Continuous-Domain Reinforcement Learning Using a Learned Qualitative State Representation[*]

Jonathan Mugan and Benjamin Kuipers
Computer Science Department
University of Texas at Austin
Austin Texas, 78712 USA
{jmugan,kuipers}@cs.utexas.edu

Abstract

We present a method that allows an agent to learn a qualitative state representation that can be applied to reinforcement learning. By exploring the environment the agent is able to learn an abstraction that consists of landmarks that break the space into qualitative regions, and rules that predict changes in qualitative state. For each predictive rule the agent learns a context consisting of qualitative variables that predicts when the rule will be successful. The regions of this context in with the rule is likely to succeed serve as a natural goals for reinforcement learning. The reinforcement learning problems created by the agent are simple because the learned abstraction provides a mapping from the continuous input and motor variables to discrete states that aligns with the dynamics of the environment.

Introduction

Reinforcement learning in continuous domains is difficult because the agent is unable to gain experience at each individual state. This means that the agent must use an abstraction that allows it to map an infinite number of input and motor states into a manageable number of abstracted states. To be useful to the agent, the abstraction must discriminate states that are different, but if the abstraction makes too many unnecessary discriminations then learning becomes inefficient. This balance is often achieved by having a human create and tune the abstraction.

Our approach to this problem is to use a qualitative state representation. In (Mugan & Kuipers 2007a; 2007b), we showed how an agent could build a qualitative representation of its environment that is not specific to any particular goal. The agent does this by breaking the world up into qualitative regions using *landmarks* and then learning *predictive rules* over changes in qualitative state.

In our approach, the agent experiences the world through a set of continuous input and motor variables. The motor variables and the derivatives of the input variables have an intrinsic landmark at 0, which creates three qualitative states for each of those variables. The continuous input variables themselves have no intrinsic landmarks, and the agent must learn landmarks on these variables as well as additional landmarks on the motor variables. A change in the qualitative state of a variable defines an *event*. The agent searches for rules that predict when one event will follow another in time. For each learned predictive rule the agent searches for regions in the state space where that rule will be reliable and delimits those regions by creating new landmarks. Each new landmark defines new events, which make it possible to learn new predictive rules, and so on.

In this paper we show that this learned qualitative representation enables the agent to do reinforcement learning to perform a simple task. The agent defines its own reinforcement learning problems using the learned predictive rules and landmarks. There has been previous work on enabling an agent to define its own reinforcement learning problems. McGovern and Barto (2001) have proposed a method whereby an agent autonomously finds subgoals based on bottleneck states that are visited often during successful trials. Subgoals have also been found by searching for "access states" (Simsek & Barto 2004; Simsek, Wolfe, & Barto 2005) that allow the agent to go from one part of the state space to another. In this paper we use a different approach to identifying goals for reinforcement learning, we define the goals for reinforcement learning to be the regions defined by landmarks in which a predictive rule is reliable. Additionally, there has been work on learning qualitative values given a model so that the level of abstraction is appropriate for the task (Sachenbacher & Struss 2005). Our work differs from this work because our focus is on learning the model and the abstraction simultaneously.

In the remainder of this paper we first give an overview of the qualitative abstraction and rule learning framework. We then explain how we incorporate reinforcement learning into this framework. Finally, we evaluate the viability of this approach by comparing it to a standard method for reinforcement learning with continuous variables on a simple task.

[*]This work has taken place in the Intelligent Robotics Lab at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Intelligent Robotics lab is supported in part by grants from the Texas Advanced Research Program (3658-0170-2007), from the National Science Foundation (IIS-0413257, IIS-0713150, and IIS-0750011), and from the National Institutes of Health (EY016089).

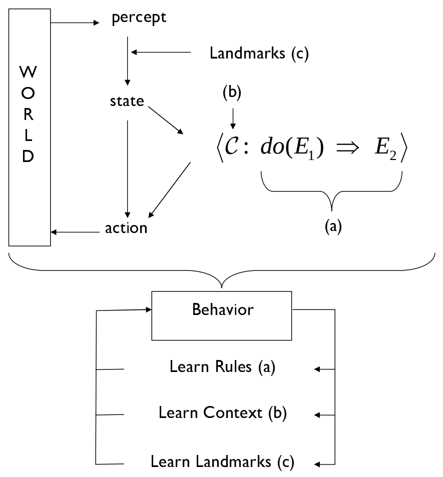


Figure 1: (a) The agent interacts with the world to learn rules stating that when the agent makes one event E_1 occur, that another event E_2 occurs. (b) For each rule the agent learns a context $\mathcal{C}$ that consists of a set of variables upon which the agent can learn a conditional probability table $CPT(r) = Pr(succeeds(r)|\mathcal{C})$ that tells the agent in which situations it can cause E_2 by making E_1 occur. (c) The agent also learns landmarks that determine how it can perceive and reason about the world. Landmarks are proposed based on the behavior of rules and in turn determine which rules can be learned (Mugan & Kuipers 2007a).

Learning Agent Architecture

The overview of the learning agent architecture is shown in Figure 1.

Qualitative State Representation

The raw input and output is represented using three types of variables: continuous motor variables, continuous input variables, and nominal input variables. Internally, the agent represents these variables qualitatively based on QSIM (Kuipers 1994). The agent creates two variables for each continuous input variable $\tilde{v}$, a discrete variable $v(t)$ that represents the magnitude of $\tilde{v}(t)$, and a discrete variable $\dot{v}(t)$ that represents the direction of change of $\tilde{v}(t)$. And for each continuous motor variable $\tilde{v}$, the agent creates a discrete magnitude variable $v(t)$. The result of this is that the agent represents its world using four types of discrete (qualitative) variables: motor variables, magnitude variables, direction of change variables, and nominal variables.

We now describe how the agent converts continuous values to qualitative values. A continuous variable $\tilde{v}(t)$ ranges over some subset of the real number line $(-\infty, +\infty)$. In QSIM, this continuous variable $\tilde{v}(t)$ is abstracted to a discrete variable $v(t)$ that ranges over a *quantity space* $\mathcal{Q}(v)$ of qualitative values. $\mathcal{Q}(v) = L(v) \cup I(v)$, where $L(v) = \{v_1^*, \cdots v_n^*\}$ is a totally ordered set of landmark values, and $I(v) = \{(-\infty, v_1^*), (v_1^*, v_2^*), \cdots (v_n^*, +\infty)\}$ is the set of mutually disjoint open intervals that $L(v)$ defines in the real number line. A quantity space with two landmarks might be described by (v_1^*, v_2^*), which implies five distinct qualitative values, $\mathcal{Q}(v) = \{(-\infty, v_1^*), v_1^*, (v_1^*, v_2^*), v_2^*, (v_2^*, +\infty)\}$.

Each direction of change variable $\dot{v}$ has a single intrinsic landmark at 0, so its quantity space is $\mathcal{Q}(\dot{v}) = \{(-\infty, 0), 0, (0, +\infty)\}$. Magnitude variables initially have no landmarks because zero is just another point on the number line, although landmarks are acquired as the agent learns. Initially, when the agent knows of no meaningful qualitative distinctions among values for $\tilde{v}(t)$, we describe the quantity space as the empty list of landmarks, $()$. Motor variables are given an initial landmark at 0, and like magnitude variables, they can acquire more landmarks as the agent learns. As an implementation note, because we evaluate the algorithm with a fine-grained discrete-timestep simulator, if v_1^* is a landmark and $\tilde{v}(t-1) < v_1^*$ and $\tilde{v}(t) > v_1^*$ then $v(t) = v_1^*$ for the purpose of rule learning.

Events

If a is a qualitative value of a discrete variable A, meaning $a \in \mathcal{Q}(A)$, then the *event* $A_t{\rightarrow}a$ is defined by $A(t-1) \neq a$ and $A(t) = a$. That is, an event takes place when a discrete variable A changes to value a at time t, from some other value. We will often drop the t and describe this simply as $A{\rightarrow}a$. We will also refer to an event as E when the variable and qualitative value involved are not important.

Predictive Rules

We break from previous work (2007a; 2007b) and take inspiration from (Pearl 2000) to define predictive rules based on actions of the agent. A *predictive rule* r has the form $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2 \rangle$ and states that if the agent executes a plan to bring about event E_1, then event E_2 will soon follow. The probability that E_2 will indeed soon follow E_1 is given in the context $\mathcal{C}$. For an event E we define $do_t(E)$ as a predicate that is true when the agent begins executing a plan at time t to bring about E and is false otherwise.

The predictive rule $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2 \rangle$ consists of one event $E_1(t)$, and another event $E_2(t')$ that takes place relatively soon after t. That E_2 takes place "relatively soon after" $E_1(t)$ is formalized in terms of an integer time-delay k (in our current implementation $k = 5$, or about 0.25 seconds)

$$soon(t, E_2) \quad \equiv \quad \exists t' \; [t \leq t' \leq t + k \; \wedge \; E_2(t')]$$

If we define $do_{r,t}(E(t'))$ to mean that $do_t(E) = \texttt{true}$ and that E occurs at time t', then we can define a predicate $succeeds(r, t)$ for the success of rule r as

$$succeeds(r, t) \quad \equiv \quad do_{r,t}(E_1(t')) \; \wedge \; soon(t', E_2)$$

This means that rule r fails if the agent's plan to bring about E_1 fails, or if E_2 does not soon follow. We shorten $succeeds(r, t)$ to the predicate $succeeds(r)$, which is true if r succeeds when activated at an arbitrary time.

Associated with rule r is a context $\mathcal{C}$ that consists of a set of variables. The context induces a conditional probability table (CPT) on the predicate $succeeds(r)$. In Bayesian network terminology, the variables in $\mathcal{C}$ are the parents of $succeeds(r)$. For a rule $r = \langle \mathcal{C} : do(A \to a) \Rightarrow B \to b \rangle$ we require that the elements of the context be magnitude or nominal variables and that for event $B \to b$ we require that B be a nominal variable or that B be a direction of change variable with $b \neq [0]$.

Learning New Predictive Rules

To learn a new predictive rule the agent searches for two events E_1 and E_2 such that observing event E_1 means that event E_2 is significantly more likely to occur than it would have been otherwise. When two such events are found the agent asserts an initial rule $\langle \emptyset : do(E_1) \Rightarrow E_2 \rangle$ with an empty context.

The set of rules grows out of the motor variables. To create a rule $\langle \emptyset : do(E_1) \Rightarrow E_2 \rangle$ we require that the agent be able to predict event E_1 using the currently existing rules.

Rule Context Greedy Search

The purpose of the context is to tell the agent when the rule will succeed and the agent greedily searches for a good context for each rule. A good context $\mathcal{C}$ for r is one for which there is some value for the variables in $\mathcal{C}$ for which the rule has high reliability. Once this is achieved, the agent desires that the context predict the outcome of the rule in all states.

We formalize the idea of having a value for which the rule is highly reliable using the notation of best reliability $brel(r)$. For a context $\mathcal{C} = \{v_1, \ldots, v_n\}$, we define the product space of qualitative values:

$$\mathcal{Q}(\mathcal{C}) = \mathcal{Q}(v_1) \times \mathcal{Q}(v_2) \times \ldots \times \mathcal{Q}(v_n). \qquad (1)$$

With sufficient observations, we then define best reliability as the maximum over this product space

$$brel(r) = \max_{w \in \mathcal{Q}(\mathcal{C})} Pr(succeeds(r)|w) \qquad (2)$$

which we can also write as $brel(r) = \max\ CPT(r)$.

Once $brel(r)$ exceeds the threshold $\theta_r = 0.7$ we determine that the rule is reliable. At this point the agent turns its attention to being able to predict the outcome of a rule in any situation. To do this it seeks to minimize the entropy. The *entropy* $H(Y)$ of a random variable is given by

$$H(Y) = -\sum_j P(Y = y_j) \log_2 P(Y = y_j).$$

The conditional entropy $H(Y|X)$ of a random variable Y given X is given by

$$H(Y|X) = \sum_i H(Y|X = x_i) P(X = x_i)$$

and is the weighted average of the entropy of Y given $X = x_i$, weighted by the probability $P(X = x_i)$. We define the entropy $H(r)$ of a rule $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2 \rangle$ as the conditional entropy of $succeeds(r)$ given $do(E_1) = \mathtt{true}$ and $\mathcal{C}$. In equation form it is

$$H(r) = H(succeeds(r)|do(E_1) = \mathtt{true}, \mathcal{C}). \qquad (3)$$

With these definitions we can now describe how the agent determines if one context is better than another. For each rule the agent hillclimbs on the quality of the context. For a rule $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2 \rangle$ with $brel(r) < \theta_r$ we say that the rule $r' = \langle \mathcal{C}' : do(E_1) \Rightarrow E_2 \rangle$ with improved context $\mathcal{C}'$ is a *sufficient improvement* over r if $brel(r') >> brel(r)$. And for a rule $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2 \rangle$ with $brel(r) > \theta_r$ we say that the rule $r' = \langle \mathcal{C}' : do(E_1) \Rightarrow E_2 \rangle$ with improved context $\mathcal{C}'$ is a *sufficient improvement* over r if $H(r') << H(r)$, where the operators $>>$ and $<<$ mean sufficiently less than and sufficiently greater than, respectively.

Learning a Context for a Predictive Rule

The context for a predictive rule is learned incrementally. For each rule $r = \langle \emptyset : do(E_1) \Rightarrow E_2 \rangle$ with an empty context, the agent searches for a magnitude or nominal variable v_1 such that if r is modified to be $r' = \langle \{v_1\} : do(E_1) \Rightarrow E_2 \rangle$ then r' is a sufficient improvement.

Using an approach inspired by Drescher (1991), once the agent has learned a rule $r' = \langle \{v_1\} : do(E_1) \Rightarrow E_2 \rangle$ it searches for another discrete magnitude or nominal variable v_2 such that if r' is modified to be $r'' = \langle \{v_1, v_2\} : do(E_1) \Rightarrow E_2 \rangle$ then r'' is a sufficient improvement. This criterion clearly generalizes, but in our current implementation we limit the size of the context to two.

Learning New Landmarks

Inserting a new landmark x^* into (x_i^*, x_{i+1}^*) allows that interval to be replaced in $\mathcal{Q}(x)$ by two intervals and the dividing landmark: (x_i^*, x^*), x^*, (x^*, x_{i+1}^*). Adding this new landmark into the quantity space $\mathcal{Q}(x)$ allows a new distinction to be made that may transform a rule r into a new rule r'. A new landmark can be learned either by improving a predictive rule or by creating an event that reliably precedes another event.

Landmarks that Improve Rules Landmark candidates are generated for a rule $r = \langle \mathcal{C} : do(A \to a) \Rightarrow B \to b \rangle$ using the success or failure of r as a reward signal. A landmark candidate for r is adopted if it sufficiently improves r. A landmark can improve r by refining the event $A \to a$ or by refining a variable in $\mathcal{C}$.

To learn new landmarks it is not necessary to store the entire history. Instead, we only store the real values of all the variables for the last 200 activations of each rule. Landmark candidates are chosen considering the number of data points in the interval and the highest gain (Fayyad & Irani 1993). Depending on the distance from the new landmark x^* to the maximum and minimum observed values of x, this search can result in either a precise numerical value, or a range of possible values for x^* on different occasions: $range(x^*) = [lb, ub]$.

Landmarks Suggested by Events A landmark x^* is created for a variable x if it is estimated that the event $x \to x^*$ will reliably predict some other event E. To find this landmark, for each event E a histogram is maintained for each continuous variable $\tilde{x}$. Each time E occurs the histogram is updated with the current value of $\tilde{x}$. One or more landmark candidates is created for $\tilde{x}$ when the distribution of $\tilde{x}$ when

E occurs is significantly different from the background distribution of $\tilde{x}$. The location of each landmark x^* is taken to be the middle of a histogram bucket where the difference is the greatest.

Acting in the World

The Controller

The controller enables the agent to learn efficiently by actively choosing rules to test. In Mugan and Kuipers (Mugan & Kuipers 2007a) active learning was motivated by the desire to achieve certain goals, in this paper the motivation for active learning is improving the reliability of rules.

Choosing a Rule to Invoke The controller chooses a rule to invoke based on its weight w. The weight of a rule r consists of two components w_1 and w_2, and $w = max(\epsilon, w_1 w_2)$, where $\epsilon = 0.001$. If we use the notation $Pr(succeeds(r)|\mathcal{C}, s)$ to mean the probability of success of r in the current state s, then the component $w_1 = Pr(succeeds(r)|\mathcal{C}, s)$. The component w_2 reflects the rate at which the reliability of the rule is increasing, inspired by the "curiosity drive" of Oudeyer and Kaplan [2004].

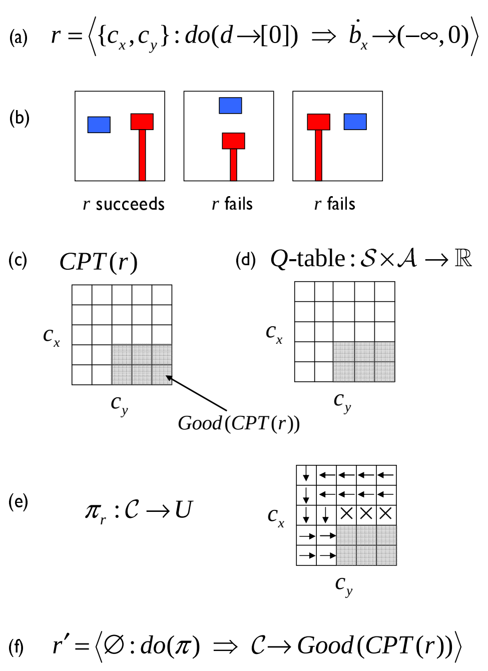
Invoking a Rule Once the rule $r = \langle \mathcal{C} : do(A \rightarrow a) \Rightarrow B \rightarrow b \rangle$ has been chosen, the agent forms a plan to achieve $A \rightarrow a$. To do this, the controller examines the context $\mathcal{C}$. We say that the context is *satisfied* if in the current state the context says the rule will be *sufficiently reliable*, where sufficiently reliable means that $Pr(succeeds(r)|\mathcal{C}) > \theta_{sr}$, where $\theta_{sr} = 0.60$. There are three cases:

1. The context is satisfied.
2. The context is not satisfied and consists of only one variable.
3. The context is not satisfied and consists of more than one variable.

If the context is satisfied, the agent sets the goal to be $do(A \rightarrow a)$. If the context is not satisfied but consists of only a single variable V, then the agent sets the goal to be $do(V \rightarrow v)$ where $v \in \mathcal{Q}(V)$ has the highest $Pr(succeeds(r)|V = v)$. If $do(V \rightarrow v)$ is successful, the agent then sets the goal to be $do(A \rightarrow a)$. If the context is not satisfied and consists of more than one variable, then the agent sets the goal to be any member of the set $Good(CPT(r))$ defined in equation (4) (if the set $Good(CPT(r))$ is empty then the context is ignored). Once this goal is achieved the agent sets the goal to be $do(A \rightarrow a)$.

Backchaining Actions

Goals of the form $do(Y \rightarrow y)$ are achieved through backchaining. The approach to achieve a goal $do(Y \rightarrow y)$ depends on the type of variable Y. **(1)** If Y is a motor variable, then a random real value is picked from the range of the qualitative value y and the action is complete. **(2)** If Y is a direction of change or nominal variable, then the agent looks for a reliable rule of the form $r = \langle \mathcal{C} : do(X \rightarrow x) \Rightarrow Y \rightarrow y \rangle$ that in the current state is predicted to succeed with reliability θ_{sr} and invokes $do(X \rightarrow x)$. If no such rule is found



(a) $r = \langle \{c_x, c_y\} : do(d \rightarrow [0]) \Rightarrow \dot{b}_x \rightarrow (-\infty, 0) \rangle$
(b)
r succeeds
r fails
r fails
(c) $CPT(r)$
c_x
c_y
$Good(CPT(r))$
(d) Q-table$: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
c_x
c_y
(e) $\pi_r : \mathcal{C} \rightarrow U$
c_x
c_y
(f) $r' = \langle \varnothing : do(\pi) \Rightarrow \mathcal{C} \rightarrow Good(CPT(r)) \rangle$

Figure 2: (a) The rule $r = \langle \{c_x, c_y\} : do(d \rightarrow [0]) \Rightarrow \dot{b}_x \rightarrow (-\infty, 0) \rangle$ is an example of a rule learned by the robot. It states that if the distance d between the hand and the block goes to 0, then the event $\dot{b}_x \rightarrow (-\infty, 0)$ of the block moving to the left will occur. The predicted success of this rule depends on the context variables c_x and c_y that give the location of the hand in the frame of reference of the block. (b) The agent gathers experience in the world to learn the context values for which r is successful. The agent learns that the hand must be to the right of and level with the block for r to be successful. (c) Based on $\mathcal{C} = \{c_x, c_y\}$ the agent creates a conditional probability table $CPT(r)$ for r and uses a threshold to determine the set $Good(CPT(r))$ of values of $\mathcal{C}$ for which the rule r is likely to succeed. (d) The agent can then define a simple reinforcement learning problem in which $\mathcal{C}$ defines the state space, and $Good(CPT(r))$ defines the goal states. The agent is rewarded for reaching a state in which the rule r is likely to succeed. To do this, the agent creates a Q-table that maps $\mathcal{S} \times \mathcal{A}$ to a value $\mathbb{R}$, where $\mathcal{A}$ is the set of primitive actions (defined by the qualitative values of the motor variables u_x and u_y). (e) The agent then defines a policy π_r by associating each cell in $\mathcal{C}$ with the primitive action with maximum value. (f) The policy π_r can then be described by a new rule r' that treats π_r as an action leading to the region $Good(CPT(r))$ where r is likely to succeed.

or if r fails, then backchaining fails. **(3)** If Y is a magnitude variable then the agent uses a special rule of the form

$h_1 = \langle do(\dot{Y} \rightarrow (0, +\infty))$ until $Y \rightarrow y\rangle$ if $\tilde{Y}(t) < y$ and $h_2 = \langle do(\dot{Y} \rightarrow (-\infty, 0))$ until $Y \rightarrow y\rangle$ if $\tilde{Y}(t) > y$. Rule h_1 fails if $do(\dot{Y} \rightarrow (0, +\infty))$ is not achieved or if after $\dot{Y} \rightarrow (0, +\infty)$ an event occurs such that $\dot{Y} \neq (0, +\infty)$ before $Y \rightarrow y$. Rule h_2 works similarly. If during backchaining an event E occurs more than once, or if events $v \rightarrow (-\infty, 0)$ and $v \rightarrow (0, +\infty)$ both occur for some variable v, then backchaining fails.

Once a motor variable is reached, its value is maintained until event $Y \rightarrow y$ occurs or one of the rules used in backchaining fails.

Reinforcement Learning Actions

The agent uses reinforcement learning to achieve goals over multiple variables. For each rule $r = \langle \mathcal{C} : do(E_1) \Rightarrow E_2\rangle$ with a context of more than one variable, the agent creates a reinforcement learning problem to enable the agent to get into a state such that doing event E_1 will cause event E_2. The overview of this process is shown in Figure 2.

The type of reinforcement learning we use is Sarsa(λ) (Sutton & Barto 1998) where $\lambda = 0.9$, α is one over the number of times the state has been visited, and the discount parameter $\gamma = 0.9$. To learn the policy π_r the agent learns a value-action function $Q : \mathcal{S}, \mathcal{A} \rightarrow \mathbb{R}$.

The state space $\mathcal{S}$ is defined by the qualitative variables in $\mathcal{C}$ and their landmarks. To define the set of primitive actions $\mathcal{A}$ we first define a set $\mathcal{Q}(U) = \mathcal{Q}(u_1) \times \ldots \times \mathcal{Q}(u_n)$ where $u_1, \ldots, u_n$ is the set of motor variables. We can then define a primitive action $a \in \mathcal{A}$ as choosing a $w \in \mathcal{Q}(U)$, taking random values from the ranges of the qualitative values in w, and maintaining those values until the state $\mathcal{S}$ changes or the real values underlying the variables that make up $\mathcal{S}$ stop changing.

For the reward function we use a goal-reward representation (Koenig & Simmons 1996). The reward is based on the set of goal states $Good(CPT(r))$ and is determined by the conditional probability table $CPT(r)$:

$$Good(CPT(r)) = \{w \in \mathcal{Q}(\mathcal{C}) \quad | Pr(succeeds(r)|w) > \theta_{sr}\} \tag{4}$$

The agent then learns the Q-table by using ϵ-greedy action selection where $\epsilon = 0.25$. An episode begins when the rule r is invoked by the controller, and the episode ends when the agent makes it to a goal state or when 20 primitive actions have been taken.

Once the Q-table is learned, a policy π_r can be created whereby the agent chooses the best primitive action for each state. In effect, this in principle leads to a new rule of the form $r' = \langle \emptyset : do(\pi_r) \Rightarrow \mathcal{C} \rightarrow Good(CPT(r))\rangle$.

Experimental Evaluation

We evaluate our algorithm using the simulated agent shown in Figure 3. The evaluation task we have chosen is for the agent to hit the block in a specified direction. To show that our representation can effectively be used for reinforcement learning, we compare our method to using a hand-created reinforcement learning agent trained specifically for this task. For this evaluation we trained ten agents total, five autonomous agents described in this paper, and five hand-created learning agents.

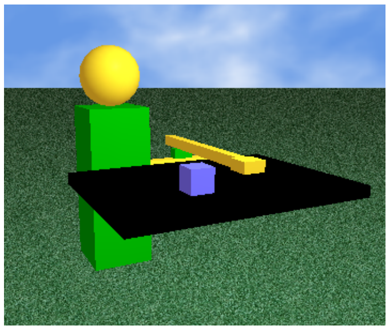


Figure 3: A simulated "robot baby" is implemented in Breve (Klein 2003). It has a torso with a 2-dof orthogonal arm and is sitting in front of a tray with a block. The robot has two motor variables $\tilde{u}_x$ and $\tilde{u}_y$ that move the hand in the x and y directions, respectively. The perceptual system creates variables for each of the two tracked objects in this environment: the hand and the block. The hand is described by two continuous variables $\tilde{h}_x(t)$, $\tilde{h}_y(t)$ that represent the location of the hand in the x and y directions, respectively, and the Boolean variable $h_a(t)$ that represents whether the hand is in view. The variables corresponding to the block are $\tilde{b}_x(t)$, $\tilde{b}_y(t)$, and $b_a(t)$ and they have the same respective meanings as the variables for the hand. The relationship between the hand and the block is represented by the continuous variables $\tilde{c}_x(t)$, $\tilde{c}_y(t)$, and $\tilde{d}(t)$. The variables $\tilde{c}_x(t)$ and $\tilde{c}_y(t)$ represent the coordinates of the center of the hand in the frame of reference of the center of the block, and the variable $\tilde{d}(t)$ represents the distance between the hand and the block. The values of all variables are updated by perceptual trackers at each timestep as the objects move.

We trained each agent in the environment shown in Figure 3 for 340,000 timesteps (almost five hours of physical experience). During this time, the hand-created agents continually repeated episodes of the task, and the autonomous agents performed the learning algorithm described in this paper. During training of the autonomous agents, if the block fell off the tray, moved out of reach of the agent, or was not moved for an extended time, the block was moved to a random location within reach of the agent. For all agents we stored the state of each agent's knowledge every 20,000 timesteps during training (corresponding to about sixteen minutes of physical experience). We then ran the evaluation for each agent using their respective stored knowledge bases.

Each evaluation consisted of 100 trials. At the beginning of each trial the block was placed in a random location within reach of the agent and the evaluator picked one of three goals: hitting the block to the left, hitting the block to the right, or hitting the block forward. The agent then had 300 timesteps to use its knowledge to hit the block in the correct direction. A trial was terminated unsuccessfully if the agent hit the block in the wrong direction. The evaluation metric was the success rate for hitting the block during the 100 trials.

We tested both types of agents under two goal selection regimes, *uniform* and *hard*. During both uniform and hard goal selection, the evaluator selects the goal randomly, with a uniform distribution, and filters out goals that are impossible to achieve. (For example, if the block is on the far left, the agent cannot get its hand on the left side of the block to move it to the right.) During *hard* goal selection, *easy* goals are also filtered out, where a goal is easy if it can be achieved with a single straight-line motion. (During the training period for the hand-created agents, a goal selection regime was randomly chosen at the beginning of each episode, and then the goal was chosen based on that regime.)

The Hand-Built Learner

The hand-built reinforcement learning agents used linear, gradient-descent Sarsa(λ) with binary features (Sutton & Barto 1998) where the binary features come from tile coding. We chose this method because tile coding is a standard method for coping with continuous variables in reinforcement learning (Santamaria, Sutton, & Ram 1997). Tile coding works by using multiple partitions of the state space such that each partition (tiling) is offset just a little from the others. This allows the agent to generalize more effectively than using a single partition with higher resolution.

We now explain the details of the tile coding implementation. The motor variables u_x and u_y were each divided into 10 equal-width bins, and the direction of change variables were each divided into 3 bins: $(-\infty, -0.05), [-0.05, 0.05], (0.05, \infty)$. The goal was represented with a discrete variable that took on three values, one for each of the three goals. The remaining variables were treated as continuous. There were 16 tilings, the tiling was done using a hashing function with a memory size of 65,536. The parameter values used were $\lambda = 0.9$, $\gamma = 0.9$, and $\alpha = 0.1$. To prevent the task diameter from being too high, during both training and testing the agent chose a new action every 10 timesteps (0.5 seconds). Action selection was ϵ-greedy where $\epsilon = 0.05$.

Results

The results are shown in Figure 4. As the agent gains more experience in the world its ability to perform the task improves. We also see that the agent has indeed learned the action as its performance under both the difficult and uniform task selection regime is comparable to that of the hand-created learner.

The hand-created learner enjoys the advantage of only being trained on the evaluation task. But the hand-created learner is at an important disadvantage, it does not know which variables are important for the task. Our agent learns which variables are important autonomously, and this allows it to perform comparably even though it learns more than how to perform the evaluation task. For example, our agent learns when the block will disappear off the tray. It also learns the limits of its movement, and it can also move away from the block instead of towards it. It can use the knowledge learned during one task to learn another. Of particular importance is that the agent learns a discretization of the action space. Each motor variable is initially given a landmark at 0, but it takes a force of 300 in the simulator to move the arm in any direction. Our agent finds those important landmarks and can then use that knowledge when learning a new task. In contrast, the hand-created learner would need to be trained from scratch for each new task and would not be able to use what it learned in previous tasks for future tasks.

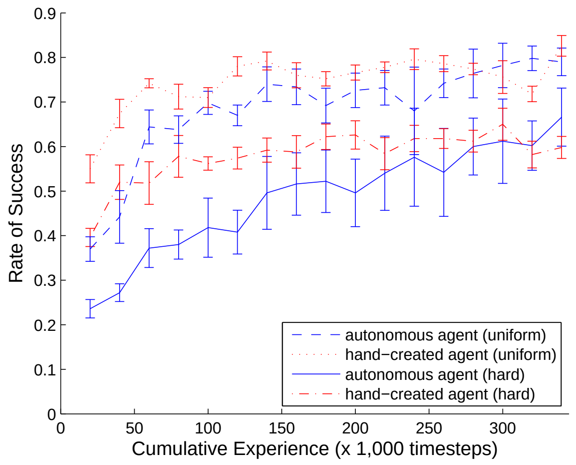


Figure 4: Both the autonomous agent and the hand-created agent improve with experience. The hand-created agent training only on the task initially has better performance, but the autonomous agent later matches it. All error bars are standard error.

Conclusion and Future Work

The agent begins with a simple set of qualitative distinctions. These distinctions allow it to learn predictive rules. By monitoring the success or failure of these rules, the agent is able to find natural joints (landmarks) in its environment that allow it to discretize its continuous input and motor variables. Using the rules and the discretization of variables, the agent is able to define many small reinforcement learning problems. These reinforcement learning problems lead to policies that allow the agent to move to the regions of the state space that maximize the reliability of its learned rules.

In future work we will focus on generalizing the notion of a rule to include policies learned during reinforcement learning as shown in Figure 2 (f). This will allow the agent learn when such policies will be successful and when they will not. We are also moving towards implementing this on a physical robot using a camera that watches an arm in a workspace.

References

Drescher, G. L. 1991. *Made-Up Minds: A Constructivist Approach to Artificial Intelligence*. Cambridge, MA: MIT Press.

Fayyad, U. M., and Irani, K. B. 1993. Multi-interval discretization of continuousvalued attributes for classification learning. In *Proceedings International Joint Conference on Articial Intelligence*, volume 2, 1022–1027.

Klein, J. 2003. Breve: a 3d environment for the simulation of decentralized systems and artificial life. In *Proceedings of the International Conference on Artificial Life*, 329–334.

Koenig, S., and Simmons, R. 1996. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning* 22(1):227–250.

Kuipers, B. 1994. *Qualitative Reasoning*. Cambridge, Massachusetts: The MIT Press.

McGovern, A., and Barto, A. G. 2001. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings International Conference on Machine Learning*, 361–368.

Mugan, J., and Kuipers, B. 2007a. Learning distinctions and rules in a continuous world through active exploration. In *Proceedings of the International Conference on Epigenetic Robotics*.

Mugan, J., and Kuipers, B. 2007b. Learning to predict the effects of actions: Synergy between rules and landmarks. In *Proceedings of the International Conference on Development and Learning*.

Oudeyer, P.-Y., and Kaplan, F. 2004. Intelligent adaptive curiosity. In *Proceedings of the International Conference on Epigenetic Robotics*.

Pearl, J. 2000. *Causality: Modeling, Reasoning, and Inference*. Cambridge: Cambridge University Press.

Sachenbacher, M., and Struss, P. 2005. Task-dependent qualitative domain abstraction. *Artificial Intelligence* 162(1-2):121–143.

Santamaria, J.; Sutton, R.; and Ram, A. 1997. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior* 6(2):163.

Simsek, O., and Barto, A. 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. *Proceedings of the Twenty-First International Conference on Machine Learning* 751–758.

Simsek, O.; Wolfe, A.; and Barto, A. 2005. Identifying useful subgoals in reinforcement learning by local graph partitioning. *Proceedings of the Twenty-Second International Conference on Machine Learning* 816–823.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. Cambridge MA: MIT Press.