from the simulator. The error measurements and the estimated trajectory are averaged over 50 instances, each estimated from a model instance that was trained with a set of 50 trajectories sampled in the simulator by following the action sequence.



(a)



(b)

Figure 5: Correlation between accuracy of the model, the number of cases retained in the case base and the number of approximation failures, where accuracy is traded for efficiency. (a) Accuracy of the approximated trajectory and (b) correlation between the approximation errors, the number of approximation failures, and the number of cases are plotted as a function of the radius employed by the neighborhood algorithm.

For our first set of experiments we generated one action sequence for the movement of the robot's left arm, which resulted in a 3-dimensional state feature vector for the joint position of the wrist. The model was trained with 50 trajectories sampled by following the previously acquired policy in the simulator. Subsequently, a trajectory was approximated using the trained model. Figure 4 illustrates the performance of the algorithm in terms of a set of error measurements. Given the n-length sampled trajectory $Tr_s = \{s_{s,0}, s_{s,1}, \ldots, s_{s,n}\}$, the n-length approximated trajectory $Tr_a = \{s_{a,0}, s_{a,1}, \ldots, s_{a,n}\}$, and the effect of each action; i.e. the true action, stated as $\Delta s_{s,t} = s_{s,t} - s_{s,t-1}$ and $\Delta s_{a,t} = s_{a,t} - s_{a,t-1}$ for the sampled and the approximated trajectories respectively, the error metrics are defined as follows:

- The **action** metric identifies the divergence of the true action from the intended action. Specifically, we denote the action metric by:

$$\left|\frac{1}{n-1}\sum_{t=0}^{n}||a_t - \Delta s_{s,t}||_2\right| - \left|\frac{1}{n-1}\sum_{t=0}^{n}||a_t - \Delta s_{a,t}||_2\right|$$

- The **minmax** metric measures whether an approximated state is within the minimum and maximum bounds of the sampled state. Assume a set of m sampled trajectories $\{s_0^{(i)}, s_1^{(i)} \ldots, s_n^{(i)}\}_{i=1}^m$ and that $err_{\text{min}} = ||\min(s_t^{([1:m])}) - s_{a,t}||_2$ and $err_{\text{max}} = ||\max(s_t^{([1:m])}) - s_{a,t}||_2$ are defined as the differences for $s_{a,t}$ from the minimum and the maximum bounds respectively, then minmax =

$$\begin{cases} 0, & \min(s_t^{([1:m])}) \leq s_{a,t} \leq \max(s_t^{([1:m])}) \\ \min(err_{\text{min}}, \\ \quad err_{\text{max}}) & \text{otherwise} \end{cases}$$

- The **delta** metric measures the difference of the approximated and the sampled true action. It is given by:

$$\frac{1}{n-1}\sum_{t=0}^{n}||\Delta s_{s,t} - \Delta s_{a,t}||_2$$

- The **position** metric analyzes similarity of the resulting states and is given by:

$$\frac{1}{n-1}\sum_{t=0}^{n}||s_{s,t} - s_{a,t}||_2$$

We set $\tau = 0.005, \sigma = 0.001$ and $\rho = 0.97$. Furthermore, we set the retrieval method to the radius-based algorithm with the radius set to 1 mm.

5 Results

The results in Figure 4(a) show that the model captures the amount of deviation from the intended action quite accurately (see action metric). The fact that the minmax error is relatively small proves that the one-step sequences manage to stay in close proximity of the experienced bounds. However, by requiring the joint displacement to be within the bounds of the sampled states, the minmax metric makes a stronger claim than the action metric which only concedes that the motion deviated from the intended action. An even stronger claim is made by the delta metric which analyzes the accuracy of the motion. As expected the delta error is comparatively larger since in a stochastic environment it is less likely to achieve the exact same effect multiple times. Though not shown in Figure 4(a) the position error is even more restrictive as it requires the location of the joints to be in the exact same place and accumulates rapidly after the first deviation of motion. Nevertheless, as can be seen in Figure 4(b), the approximated trajectory is almost identical to that of the sampled trajectory.

The case retrieval method used for this experiment was the radius-based neighbor algorithm, a variant of k-nearest

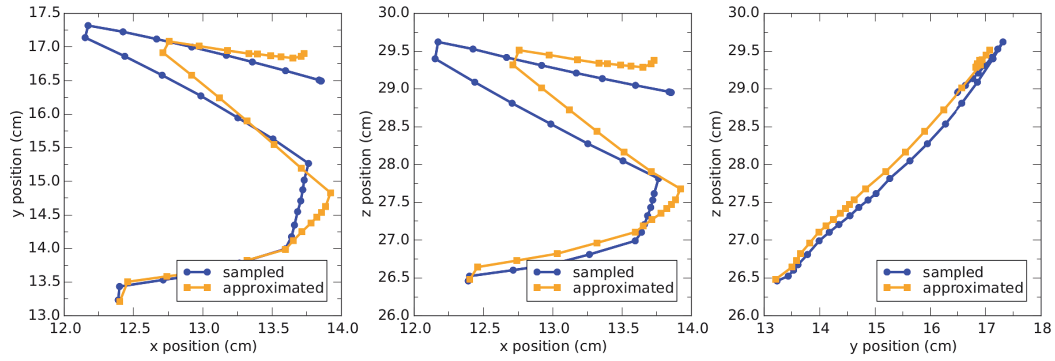


Figure 6: Comparison of the trajectory extracted from controlling the left arm in the simulator and the trajectory approximated by the model after it was trained with ten distinctive trajectories. The deviations of the wrist joint position in the x-, y-, and z-directions are scaled to cm. The approximated trajectory shown is an average over 20 instances.

neighbor that finds the cases within a given radius of the query case. By allowing the model to only select cases within the current state's vicinity, joint limits can be modeled. The drawback is that, in the case of uncertainty in the model, it is less likely to find any solution. To gain a better understanding, we compared the effect of varying the training radius on the model's performance. Otherwise the setup was similar to that of the previous experiment. Figure 5(a) shows that as the size of the radius increases, the accuracy of the model drops. This outcome is intuitive considering that as the radius increases, the solution includes more cases that diverge from the set of states representing this step in the trajectory. At the same time the number of cases retained by the case base decreases. Therefore, the found solutions are less likely to be exact matches to that state. Interestingly, even though the delta measure is more restrictive than the minmax metric, it does not grow as rapidly. This can be explained by the *reuse* method of the case base, which requires cases to have similar actions to be considered part of the solution. This further suggests that the model correctly predicts the effects of the actions and thus generalizes effectively. As is evident from Figure 5, the accuracy of the model is correlated with the number of approximation failures, defined as incomplete trajectories. The effect is that more attempts have to be made to estimate the trajectory from the model with smaller radii. This makes sense since deviations from the sampled trajectories are more likely to result in a scenario where no cases with similar states and actions can be found. In this experiment the problem is compounded since no attempt at exploration is made. There is an interesting spike in the number of failures when the radius is set to 10 mm (see Figure 5(b)). An explanation for this phenomenon is that for small radii, there is a lesser likelihood for large deviations. Therefore, the solution mostly consists of the set of cases that were sampled at this step in the trajectory. On the other hand with larger radii, the solution may include more cases which allows for generalization from other states. For medium radii, there is a greater potential for the occurrence of larger deviations when the radius does not encompass sufficient cases for good generalization.

A third experiment illustrates the performance of the algorithm on estimating a trajectory that the model had not previously encountered during learning. For this purpose the model was trained on a set of ten distinctive trajectories which were derived from following unique policies. The policies were not required to be of the same length, however each trajectory started from the same initial pose. The model was then used to estimate the trajectory of a previously unseen policy. We continued to utilize the radius-based neighboring algorithm as the similarity measure and set the radius to 15 mm. We also set $\rho = 0.95$ to allow for more deviation from the action. This was necessary, as the model needed to generalize more on unseen states than were required in the previous experiments. There were also far fewer instances to generalize from since the model was only trained on 10 trajectories. Furthermore, only 30% of the experienced instances were retained by the model's case base. A comparison of the approximated and the sampled trajectory is depicted in Figure 6 which shows that the model is capable of capturing the general path of the trajectory. Figure 7 quantitatively supports this claim. Even the position metric, which is the most restrictive, shows that the location of the robot's wrist deviates on average by only 3.45 mm from the expected location.

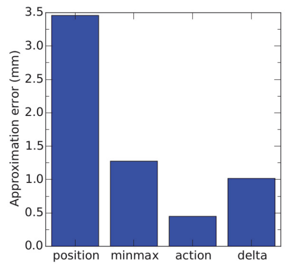


Figure 7: The accuracy of the model after ten training samples. The results are averages over 20 approximation trajectories. The minmax error was calculated by sampling 20 trajectories in the simulator for the validation.

6 Related Work

Prior work on model learning has utilized a variety of representations. Hester and Stone (2009) use decision trees to learn states and model their relative transitions. They demonstrate that decision trees provide good generalization to unseen states. Diuk, Li, and Leffler (2009) learn the structure of a Dynamic Bayesian Network (DBN) and the conditional probabilities. The possible combinations of the input features are enumerated as elements and the relevance of the elements are measured in order to make a prediction. Both these model learning algorithms only operate in discretized state and action spaces and do not easily scale to problems with inherently large or continuous domains.

Most current algorithms for reinforcement learning problems in the continuous domain rely on model-free techniques. This is largely due to the fact that even if a model is known, a usable policy is not easily extracted for all possible states (Van Hasselt 2012). However, research into model-based techniques has become more prevalent as the need to solve complex real-world problems has risen. Ormoneit and Sen (2002) describe a method for learning an instance-based model using the kernel function. All experienced transitions are saved to predict the next state based on an average of nearby transitions, weighted using the kernel function. Deisenroth and Rasmussen (2011) present an algorithm called Probabilistic Inference for Learning Control (PILCO) which uses Gaussian Process (GP) regression to learn a model of the domain and to generalize to unseen states. Their method alternately takes batches of actions in the world and then re-computes its model and policy. Jong and Stone (2007) take an instance-based approach to solve for continuous-state reinforcement learning problems. By utilizing the inductive bias that actions have similar effect in similar states their algorithm derives the probability distribution through Gaussian weighting of similar states. Our approach differs from theirs by deriving the successor states while taking the actions performed in similar states into account. The probability distribution is then determined by a Hidden Markov Model.

7 Conclusion and Future Work

In this paper, we presented CASML, a model learner for continuous states and actions that generalizes from a small set of training instances. The model learner integrates a case base reasoner and a Hidden Markov Model to derive successor state probability distributions for unseen states and actions. Our experiments demonstrate that the learned model effectively expresses the system dynamics of a humanoid robot. All the results indicate that the model learned with CASML was able to approximate the desired trajectory quite accurately.

In future work we are planning to add a policy for revising cases in the model's case base to account for the innate bias towards the initially seen instances. We believe this will lead to an improvement in the case base's performance since fewer cases will be required to achieve good generalization. In addition we plan on evaluating the performance of the model learner in combination with a value iteration planner for generating optimal policies in continuous state and action spaces.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7(1):39–59.

Deisenroth, M., and Rasmussen, C. E. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the International Conference on Machine Learning (ICML-11)*, 465–472.

Diuk, C.; Li, L.; and Leffler, B. R. 2009. The adaptive k-meteorologists problem and its application to structure learning and feature selection in reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 249–256. ACM.

Hester, T., and Stone, P. 2009. Generalized model learning for reinforcement learning in factored domains. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 717–724. International Foundation for Autonomous Agents and Multiagent Systems.

Hester, T., and Stone, P. 2012. Learning and using models. In *Reinforcement Learning*. Springer. 111–141.

Jong, N. K., and Stone, P. 2007. Model-based exploration in continuous state spaces. In *Abstraction, Reformulation, and Approximation*. Springer. 258–272.

Kolobov, A. 2012. Planning with Markov decision processes: An AI perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6(1):1–210.

Kuvayev, L., and Sutton, R. S. 1996. Model-based reinforcement learning with an approximate, learned model. In *in Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*. Citeseer.

Molineaux, M.; Aha, D. W.; and Moore, P. 2008. Learning continuous action models in a real-time strategy environment. In *FLAIRS Conference*, volume 8, 257–262.

Ormoneit, D., and Sen, . 2002. Kernel-based reinforcement learning. *Machine Learning* 49(2-3):161–178.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to reinforcement learning*. MIT Press.

Van Hasselt, H. 2012. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*. Springer. 207–251.