Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction

Richard S. Sutton, Joseph Modayil, Michael Delp
Thomas Degris, Patrick M. Pilarski, Adam White
Reinforcement Learning and Artificial Intelligence Laboratory
Department of Computing Science, University of Alberta, Canada

Doina Precup
School of Computer Science, McGill University, Montreal, Canada

ABSTRACT

Maintaining accurate world knowledge in a complex and changing environment is a perennial problem for robots and other artificial intelligence systems. Our architecture for addressing this problem, called *Horde*, consists of a large number of independent reinforcement learning sub-agents, or *demons*. Each demon is responsible for answering a single predictive or goal-oriented question about the world, thereby contributing in a factored, modular way to the system's overall knowledge. The questions are in the form of a value function, but each demon has its own policy, reward function, termination function, and terminal-reward function unrelated to those of the base problem. Learning proceeds in parallel by all demons simultaneously so as to extract the maximal training information from whatever actions are taken by the system as a whole. Gradient-based temporal-difference learning methods are used to learn efficiently and reliably with function approximation in this off-policy setting. Horde runs in constant time and memory per time step, and is thus suitable for learning online in real-time applications such as robotics. We present results using Horde on a multi-sensored mobile robot to successfully learn goal-oriented behaviors and long-term predictions from off-policy experience. Horde is a significant incremental step towards a real-time architecture for efficient learning of general knowledge from unsupervised sensorimotor interaction.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Algorithms, Experimentation

Keywords

artificial intelligence, knowledge representation, robotics, reinforcement learning, off-policy learning, real-time, temporal-difference learning, value function approximation

Cite as: Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction, Richard S. Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M. Pilarski, Adam White, and Doina Precup, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 761-768.

1. THE PROBLEM OF EXPRESSIVE AND LEARNABLE KNOWLEDGE

How to learn, represent, and use knowledge of the world in a general sense remains a key open problem in artificial intelligence (AI). There are high-level representation languages based on first-order predicate logic and Bayes networks that are very expressive, but in these languages knowledge is difficult to learn and computationally expensive to use. There are also low-level languages such as differential equations and state-transition matrices that can be learned from data without supervision, but these are much less expressive. Knowledge that is even slightly forward looking, such as 'If I keep moving, I will bump into something within a few seconds' cannot be expressed directly with differential equations and may be expensive to compute from them. There remains room for exploring alternate formats for knowledge that are expressive yet learnable from unsupervised sensorimotor data.

In this paper we pursue a novel approach to knowledge representation based on the notion of value functions and on other ideas and algorithms from reinforcement learning. In our approach, knowledge is represented as a large number of approximate value functions learned in parallel, each with its own policy, pseudo-reward function, pseudo-termination function, and pseudo-terminal-reward function. Learning systems using multiple approximate value functions of this type have previously been explored as temporal-difference networks with options (Sutton, Rafols & Koop 2006; Sutton, Precup & Singh 1999). Our architecture, called *Horde*, differs from temporal-difference networks in its more straightforward handling of state and function approximation (no predictive state representations) and in its use of more efficient algorithms for off-policy learning (Maei & Sutton 2010; Sutton et al. 2009). The current paper also extends prior work in that we demonstrate real-time learning on a physical robot.

Previous work on the problem of representing a general sense of knowledge while being grounded in and learnable from sensorimotor data goes back at least to Cunningham (1972) and Becker (1973). Drescher (1991) considered a simulated robot baby learning conditional probability tables for boolean events. Ring (1997) explored continual learning of a hierarchical representation of sequences. Cohen et al. (1997) explored the formation of symbolic fluents from simulated experience. Kaelbling et al. (2001) and Pasula et al. (2007)

explored the learning of relational rule representations in stochastic domains. All these systems involved learning significant knowledge but remained far from learning from sensorimotor data. Previous researchers who did learn from sensorimotor data include Pierce and Kuipers (1997), who learned spatial models and control laws, Oates et al. (2000), who learned clusters of robot trajectories, Yu and Ballard (2004), who learned word meanings, and Natale (2005), who learned goal-directed physical actions. All of these works learned significant knowledge but specialized on knowledge of a particular kind; the knowledge representation they used is not as general as that of multiple approximate value functions.

2. VALUE FUNCTIONS AS SEMANTICS

A distinctive, appealing feature of approximate value functions as a knowledge representation language is that they have an explicit semantics, a clear notion of truth grounded in sensorimotor interaction. A bit of knowledge expressed as an approximate value function is said to be true, or more precisely, *accurate*, to the extent that its numerical values match those of the mathematically defined value function that it is approximating. A value function asks a *question*—what will the cumulative future reward be?—and an approximate value function provides an *answer* to that question. The approximate value function is the knowledge, and its match to the value function—to the actual future reward—defines what it means for the knowledge to be accurate. The idea of the present work is that the value-function approach to grounding semantics can be extended beyond reward to a theory of all world knowledge. In this section we define these ideas formally for the case of reward and conventional value functions (and thereby introduce our notation), and in the next section we extend them to knowledge and general value functions.

In the standard reinforcement learning framework (Sutton & Barto 1998), the interaction between the AI agent and its world is divided into a sequence of discrete time steps, $t = 1, 2, 3, \ldots$, each corresponding perhaps to a fraction of a second. The state of the world at each step, denoted $S_t \in \mathcal{S}$, is sensed by the agent, perhaps incompletely, and used to select an action $A_t \in \mathcal{A}$ in response. One time step later the agent receives a real-valued reward $R_{t+1} \in \mathbb{R}$ and a next state $S_{t+1} \in \mathcal{S}$, and the cycle repeats. Without loss of significant generality, we can consider the rewards to be generated according to a deterministic *reward function* $r : \mathcal{S} \to \mathbb{R}$, with $R_t = r(S_t)$.

The focus in conventional reinforcement learning is on learning a stochastic action-selection *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that gives the probability of selecting each action in each state, $\pi(s, a) = \mathbb{P}(A_t = a | S_t = s)$. Informally, a good policy is one that results in the agent receiving a lot of reward summed over time steps. For example, in game playing the reward might correspond to points won or lost on each turn, and in a race the reward might be -1 on each time step. In *episodic* problems, the agent–world interaction consists of multiple finite trajectories (episodes) that can terminate in better or worse ways. For example, playing a game may generate a sequence of moves that eventually ends with a win, loss, or draw, with each outcome having a different numerical value, perhaps $+1$, -1 and 0. A race may be completed successfully or end in disqualification, two very different outcomes even if the number of seconds elapsed is the same. Another example is optimal control, in which it is common to have costs for each step (e.g., related to energy expenditure) plus a terminal cost (e.g., relating to how far the final state is from a goal state). In general, a problem may have both a reward function as already formulated and also a *terminal-reward function*, $z : \mathcal{S} \to \mathbb{R}$, where $z(s)$ is the terminal reward received if termination occurs upon arrival in state s.

We turn now to formalizing the process of termination. In many reinforcement learning problems, particularly non-episodic ones, it is common to give less weight to delayed rewards, in particular, to *discount* them by a factor of $\gamma \in [0, 1)$ for each step of delay. One way to think about discounting is as a constant probability of termination, of $1-\gamma$, together with a terminal reward that is always zero. More generally, we can consider there to be an arbitrary *termination function*, $\gamma : \mathcal{S} \to [0, 1]$, with $1 - \gamma(s)$ representing the probability of terminating upon arrival in state s, at which time a corresponding terminal reward of $z(s)$ would be registered. The overall *return*, a random variable denoted G_t for the trajectory starting at time t, is then the sum of the per-step rewards received up until termination occurs, say at time T, *plus* the final terminal reward received in S_T:

$$G_t \;=\; \sum_{k=t+1}^{T} r(S_k) \;+\; z(S_T). \qquad (1)$$

The conventional *action-value function* $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is then defined as the expected return for a trajectory starting from the given state and action and selecting actions according to policy π until terminating according to γ (thus determining the time of termination, T):

$$Q^\pi(s, a) \;=\; \mathbb{E}[G_t \mid S_t = s, A_t = a, A_{t+1:T-1} \sim \pi, T \sim \gamma] \,.$$

This expectation is well defined given a particular state-transition structure for the world (say as a Markov decision process). If an AI agent were to possess an approximate value function, $\hat{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, then it could be assessed for accuracy according to its closeness to Q^π, for example, according to the expectation of its squared error, $(Q^\pi(s, a) - \hat{Q}(s, a))^2$, over some distribution of state–action pairs. In practice it is rarely possible to measure this error exactly, but the value function Q^π still provides a useful theoretical semantics and ground truth for the knowledge $\hat{Q}$. The value function is the exact numerical answer to the precise, grounded question 'What would the return be from each state–action pair if policy π were followed?', and the approximate value function offers an approximate numerical answer. In this precise sense the value function provides a semantics for the knowledge represented by the AI agent's approximate value function.

Finally, we note that the value function for a policy is often estimated solely for the purpose of improving the policy. Given a policy π and its value function Q^π, we can construct a new deterministic *greedy* policy $\pi' = greedy(Q^\pi)$ such that $\pi'(s, \arg\max_a Q^\pi(s, a)) = 1$, and the new policy is guaranteed to be an improvement in the sense that $Q^{\pi'}(s, a) \geq Q^\pi(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, with equality only if both policies are optimal. Through successive steps of estimation and improvement, a policy that optimizes the expected return can be found. In this way the theory of value functions provides a semantics for goal-oriented knowledge (control) as well as for predictive knowledge.

3. FROM VALUES TO KNOWLEDGE (GENERAL VALUE FUNCTIONS)

Having made clear how a conventional value function provides a grounded semantics for knowledge about upcoming reward, in this section we show how *general value functions (GVFs)* provide a grounded semantics for a more general kind of world knowledge. Using the ideas and notation developed in the previous section, this is almost immediate.

First note that although the action-value function Q^π is conventionally superscripted only by the policy, it is equally dependent on the reward and terminal-reward functions, r and z. These functions could equally well have been considered inputs to the value function in the same way that π is. That is, we might have defined a more general value function, which might be denoted $Q^{\pi,r,z}$, that would use returns (1) defined with arbitrary functions r and z acting as *pseudo*-reward function and *pseudo*-terminal-reward function. For example, suppose we are playing a game, for which the base terminal rewards are $z = +1$ for winning and $z = -1$ for losing (with a per-step reward of $r = 0$). In addition to this, we might pose an independent question about how many more moves the game will last. This could be posed as a general value function with pseudo-reward function $r = 1$ and pseudo-terminal-reward function $z = 0$. Later in this paper we consider several more examples from a robot domain.

The second step from value functions to GVFs is to convert the termination function γ to a pseudo form as well. This is slightly more substantive because, unlike the rewards and terminal rewards, which do not pertain to the state evolution in any way, termination conventionally refers to an interruption in the normal flow of state transitions and a reset to a starting state or starting-state distribution. For pseudo termination we simply omit this additional implication of conventional termination. The real, base problem may still have real terminations or it may have no terminations at all. Yet we may consider pseudo terminations to have occurred at any time. For example, in a race, we can consider a pseudo-termination function that terminates at the half way point. This is a perfectly well defined problem with a value function in the general sense. Or, if we are the racer's spouse, then we may not care about when the race ends but rather about when the racer comes home for dinner, and that may be our pseudo termination. For the same world—the same actions and state transitions—there are many predictive questions that can be defined in the form of general value functions.

Formally, we define a *general value function*, or GVF, as a function $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with four auxiliary functional inputs π, γ, r, and z, defined over the same domains and ranges as specified earlier, but now taken to be arbitrary and with no necessary relationship to the base problem's reward, terminal-reward, and termination functions:

$$q(s, a; \pi, \gamma, r, z) = \mathbb{E}[G_t \mid S_t = s, A_t = a, A_{t+1:T-1} \sim \pi, T \sim \gamma] ,$$

where G_t is still defined by (1) but now with respect to the given functions. The four functions, π, γ, r, and z, are referred to collectively as the GVF's *question functions*; they define the question or semantics of the GVF. Note that conventional value functions remain a special case of GVFs. Thus, we can consider all value functions to be GVFs. In the rest of the paper, for simplicity, we sometimes use the expression "value function" to mean the general case, using "conventional value function" when needed to disambiguate. We also drop the 'pseudo-' prefix from the question functions when it can be done without ambiguity. In the robot experiments that we present later there are no privileged base problems, so there should be no confusion.

4. THE HORDE ARCHITECTURE

The Horde architecture consists of an overall agent composed of many sub-agents, called *demons*. Each demon is a independent reinforcement-learning agent responsible for learning one small piece of knowledge about the base agent's interaction with its environment. Each demon learns an approximation, $\hat{q}$, to the GVF, q, that corresponds to the demon's setting of the four question functions, π, γ, r, and z.

We turn now to describing Horde's mechanisms for approximating GVFs with a finite number of weights, and for learning those weights. In this paper we adopt the standard linear approach to function approximation. We assume that the world's state and action at each time step, S_t and A_t, are translated, presumably incompletely via sensory readings, into a fixed-size *feature vector* $\phi_t = \phi(S_t, A_t) \in \mathbb{R}^n$ where $n \ll |\mathcal{S}|$. We refer to the set of all features, for all state–action pairs, as Φ. In our experiments, the feature vector is constructed via tile coding and thus is binary, $\phi_t \in \{0, 1\}^n$, with a constant number of 1 features (see Sutton & Barto 1998). We also focus on the case where $|\mathcal{S}|$ is large, possibly infinite, but $|\mathcal{A}|$ is finite and relatively small, as is common in reinforcement learning problems. These are convenient special cases, but none of them is essential to our approach. Our approximate GVFs, denoted $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to \mathbb{R}$, are linear in the feature vector:

$$\hat{q}(s, a, \theta) = \theta^\top \phi(s, a),$$

where $\theta \in \mathbb{R}^n$ is the vector of weights to be learned, and $v^\top w = \sum_i v_i w_i$ denotes the inner product of two vectors v and w.

For learning the weights we use recently developed gradient-descent temporal-difference algorithms (Sutton et al. 2009, 2008; Maei et al. 2009, 2010). These algorithms are unique in their ability to learn stably and efficiently with function approximation from off-policy experience. Off-policy experience means experience generated by a policy, called the *behavior policy*, that is different from that being learned about, called the *target policy*. To learn knowledge efficiently from unsupervised interaction one seems inherently to face such a situation because one wants to learn in parallel about many policies—the different target policies π of each GVF—but of course one can only be behaving according to one policy at a time.

For a typical GVF, the actions taken by the behavior policy will match its target policy only on occasion, and rarely for more than a few steps in a row. For efficient learning, we need to be able to learn from these snippets of relevant experience, and this requires off-policy learning. The alternative—on-policy learning—would require learning only from snippets that are complete in that the actions match those of the GVF's target policy all the way to pseudo-termination, a much less common occurrence. If learning can be done off-policy from incomplete snippets of experience then it can be massively parallel and potentially much faster than on-policy learning.

Only in the last few years have off-policy learning algorithms become available that work reliably with function approximation and that scale appropriately for real-time learning and prediction (Sutton et al. 2008, 2009). Specifically, in this work we use the GQ(λ) algorithm (Maei & Sutton 2010). This algorithm maintains, for each GVF, a second set of weights $w \in \mathbb{R}^n$ in addition to θ and an eligibility-trace vector $e \in \mathbb{R}^n$. All three vectors are initialized to zero. Then, on each step, GQ(λ) computes two temporary quantities, $\bar{\phi}_t \in \mathbb{R}^n$ and $\delta_t \in \mathbb{R}$:

$$\bar{\phi}_t = \sum_a \pi(S_{t+1}, a)\phi(S_{t+1}, a),$$

$$\delta_t = r(S_{t+1}) + (1-\gamma(S_{t+1}))z(S_{t+1}) + \gamma(S_{t+1})\theta^\top\bar{\phi}_t - \theta^\top\phi(S_t, A_t),$$

and updates the three vectors:

$$\theta_{t+1} = \theta_t + \alpha_\theta\left(\delta_t e_t - \gamma(S_{t+1})(1-\lambda(S_{t+1}))(w_t^\top e_t)\bar{\phi}_t\right),$$

$$w_{t+1} = w_t + \alpha_w\left(\delta_t e_t - (w_t^\top\phi(S_t, A_t))\phi(S_t, A_t)\right),$$

$$e_t = \phi(S_t, A_t) + \gamma(S_t)\lambda(S_t)\frac{\pi(S_t, A_t)}{b(S_t, A_t)}e_{t-1},$$

where $b : \mathcal{S} \times \mathcal{A} \to [0,1]$ is the behavior policy and $\lambda : S \to [0,1]$ in an *eligibility-trace function* which determines the rate of decay of the eligibility traces as in the TD(λ) algorithm (Sutton 1988). Note that the per-time-step computation of this algorithm scales linearly with the number of features, n. Moreover, if the features are binary, then with a little care the per-time-step complexity can be kept a small multiple of the number of 1 features.

The approximation that will be found asymptotically by the GQ(λ) algorithm depends on the feature vectors Φ, the behavior policy b, and the eligibility-trace function λ. These three are collectively referred to as the *answer functions*. In this paper's experiments we always used constant λ, and all demons shared the same Φ and b. Finally, we note that Maei and Sutton defined a termination function, β, that is of the opposite sense as our γ; that is, $\beta(s) = 1 - \gamma(s)$. This is purely a notational difference and does not affect the algorithm in any way.

We can think of the demons as being of two kinds. A demon with a given target policy, π, is called a *prediction demon*, whereas a demon whose target policy is the greedy policy with respect to its own approximate GVF (i.e., $\pi = greedy(\hat{q})$, or $\pi(s, \arg\max_a \hat{q}(s, a, \theta)) = 1$) is called a *control demon.* Control demons can learn and represent how to achieve goals, whereas the knowledge in prediction demons is better thought of as declarative facts. One way in which the demons are not completely independent is that a prediction demon can reference the target policy of a control demon. For example, in this way one could ask questions such as 'If I follow this wall as long as I can, will my light sensor then have a high reading?'. Demons can also use each others' answers in their questions (as in temporal-difference networks). This allows one demon to learn a concept such as 'near an obstacle,' say as the probability of a high bump-sensor reading within a few seconds of random actions, and then a second demon to learn something based on this, such as 'If I follow this wall to its end, will I then be *near an obstacle*?' by using the first demon's approximate GVF in its terminal-reward function (e.g., $z(s) = \max_a \hat{q}(s, a, \theta_{\text{first\_demon}})$).

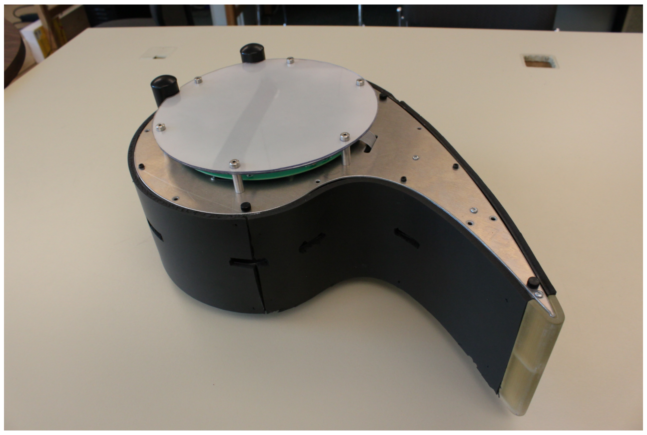


Figure 1. The Critterbot robotic platform.

5. RESULTS WITH HORDE ON THE CRITTERBOT

To evaluate the effectiveness of the Horde architecture, we deployed it on the *Critterbot*, a custom-built mobile robot (Figure 1). The Critterbot has a comma-shaped frame with a 'tail' that facilitates object interaction and is driven by three omni-directional wheels separated by 120 degrees. A diverse set of sensors are deployed on the top of the robot, including sensors for ambient light, heat, infrared light, magnetic fields, and sound. Another batch of sensors captures proprioceptive information including battery voltages, acceleration, rotational velocity, motor velocities, motor currents, motor temperatures, and motor voltages. The robot can detect nearby obstacles with ten infrared proximity sensors distributed along its sides and tail. The robot has been designed to withstand the rigors of reinforcement learning experiments; it can drive into walls for hours without damage or burning out its motors, it can dock autonomously with its charging station, and it can run continuously for twelve hours without recharging.

The Critterbot's sensors provide useful information about its interaction with the world, but this information can be challenging to model explicitly. For example, the sensor readings from the magnetometer may be influenced by the operation of data servers in the next room, and the ambient light sensors are affected by natural daylight, indoor florescent lights, shadows from looming humans, and reflections from walls. Manually modeling these interactions is difficult and potentially futile. The Horde architecture presents an alternative wherein each demon autonomously learns a little bit about the relationships among the sensors and actuators from unsupervised experience.

We performed a series of experiments to examine how well the architecture supports learning. In each experiment, the observations and actions were tiled to form a state–action feature representation Φ. A discrete set of actions were selected, matching the formulation of the GQ(λ) algorithm. With these choices, the entire architecture operates in constant time per step. We have run the Horde architecture in real-time with thousands of demons using billions of binary features of which a few thousand were active at a time, using laptop computers.

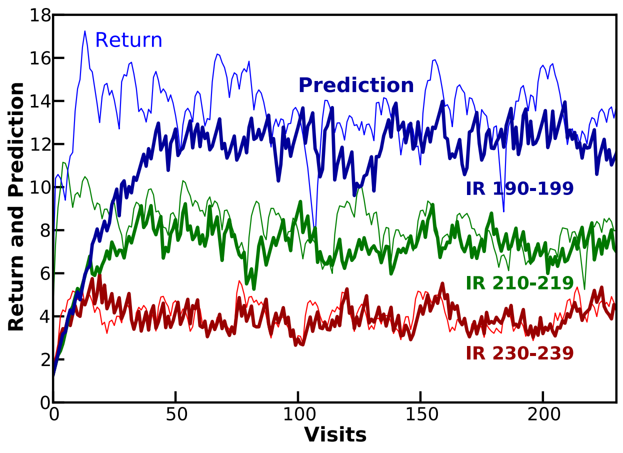


Figure 2. Accurately predicting time-to-obstacle. The robot was repeatedly driven toward a wall at a constant wheel speed. For each of three regions of the sensor space, for each time step spent in that region , we plot the demon prediction $\hat{q}$ on that step (bold line) and the actual return from that step (thin line).

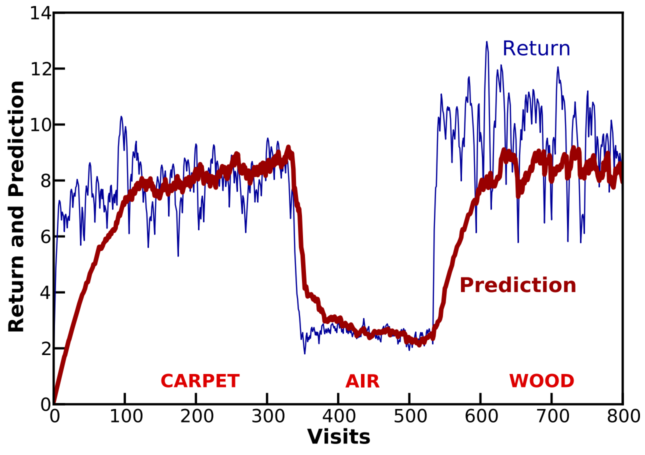


Figure 3. Accurately tracking time-to-stop. The robot was repeatedly rotated up to a standard wheel speed, then switched to a policy that always took the STOP action, on three different floor surfaces. Shown is the prediction $\hat{q}$ made on visits to a region of high velocity while stopping (bold line) together with the actual return from that visit (thin line). The floor surface was changed after visits 338 and 534.

5.1 Subjective prediction experiments

Our first two experiments dealt with Horde's ability to answer subjectively posed predictive questions. Figures 2 and 3 show results on the Critterbot with instances of the Horde architecture each with a single prediction demon. The specific questions posed are ones that might be useful in ensuring safety: 'How much time do I have before hitting an obstacle?' and 'How much time do I need to stop?'. In both cases accurate predictions were made, and in the latter case they were adapted so as to remain accurate as the experiment was changed from stopping on carpet, to stopping when suspended in the air, to stopping on a wood floor. The time step used in these experiment was approximately 30ms in length.

Figure 2 shows a comparison between predicted and observed time steps needed to reach obstacles when driving forward. Shown are the demon predictions $\hat{q}$ on each step (bold line) for each time step spent in a region of the sensor space (a visit), and the actual return from that step (thin line). The prediction was learned from a behaviour policy that cycled between three actions: driving forward, reverse, and resting. This is plotted for each of three regions of the sensor space: IR=190–199, IR=210–219, and IR=230–239. These represent three different value ranges of the Critterbot's front IR proximity sensor.

The question functions for this demon were: $\pi(s, \text{FORWARD}) = 1, r(s) = 1, z(s) = 0, \forall s \in \mathcal{S}$, and $\gamma(s) = 0$ if the value of the Critterbot's front-pointing IR proximity sensor was over a fixed threshold, else $\gamma(s) = 1$. The remaining answer functions were $\lambda(s) = 0.4, \forall s \in \mathcal{S}$, and Φ = a single tiling into twenty-six regions of the front IR sensor. The GQ(λ) step sizes were $\alpha_\theta = 0.3$ and $\alpha_w = 0.00001$. As shown in Figure 2, this demon learned to accurately predict the return (time steps to impact) for each range of its sensors.

Figure 3 demonstrates a demon's ability to accurately predict stopping times on different surfaces. Shown is the prediction $\hat{q}$ made on visits to a region of high velocity while stopping (bold line) together with the actual return from that visit (thin line). For this predictive question, we defined a single demon that predicts the number of timesteps until one of the robot's wheels approaches zero velocity (i.e., comes to a complete stop) under current environmental conditions. The robot's behaviour policy was to alternate at fixed intervals between spinning at full speed and resting. The floor surface, and thus the nature of the stopping problem, was changed after visits 338 and 534.

The question functions for this demon were: $\pi(s, \text{STOP}) = 1, r(s) = 1, z(s) = 0, \forall s \in \mathcal{S}$, and $\gamma(s) = 0$ if the wheel's velocity sensor was below a fixed threshold, else $\gamma(s) = 1$. The remaining answer functions were $\lambda(s) = 0.1, \forall s \in \mathcal{S}$, and Φ = a single tiling into eight regions of the wheel's velocity sensor. The GQ(λ) step sizes were $\alpha_\theta = 0.1$ and $\alpha_w = 0.001$. As illustrated in Figure 3, this demon learned to correctly predict the return (time steps to stopping) on carpet, then adapted its prediction when the environment changed to air and then to wood flooring.

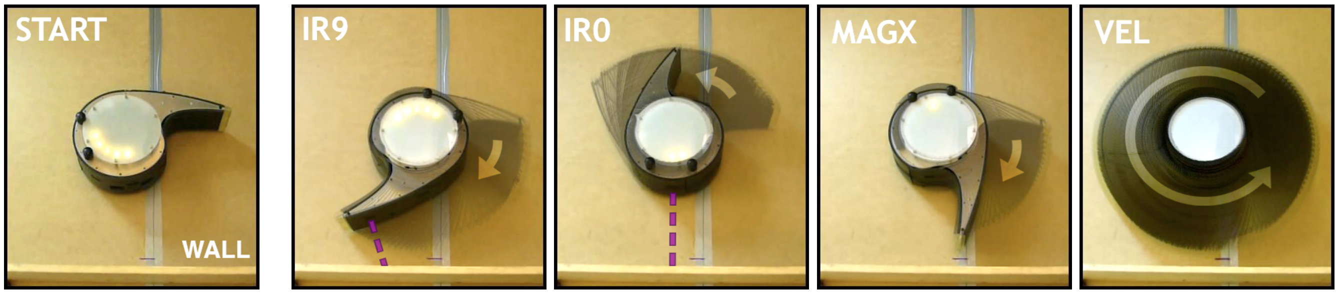


Figure 4. Illustration of policies learned by four control demons in the spinning experiment. The first panel shows the standard starting position, and the other four panels show the motions from that position produced when control was given to one of the eight learned demon policies each tasked to maximize a different sensor. By maximized sensor: IR9) Robot quickly rotates clockwise and stops in the position that maximizes the IR proximity sensor on the side of the robot's tail; IRO) Robot quickly rotates counterclockwise, overshoots a bit, then settles in a position that maximizes the proximity sensor between the robot's 'eyes'; MAGX) Robot rotates clockwise and stops at a position that maximizes the magnetic x-axis sensor; VEL) Robot spins continuously, maximizing the wheel velocity sensor.

5.2 Off-policy learning of multiple spinning control policies

Our third experiment examined whether control demons can learn policies in parallel while following a random behavior policy, in other words, whether the demons can learn off-policy, a crucial ability for the scalability of the architecture. The action set in this experiment was {ROTATE-RIGHT, ROTATE-LEFT, STOP}. The behavior policy was to randomly select one of the three actions, with a bias (50% probability) toward repeating the action taken on the previous time step. The result of this behavior policy was that the robot would spin in place in both directions with a variety of speeds and durations over time. The state space was represented with four overlapping joint tilings across three sensors: the magnetometer, one of the IR sensors, and the velocity of one of the wheels. Each sensor was divided into eight regions for the tilings, resulting in a total of $3 \times 4 \times 8^3 = 6144$ binary features. One additional feature was provided as a bias unit (always =1), and three additional binary features were used to encode the previous action. The time step corresponded to approximately 100ms. The other parameters were $\alpha_\theta = 0.1$, $\alpha_w = 0.001$, and $\lambda(s) = 0.4, \forall s \in \mathcal{S}$. Learning was done online, but the data was also saved so that the whole learning process could be repeated without using the robot if desired (this is one of the advantages of an off-policy learning ability).

In this experiment we ran eight control demons in parallel for 100,000 time steps of off-policy learning with actions selected according to the behavior policy. Each demon was tasked with learning how to maximize a different sensor value. That is, their question functions were $\pi = greedy(\hat{q})$ and, for all $s \in \mathcal{S}$, $\gamma(s) = 0.98$, $z(s) = 0$, and $r(s) =$ the value of one of eight sensors approximately normalized to a 0 to 1 range. The eight sensors used as rewards were four of the IR proximity sensors, the magnetometer, the velocity sensor for one of the wheels, one of the thermal sensors, and an IR beacon sensor for the charging station. To objectively measure the quality of the policies learned by the eight demons, we occasionally interrupted learning to evaluate them on-policy. That is, with learning turned off, the robot followed one of the eight learned demon policies for 250 time steps and we measured the demon's return. We repeated this for each demon ten times from each of three initial starting positions (angles) to produce 30 measures of the effectiveness of each demon's policy at that point in the training. These numbers were averaged together to produce the learning curves shown in Figure 5.

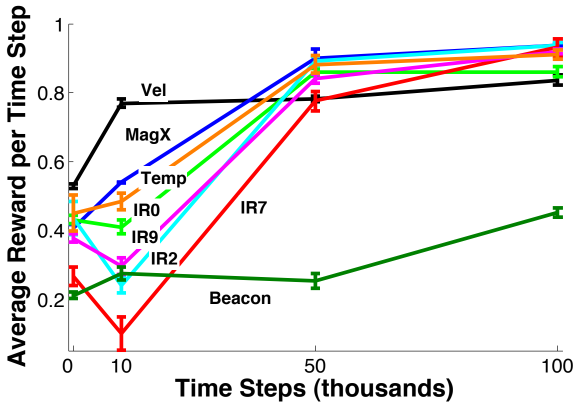
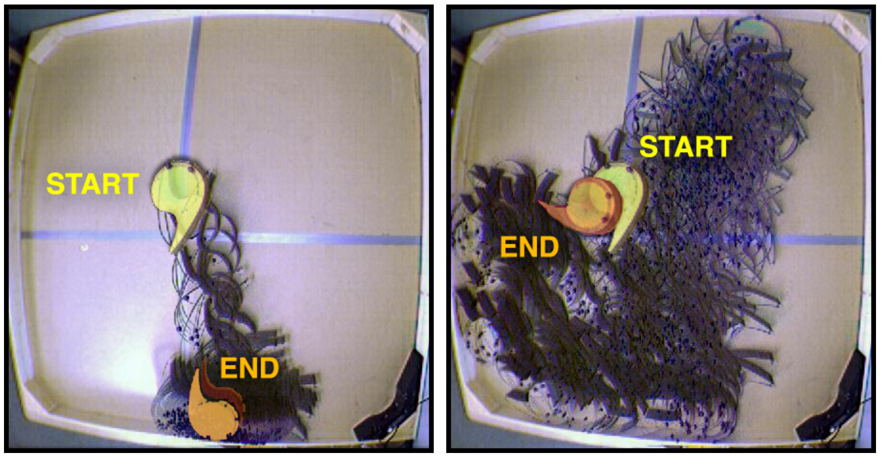


Figure 5. Learning curves for eight control demons learning off-policy in the spinning experiment. From extensive experience spinning, eight control demons learned different policies each maximizing a different sensor. The graph shows the performance of the policies, gathered in special on-policy evaluation sessions during which learning was turned off. All demons learned to perform near optimally. Rewards were scaled to the range [0, 1], but because the beacon light flashes on and off, its maximal average was 0.5.

Examples of the final learned behavior from four of the demons are shown in Figure 4. These photos show typical behavior, which in the case of all eight demons appeared to successfully maximize the targeted sensor. In separate runs we found that it would take approximately 25,000 steps each to learn similarly competent control policies for a single demon while behaving according to its policy as it was learned (on-policy training). In only four times longer, we learned eight demons in parallel, and could potentially have learned thousands or millions more using off-policy learning.



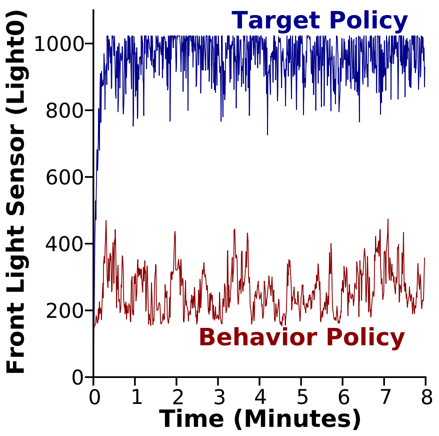


Figure 6. Learning light-seeking behavior from random behavior. Shown are superimposed images of robot positions: Left) In testing, the robot under control of the demon policy turns and drives straight to the light source at the bottom of the image; Middle) Under control of the random behavior policy for the same amount of time, the robot instead wanders all over the pen; Right) Light sensor readings averaged over seven such pairs of runs, showing much higher values for the learned target policy.

5.3 Off-policy learning of light-seeking

A final experiment examined whether a control demon could learn a goal-directed policy when given a much greater breadth of experience. In particular, we chose question functions corresponding to the goal of maximizing the near-term value of one of the light sensors: $\pi = greedy(\hat{q})$, $\gamma(s) = 0.9$, $z(s) = 0$, $r(s) =$ a scaled reading from the front light sensor. The behavior policy was to pick randomly from the set $\{+10, -10, 0\}^3$ interpreted as velocities for the robot's three wheels, for a total of 27 possible actions. The state space was represented with 32 individual tilings over each of the four directional light sensors, where each tile covered about 1/8th of the range. With the addition of a bias unit, this made for a total of $27 \times (32 \times 4 \times 8 + 1) = 27,675$ binary features, of which $32 \times 4 + 1 = 129$ were active on each time step. The time step corresponded to approximately 500ms.

Using the random behavior policy, we collected a training set of 61,200 time steps (approximately 8.5 hours) with a bright light at nearly floor level on one side of the pen. During this time the robot wandered all over the pen in many orientations. We trained the control demon off-line and off-policy in two passes over the training set. To assess what had been learned, we then placed the robot in the middle of the pen facing away from the light and gave control to the demon's learned policy. The robot would typically turn immediately and drive toward the light, as shown in the first panel of Figure 6. This result demonstrates that demons can learn effective goal-directed behavior from substantially different training behavior.

Together, our results show that the Horde architecture can be applied to robot systems to learn potentially useful bits of knowledge in real-time from unsupervised experience. The approach works across a range of feature representations, parameters, questions, and goals. The robot is able to learn bits of knowledge that could serve as useful components for solving more complex tasks.

6. CONCLUSION

The Horde architecture is an experiment in knowledge representation and learning built upon ideas and algorithms from reinforcement learning. The approach is to express knowledge in the form of generalized value functions (GVFs) and thereby ground its semantics in sensorimotor data. This approach is promising because 1) value functions make it possible to capture temporally extended predictive and goal-oriented knowledge, 2) a large amount of important knowledge is of this form, 3) conventional knowledge representations of the grounded type (such as differential equations) have difficulty representing knowledge of this form, and 4) conventional methods that can capture this kind of knowledge (high-level, symbolic methods such as rules, operators, and production systems) are not as grounded and therefore not as learnable as value functions. Although value functions have always been potentially learnable, only recently have scalable learning methods become available that make it practical to explore the idea of GVFs with off-policy learning and function approximation. This work presents a first look at the application and interpretation of GVFs in an architecture with parallel off-policy learners.

In this paper we have focused on representing and learning knowledge as GVFs, and as such we have made only suggestive comments about how such knowledge could be used. Although this is an important limitation of our work, we believe that it is an appropriate way to break down the problem. The issues in learning and representation with GVFs that we address here are non-trivial and have not been adequately addressed before—certainly not in an embodied, robotic form. In addition, reinforcement-learning ideas such as value functions are already closely connected to known action-selection and planning methods; it is not a great leap to imagine several ways in which GVFs could be used to generate and improve behavior. We have briefly demonstrated some of these, such as passing control to the learned policy of single demons (e.g., the sensor-maximization demons in Section 5.2 and the light-seeking demon in Section 5.3), and indicated how several demons could be combined to modulate an existing policy (e.g., varying behavior based on impact and stopping time predictions as suggested by Section 5.1). A rich and varied collection of demons and questions, as made possible by the Horde architecture, allows for a broad set of fusions of this kind. We have not developed here the natural possibility of using GVFs to represent multi-scale policy-contingent models of the world's dynamics (option models; Sutton, Precup & Singh 1999), and then using the models for planning as in dynamic programming, Monte Carlo tree search (see Chaslot 2010), or Dyna architectures (Sutton 1990). This is another natural direction for future work.

7. ACKNOWLEDGMENTS

The authors are grateful to Anna Koop, Mark Ring, Hamid Maei, and Chris Rayner for insights into the ideas presented in this paper. We also thank Michael Sokolsky and Marc Bellemare for assistance with the design, creation, and maintenance of the Critterbot. This research was supported by iCORE and Alberta Ingenuity, both part of Alberta Innovates – Technology Futures, by the Natural Sciences and Engineering Research Council of Canada, and by MITACS.

8. REFERENCES

Becker, J. D. (1973). A model for the encoding of experiential information. In *Computer Models of Thought and Language*, Schank, R. C., Colby, K. M., Eds. W. H. Freeman and Company.

Chaslot, G. M. J-B. (2010). Monte-Carlo tree search. PhD thesis, Dutch Research School for Information and Knowledge Systems.

Cohen, P. R., Atkin, M. S., Oates, T., Beal, C. R. (1997). Neo: Learning conceptual knowledge by sensorimotor interaction with an environment. In *Agents '97*, Marina del Rey, CA. ACM.

Cunningham, M. (1972). *Intelligence: Its Organization and Development*. Academic Press.

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

Kaelbling, L. P., Oates, T., Hernandez, N., Finney, S. (2001). Learning in worlds with objects. *Working Notes of the AAAI Stanford Spring Symposium on Learning Grounded Representations*.

Maei, H. R., Sutton, R. S. (2010). GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces. In *Proceedings of the Third Conference on Artificial General Intelligence*, Lugano, Switzerland.

Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Precup, D., Silver, D., Sutton, R. S. (2009). Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems 22*, Vancouver, BC. MIT Press.

Maei, H. R., Szepesvári, Cs., Bhatnagar, S., Sutton, R. S. (2010). Toward off-policy learning control with function approximation. In *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel.

Natale, L. (2005). Linking action to perception in a humanoid robot: A developmental approach to grasping. MIT PhD thesis.

Oates, T., Schmill, M. D., Cohen, P. R. (2000). A method for clustering the experiences of a mobile robot that accords with human judgments. *Proceedings AAAI*, 846–851, AAAI/MIT Press.

Pasula, H., Zettlemoyer, L., Kaelbling L. (2007). Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research 29*:309–352.

Pierce, D. M., Kuipers, B. J. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence 92*:169–227.

Ring, M. B. (1997). CHILD: A first step toward continual learning. *Machine Learning 28*:77–104.

Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning 3*:9–44.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224. Morgan Kaufmann, San Mateo, CA.

Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvari, Cs., Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada.

Sutton, R. S., Precup D., Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence 112*:181–211.

Sutton, R. S., Rafols, E. J., Koop, A. (2006). Temporal abstraction in temporal-difference networks. *Advances in Neural Information Processing Systems 18*.

Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2008). A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximation. *Advances in Neural Information Processing Systems 21*.

Yu, C., Ballard, D. (2004). A multimodal learning interface for grounding spoken language in sensory perceptions. *ACM Transactions on Applied Perception 1*:57–80.