# Prediction in Intelligence: An Empirical Comparison of Off-policy Algorithms on Robots

Banafsheh Rafiee, Sina Ghiassian, Adam White, Richard S. Sutton
Department of Computing Science, University of Alberta, Edmonton, Canada
{rafiee,ghiassia,amw8,rsutton}@ualberta.ca

## ABSTRACT

The ability to continually make predictions about the world may be central to intelligence. Off-policy learning and general value functions (GVFs) are well-established algorithmic techniques for learning about many signals while interacting with the world. In the past couple of years, many ambitious works have used off-policy GVF learning to improve control performance in both simulation and robotic control tasks. Many of these works use semi-gradient temporal-difference (TD) learning algorithms, like Q-learning, which are potentially divergent. In the last decade, several TD learning algorithms have been proposed that are convergent and computationally efficient, but not much is known about how they perform in practice, especially on robots. In this work, we perform an empirical comparison of modern off-policy GVF learning algorithms on three different robot platforms, providing insights into their strengths and weaknesses. We also discuss the challenges of conducting fair comparative studies of off-policy learning on robots and develop a new evaluation methodology that is successful and applicable to a relatively complicated robot domain.

## KEYWORDS

artificial intelligence; robotics; reinforcement learning; off-policy learning; temporal-difference learning; general value functions

**ACM Reference Format:**
Banafsheh Rafiee, Sina Ghiassian, Adam White, Richard S. Sutton. 2019. Prediction in Intelligence: An Empirical Comparison of Off-policy Algorithms on Robots. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

The ability to make off-policy predictions may be central to intelligence. For example, Littman, Sutton, and Singh (2002) showed that the state of a partially observable dynamical system can be represented in terms of predictions about the low-level data acquired from interaction with the world. It has also been argued that a general sense of knowledge can be represented using networks of interrelated predictions while being grounded in sensorimotor data (Tanner and Sutton, 2005; Rafols et al., 2006; Sutton, 2009; Sutton and Tanner, 2005). Sutton et al. (2011) proposed an architecture for knowledge representation consisting of many sub-agents each trying to answer a predictive question about the environment. These predictive questions are represented with general value functions (GVFs); a value function with a generalized notion of target and termination that can be learned online with conventional temporal difference learning methods. Finally, Ring (in preparation) demonstrated how a collection of predictions, encoded as GVFs, can be learned and combined layer-by-layer from low-level statements about the data-stream all the way up to high-level concepts.

Several recent large-scale learning systems have utilized large collections of GVFs learned off-policy from a single stream of data. The UNREAL learning system (Jaderberg et al., 2016) used hundreds of auxiliary prediction and control tasks to improve a shared state representation, resulting in state-of-the-art control performance across numerous baselines. The UVFA architecture (Schaul et al, 2015) combines dozens of GVFs to generalize learned policies to novel goals never experienced by the system during training. The Predictron (Silver et al., 2017) uses a network of GVFs to construct an implicit model of the world. The HYDRA learning system (Van Seijen et al., 2017) shows how a complex task can be decomposed into a collection of GVFs to dramatically speed learning in large-scale applications like Atari.

The benefits of off-policy GVF learning could be more pronounced in physical systems like robots, where the agent can learn about multiple ways of behaving from a single stream of experience. GVFs have been used to generate better exploratory behavior in high-degree-of-freedom robotic control tasks. The Scheduled Auxiliary Task (SAC) architecture (Riedmiller et al., 2018) learns a collection of GVFs corresponding to primitive behaviors like reaching to touch an object. The system learns which primitive-behavior GVF to follow for dramatically speeding up exploration and allowing the system to learn complex behavior like block stacking from scratch, with no prior knowledge. In simulations an agent can learn from multiple instances of the environment, substantially speeding up learning (Jaderberg et al., 2016). This approach can in principle be employed in settings with multiple robots (Gu et al., 2017); however, in practice asynchronous updating only works well if the environments (including the robot) are nearly identical. Ultimately we want our AI systems to run on autonomous mobile robot platforms, learning online in a changing world from a lifetime of experience—in these settings off-policy GVF learning may be critical for scaling up learning.

Many of the demonstrations of GVF learning above use potentially divergent learning algorithms. The UNREAL, UVFA, and Hydra architectures use Q-learning, which can diverge with function approximation (Baird,1995; Sutton and Barto, 2018), while SAC uses the Retrace algorithm (Munos et al., 2016), which has been recently shown to diverge with approximation (Touati et al., 2018). These algorithms remain popular in application because they are well understood, computationally efficient (linear in the number of weights), work well with function approximation, and divergence cases are rare in practice (Van Hasselt et al., 2018).

*Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada.* 

Recently several families of methods have been introduced that are also computationally efficient, based on temporal-difference updates, and guaranteed convergent with approximation (e.g., Sutton et al., 2009; Sutton, Mahmood, and White, 2016). However, much less is known about these new methods in practice. A handful of simulation studies (Dann et al., 2014; Geist and Scherrer, 2014; White and White, 2016; Ghiassian et al., 2018) have suggested that these new methods achieve good performance in small problems, but we have little data on how these methods perform in larger control domains. Prior works have demonstrated that Gradient-TD methods can successfully learn accurate predictions and effective policies on robots (Sutton et al., 2011; White, 2015). However, these work do not include algorithmic comparisons, parameter sensitivity analysis, or averaging over multiple runs. To our knowledge, there has been no work comparing off-policy GVF learning algorithms on real physical systems.

Comparing off-policy learning algorithms on robots is challenging for several reasons. Off-policy learning allows the system to learn about many—potentially thousands—GVFs in parallel. The GVFs will likely learn at different rates. Easy predictive questions may be learned quickly and others will require significantly more data. Therefore, tracking the accuracy of only a subset of the GVFs may not well reflect the accuracy of the entire collection. Each GVF can be contingent on a different policy—thousands of GVFs may make use of thousands of policies—we must somehow access the accuracy of GVFs about policies that may never be executed during normal operation. On a robot, every time step requires time in the real world (and human supervision), and thus empirical comparisons must judiciously allocate computation and samples in the most effective way possible. Currently, there are no well established best practices on how to conduct meaningful and fair comparisons of off-policy learning algorithms on robots.

This paper has three primary contributions. First, we present three case studies in which we empirically compare several modern off-policy GVF learning algorithms on robot platforms. Second, we propose a new methodology for evaluating off-policy algorithms on robots. Third, we provide several new insights into the performance of the algorithms. In particular our results suggest: (i) Emphatic-TD methods can substantially outperform other methods in prediction tasks (fixed policy GVFs) but can be more sensitive to their choice of learning rate parameter. (ii) In prediction tasks, Emphatic-TD methods work better when the behavior is closer to on-policy, compared with Gradient-TD methods that do better when the behavior is more random. (iii) Eligibility traces significantly improve performance over one-step methods in prediction tasks. (iv) In GVF control learning, Greedy-GQ exhibited substantially less variance across test runs, compared with conventional Q-learning. (v) ABQ, an off-policy method that does not use importance sampling, did not show a significant advantage over importance sampling-based Gradient-TD methods. Our experiments are the most extensive and thorough comparisons of off-policy GVF learning performed on robots to date, including many algorithms and a wide range of parameters.

## 2 BACKGROUND

In this paper, we consider the problem of estimating the value function for a Markov decision process (MDP) where there is an interaction loop between an agent and its environment. The agent and environment interact continually at discrete time steps. At each time step $t$, the agent is in a state $S_t \in \mathcal{S}$ and takes an action $A_t \in \mathcal{A}$ according to its behavior policy $b : |\mathcal{S}| \times |\mathcal{A}| \to [0, 1]$. The environment, in turn, emits a reward $R_{t+1} \in \mathcal{R}$ and takes the agent to the next state $S_{t+1} \in \mathcal{S}$. The objective is to estimate the expected return for a target policy $\pi : |\mathcal{S}| \times |\mathcal{A}| \to [0, 1]$, that is sum of the discounted rewards given that the agent takes its actions according to $\pi$. The expectation of return is known as the value function and is denoted by $v_\pi$:

$$v_\pi(s) = E\left[ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_{t:\infty} \sim \pi \right]$$

where $\gamma$ is the discount factor.

In this work, we consider a more general notion of value function, introduced by Sutton et al. (2011), that is not limited to reward and is called a general value function (GVF). A GVF, similar to a value function, can be written as the expectation of a weighted sum of a signal of interest:

$$v_{\pi,\gamma,c}(s) = E\left[ \sum_{k=0}^{\infty} \left( \prod_{j=1}^{k} \gamma(S_{t+j}) \right) c(S_{t+k+1}) \;\middle|\; S_t = s, A_{t:\infty} \sim \pi \right]$$

where $\gamma : \mathcal{S} \to [0, 1]$ is a generalization of the discount factor to a horizon (termination) function. The horizon function specifies the probability of the GVF terminating at each state. Therefore, it specifies the time scale of the prediction: $T = \frac{\Delta t}{1-\gamma}$, where $\Delta t$ is the agent-environment update cycle. $c : \mathcal{S} \to \mathcal{R}$ is the signal of interest and is called the cumulant of the prediction.

For estimating value functions, we use temporal-difference (TD) learning methods. The most classic TD method, TD($\lambda$), in the case of linear function approximation can be summarized as follows:

$$\begin{aligned} \delta_t &\doteq R_{t+1} + \gamma_{t+1} \mathbf{w}_t^T \mathbf{x}(S_{t+1}) - \mathbf{w}_t^T \mathbf{x}(S_t) \\ \mathbf{z}_t &\leftarrow \gamma_t \lambda \mathbf{z}_{t-1} + \mathbf{x}_t \\ \mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t \end{aligned} \tag{1}$$

where the learned value function for state $S_t$ is denoted by $\mathbf{w}_t^T \mathbf{x}(S_t)$ and is the dot product of the weight vector $\mathbf{w}$ and the feature vector $\mathbf{x}(S_t)$. $\delta_t$ is the TD error and can be thought of as the difference between the learned value of $S_t$ and a more accurate estimate of it, $R_{t+1} + \gamma_{t+1} \mathbf{w}_t^T \mathbf{x}(S_{t+1})$. $\mathbf{z}$ is the eligibility trace vector. The eligibility trace at each time step determines the eligibility of each component of the weight vector to be affected by the TD error of that time step. The eligibility of the components of the weight vector decays over time and increases whenever they participate in forming an estimate of the value of a visited state. The rate of decaying is determined by the trace parameter $\lambda$ which can also be thought of as a bias-variance knob.

We consider off-policy learning that is to learn about a policy using the data generated from a different policy. In order to learn about a target policy, $\pi$, based on the data generated by a different behavior policy, $b$, we have to correct for the differences between

the two policies. This is usually done using the importance sampling ratio $\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$. Off-policy TD($\lambda$) uses the importance sampling ratio in the update of the eligibility trace to account for the differences between the two policies. Unfortunately, off-policy TD($\lambda$) is proven divergent. However, there are several families of TD methods that incorporate importance sampling ratio in different ways to achieve convergence (e.g., Emphatic-TD methods and Gradient-TD methods).

To this point, we considered the problem of estimating the value function for a fixed policy, also known as the policy evaluation or prediction problem. Another category of problems focuses on finding the optimal policy—a policy that maximizes the sum of discounted rewards—also known as the control problem. Solving the control problem involves a policy improvement step in addition to estimating the value function of the learned policy. A well-known off-policy TD algorithm for solving control problems is Q-learning (Watkins,1989). Despite being successful in many domains, including Atari games (Mnih et al., 2016), Q-learning does not have convergence guarantees when combined with function approximation. Although classic off-policy TD($\lambda$) and Q-learning do not have convergence guarantees, various TD methods have recently been proposed that are convergent under off-policy training. In the next section, we discuss several of these methods.

## 3 OFF-POLICY ALGORITHMS

We consider both prediction and control algorithms from the TD family of methods. We focus on algorithms that are computationally linear in the number of function approximation parameters. We do not consider methods like Least-squares TD methods (Bradtke and Barto, 1996; Boyan, 1999) because they are computationally expensive and are not applicable to large-scale learning systems.

For off-policy prediction, we consider several TD methods in addition to the classic TD($\lambda$). Gradient-TD methods were the first to provide convergence guarantees with two learned weight vectors. These methods use stochastic gradient descent to minimize the Mean Squared Projected Bellman Error (MSPBE) objective. We consider GTD($\lambda$) and GTD2($\lambda$) (Sutton et al., 2009; Maei, 2011) from the Gradient-TD family. It was known early on that TD($\lambda$) can be superior to GTD($\lambda$) in the on-policy case. This motivated the creation of another algorithm that we consider, called HTD($\lambda$) (Hackman, 2012; White and White, 2016). HTD($\lambda$) performs TD($\lambda$)-like updates when data is sampled on-policy and GTD($\lambda$)-like updates when data is sampled off-policy, while providing the same convergence guarantees as GTD($\lambda$). Different derivations are possible if we consider saddle point formulation of the objective function. Examples of such methods are proximal-GTD and proximal-GTD2 algorithms which we consider in our study (Mahadevan et al., 2014; Liu et al., 2015).

Emphatic-TD methods were the first family to provide convergence guarantees under off-policy training with function approximation using only one learned weight vector. We study the original ETD($\lambda$) (Sutton et al., 2016) as well as a slightly modified version of it, ETD($\lambda$, $\beta$) (Hallak et al., 2016). ETD($\lambda$, $\beta$) has an extra parameter, $\beta$, that acts as a bias-variance parameter. Off-policy TD and ETD are special cases of ETD($\lambda$, $\beta$) for $\beta = 0$ and $\beta = \gamma$ respectively.

Off-policy learning can suffer from high variance introduced by the importance sampling ratio. Methods have been proposed that tackle this problem by omitting an explicit use of an importance sampling ratio. These methods use a variable $\lambda$ to reduce the effect of importance sampling ratio on the update when it is large. Examples of these algorithms are V-trace($\lambda$), Tree-Backup($\lambda$), and ABQ($\zeta$) (Espeholt et al., 2018; Precup et al., 2000; Mahmood et al., 2017). In this work, we consider ABQ($\zeta$) from this group since it was shown to slightly outperforming the other two methods (Ghiassian et al., 2018). The original ABQ($\zeta$) method was proposed for control, we consider a variant of it that is for policy evaluation proposed by Ghiassian et al. (2018), called ABTD($\zeta$). The original ABQ($\zeta$) method also uses gradient corrections and is proven stable under off-policy training with function approximation. We, however, study a simpler version of ABQ($\zeta$) that does not use gradient corrections.

For off-policy control, we consider two algorithms in addition to the classic Q-learning method. First one is Greedy-GQ($\lambda$) from the Gradient-TD family of methods. This algorithm extends the original GTD($\lambda$) method to the control case (Maei et al., 2010). The second method is Greedy-ABQ($\zeta$) for control. We derived this method using the same greedification process that was used to derive Greedy-GQ($\lambda$) from GTD($\lambda$).

## 4 THE DYNAMIXEL CASE STUDY

We begin our comparative study of off-policy learning with a simple robot prediction task since the interpretation of the results in this case is more straightforward. We constructed the robot using two Dynamixel AX-12 motors and will refer to it as the Dynamixel robot. We used this robot to design and solve a prediction task; to learn how soon one of the motors reaches a particular target angle, given that it is going back and forth between two limiting angles. The two limiting angles are at horizontal and vertical positions (see Figure 1) and have the values of 0 and 1.5 radians respectively. At each time step, the motor can move left or right for 0.05 radians. We formulated this prediction task as a single GVF. The target policy was to move back and forth between the two limiting angles. The horizon function returned 0.9 unless the distance between the current angle and the target angle was less than 0.05 (termination condition) in which case it returned 0. The cumulant function returned 1 when the distance between the current angle and the target angle was less than 0.05; otherwise, it returned 0.

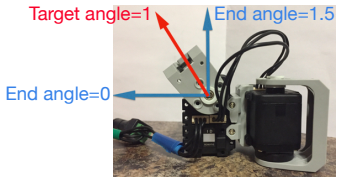


**Figure 1: The Dynamixel robot**

To approximate this GVF, we used a linear function with a tile coded representation. To construct the feature vector, we used the angle and velocity of the motor. Angle is between −0.1 and 1.6. Velocity is 1 whenever the difference between the current angle and the previous angle is positive and has a value of −1 otherwise.

To produce features, the angle was tile coded using 8 tilings each with 4 tiles resulting in a binary vector of size $8 \times 4$. The final feature vector was of size $2 \times 8 \times 4$ where each of the 2 parts corresponded to one of the values of velocity.

We applied the prediction algorithms discussed in Section 3 to this prediction task. We made many instances of them by considering a wide range of parameters. Step-size was in the form $\frac{\alpha}{\text{number of tilings}}$ where $\alpha \in \{0.1 \times 2^x | x = -10, -9, ..., 3\}$. For Gradient-TD methods the second step-size was in the following form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{2^x | x = -10, -12, ..., 2\}$. The trace parameters, $\zeta$ and $\lambda$, were a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1\}$. $\beta$ for ETD$(\lambda, \beta)$ was a number in $\{0.2, 0.4, 0.6, 0.8, 1\}$.

The behavior policy was to move in the same direction as target policy with probability 0.9 and to move in the opposite direction with probability 0.1. We generated 30 runs of data. Each run consisted of 20,000 steps and took approximately 100 minutes. We then used the data to learn the value function offline; meaning that gathering the data and learning happened in two separate phases.

Evaluating the learning algorithms on a robot can be challenging. In simulated domains, the difference between the estimated and true value function is used as the error measure. In this domain, like all other robot domains, we do not have access to the true value function. Therefore, we cannot evaluate the algorithms by looking at the difference between the estimated and true values. To evaluate the algorithms, we sampled several time steps following the behavior policy and calculated the return corresponding to those time steps by following the target policy prior to learning. We recorded the observations corresponding to the sampled time steps along with the corresponding returns to form the evaluation data. During learning time, we computed the squared difference between the estimated predictions and the returns and computed the average over the evaluation data. We denote our performance measure by $\widehat{\text{MSRE}}(\mathbf{w})$ because it is an estimation of the mean squared return error (MSRE). MSRE can be defined as the difference between the estimated value and the return, squared and averaged over all states:

$$\text{MSRE}(\mathbf{w}) = \sum_{s \in S} d_b(s) E[(\mathbf{w}^T \mathbf{x}(S_t) - G_t)^2 | S_t = s]$$

where $d_b(s)$ denotes the probability of state $s$ under the behavior policy. $w$ and $x(S_t)$ are respectively the weight vector and the feature vector and their dot product produces the estimation of the value of $S_t$. $G_t$ is the return at time step $t$.

To demonstrate the performance of the algorithms, we used learning curves and parameter studies. Learning curves show the mean squared return error at each time step. The parameter studies show the asymptotic performance for different values of step-size. To estimate the asymptotic performance, we computed the average of error over the last 0.25 percent of each run and averaged over 30 runs. The learning curves and parameter studies of the asymptotic performance over step-size for all the algorithms for $\lambda = 0$ are shown in Figure 2. We report the results for $\lambda = 0$ because in this case, the difference between the algorithms was more evident. All the other parameters (e.g., GTD's second step-size) were set to values resulting in the lowest asymptotic performance.

According to the learning curves, ETD(0) and ETD$(0, \beta)$ substantially outperformed other algorithms in terms of asymptotic

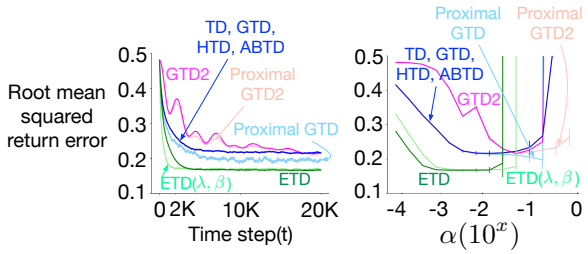


**Figure 2: The learning curves and parameter studies of the asymptotic performance over step-size for the case of $\lambda = 0$ for the Dynamixel case study. ETD(0) and ETD$(0, \beta)$ reached a lower level of asymptotic error and were faster.**

performance and speed with ETD$(0, \beta)$ being the fastest. Proximal-GTD(0) achieved the next best level of asymptotic error. However, it achieved its best performance for one specific parameter setting whereas ETD(0) and ETD$(0, \beta)$ achieved theirs for a wide range of step-sizes according to the parameter studies.

The effect of step-size on the asymptotic performance of TD$(\lambda)$ and ETD$(\lambda)$ for different values of $\lambda$ is shown in Figure 3. ETD$(\lambda)$, unlike TD$(\lambda)$, was not sensitive to the trace parameter, $\lambda$, and achieved the same level of asymptotic error for all values of it. TD$(\lambda)$, however, worked better with eligibility traces. For higher values of $\lambda$, TD$(\lambda)$ achieved a lower level of error but converged for smaller values of step-size.

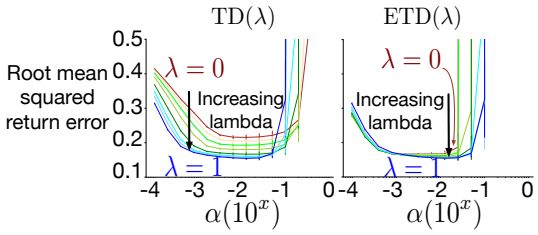


**Figure 3: The parameter studies of the asymptotic performance over step-size for TD$(\lambda)$ and ETD$(\lambda)$ for the Dynamixel case study.**

The effect of step-size on the asymptotic performance of GTD$(\lambda)$, HTD$(\lambda)$, and GTD2$(\lambda)$ for different values of $\lambda$ and second step-size is shown in Figure 4. Each line corresponds to a specific value of $\lambda$ and second step-size with the curves of the same color corresponding to the same value of $\lambda$. All methods worked better with eligibility traces. GTD$(\lambda)$ and HTD$(\lambda)$ were quite robust to the second step-size and the sensitivity of GTD$(\lambda)$ reduced as $\lambda$ got bigger. GTD2$(\lambda)$, however, was more sensitive to the second step-size.

## 5 THE COLLISION CASE STUDY

In the previous case study, we considered the problem of predicting how soon an event happens—how soon the Dynamixel motor reaches a target angle. In this case study, we consider a similar prediction task in a more complicated robot domain with a higher dimension input space and a more exploratory behavior policy. For

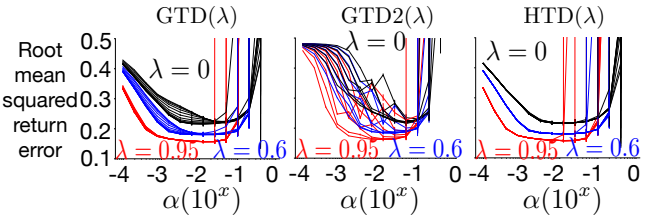


**Figure 4: The parameter studies of the asymptotic performance over step-size for Gradient-TD algorithms for the Dynamixel case study. Each curve corresponds to a specific value of $\lambda$ and second step-size.**

this case study, we used a Kobuki robot. The robot wanders in a pen and tries to learn how soon it will bump into something if it goes forward (Figure 5). The sensors available to the agent are Kobuki's camera and two bump sensors. We formulated this task as a single GVF. The target policy was to pick the forward action in all states. The horizon function returned zero whenever the robot bumped into something and returned 0.97 otherwise (Given $\gamma = 0.97$ and Kobuki's update cycle of 0.1 second, the time scale of the GVF is about 3 seconds). The cumulant function returned a binary value that became 1 whenever either of the bump sensors was on.

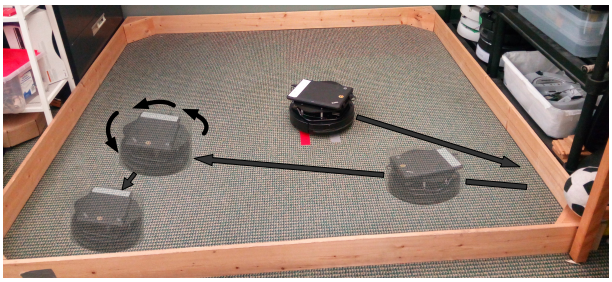

**Figure 5: The Kobuki wandering in the pen.**

To approximate this GVF we used a linear function with tile coded features. To construct the feature vector we used the tile coding software publicly available on Richard Sutton's website[1]. The input to tile coding was 50 RGB pixels randomly selected from the camera, represented as a vector of size 150. We tile coded each vector element separately using 8 tilings each with 4 tiles. (The software uses a hash table. The feature vector size is the same as the size of the hash table—9600 in our case.)

We applied the aforementioned prediction algorithms to this prediction task. We made many instances of each method by considering a wide range of parameters. The parameter settings that we tried for the Collision case study were the same as the Dynamixel case study except that $\alpha$ was a number in $\{0.1 \times 2^x | x = -10, -9, ..., 5\}$.

Behavior policy was to move forward with probability 0.9 and turn left otherwise. Although in both Dynamixel and Collision case studies, the behavior policy differed from the target policy 10 percent of the time, in the latter the behavior is more exploratory. This is because time-wise it takes longer to drive across the pen than to rotate on the Dynamixel; therefore, it is more probable that interruption happens during the operation of the target policy.

To evaluate the methods, we used a methodology similar to the Dynamixel case study. However, instead of collecting the evaluation and learning data separately, we collected all the data at the same time. The data collection process consisted of three phases: learning-data collection, excursion, and recovery (Figure 6). In the learning-data collection phase, the agent followed the behavior policy and the stream of observations and actions was stored to be used for learning offline. In the excursion phase, the agent switched from the behavior policy to the target policy to compute the return for the state at which the switching happened. The probability of starting an excursion at each time step of the learning-data collection phase was 0.01. At the end of each excursion, we recorded the return and the observations for the state from which the excursion was started. This information was used later for evaluation. After the excursion, the agent entered the recovery phase where it followed the behavior policy for a while, to come back to the distribution of the behavior policy. Following this procedure, we collected 30 runs of learning and evaluation data. The data collection process for all 30 runs took about 60 hours. Each run contained 150 excursions. Therefore, each run contained 150 streams of learning data and an evaluation data with 150 samples.

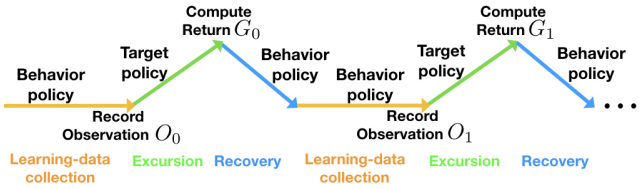


**Figure 6: The process of collecting the learning and evaluation data for the Collision case study.**

After collecting the data, for each run, we concatenated the 150 streams of learning data and applied the algorithms to it to learn the predictions offline. To evaluate the learned predictions, we computed an estimation of MSRE: we computed the squared difference between the learned predictions and returns and computed the average over the evaluation data collected from the 150 excursions.

The learning curves and parameter studies of the asymptotic performance over the step-size for $\lambda = 0$ is shown in Figure 7. We report the results for $\lambda = 0$ because all the methods performed more similarly as $\lambda$ got larger. All the parameters were set to values resulting in the lowest asymptotic performance. To estimate the asymptotic performance, we computed the average of the error for the last 100 time steps of each run and averaged over 30 runs.

According to Figure 7, all methods performed similarly in terms of asymptotic performance. GTD(0) converged to a lower level of asymptotic error and was faster. ETD(0, $\beta$) also achieved a low level of asymptotic error. According to the parameter studies, ETD(0) converged with much smaller values of step-size, but still performed well. The $\beta$ parameter of ETD(0, $\beta$) helped improve ETD(0)'s sensitivity to step-size. TD(0), HTD(0), and ABTD(0) had the best

[1] http://incompleteideas.net/tiles/tiles3.html

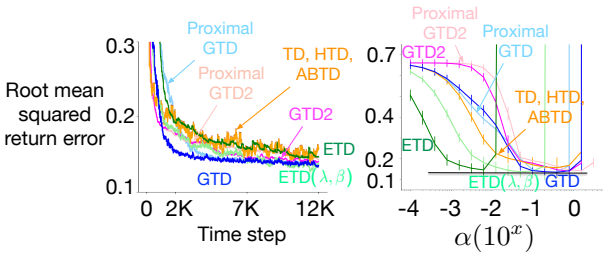


**Figure 7: The learning curves and parameter studies of the asymptotic performance over step-size for $\lambda = 0$ for the Collision case study. All methods performed similarly, but GTD(0) were slightly better in terms of asymptotic error and speed.**

sensitivity to step-size; however, they converged to a higher level of error.

Parameter sensitivity results agree with those of the Dynamixel case study. ETD($\lambda$) achieved the same level of asymptotic error for all values of $\lambda$ (Figure 8). However, all other methods worked better with eligibility traces. GTD($\lambda$) and HTD($\lambda$) were robust to the second step-size and their sensitivity reduced as $\lambda$ got larger. GTD2($\lambda$), however, was more sensitive to the second step-size (Figure 9). Another interesting result is that ABTD($\zeta$) converged for a wider range of step-sizes compared to TD($\lambda$) for high values of $\lambda$ (Figure 8).

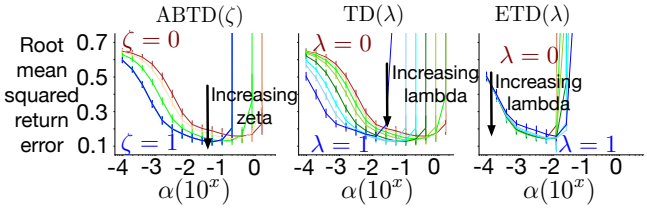


**Figure 8: The parameter studies of the asymptotic performance over step-size for ABTD($\lambda$), TD($\lambda$), and ETD($\lambda$) for the Collision case study.**

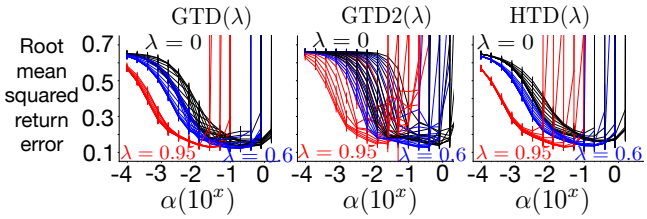


**Figure 9: The parameter studies of the asymptotic performance over step-size for Gradient-TD algorithms for the Collision case study. Each curve corresponds to a specific value of $\lambda$ and second step-size.**

## 6 THE TIME-TO-ALIGN CASE STUDY

In the first two case studies, we focused on solving the prediction problem. In this case study, we consider a control task on the Kobuki. The task is to learn to align with the Kobuki's charging station as fast as possible. To do this, the Kobuki can turn left and right in place (Figure 10). There are two sensors available to the robot to learn this control task. One is the infrared receiver at the front of the Kobuki which produces a unique reading when the Kobuki is aligned with the charging station. The other is the Kobuki's gyroscope which provides the orientation of the robot; turning left and right increases and decreases the orientation respectively. It takes about 100 steps for the robot to complete a cycle with the velocity at which it is turning. We formulated this control task as a single GVF. The target policy was greedy with respect to the action-value function. The horizon function returned 0 when the Kobuki was aligned with the charging station, otherwise, it returned 1. The cumulant function always returned $-1$. Given that the cumulant is always $-1$ and the policy is greedy, the agent is trying to minimize the number of time steps till aligning with the charging station. Therefore, we call this task the Time-to-align control task.

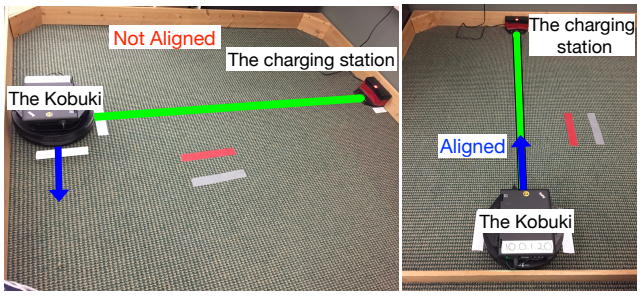


**Figure 10: The Kobuki alignment with the charging station.**

To learn this GVF we used a linear function with tile coded features. To construct the feature vector, we used the orientation of the Kobuki which was a real value in $(-1, 1)$ that wraps around 1. We fed the orientation to a wrap tile-coder with 16 tilings each with 4 tiles. (We set the size of the tile-coder's hash table to 4096.)

We used Q-learning, Greedy-GQ($\lambda$), and Greedy-ABQ($\zeta$) methods. We made several instances of the algorithm each corresponding to a parameter setting. Step-size was in the form $\frac{\alpha}{\text{number of tilings}}$ where $\alpha \in \{0.1 \times 2^x | x = -4, -3, ..., 1\}$. $\lambda$ and $\zeta$ were a number in $\{0, 0.5, 0.9\}$ for Greedy-GQ and Greedy-ABQ. Second step-size was in the form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{0.0625, 0.25, 1, 4\}$.

We used a behavior policy that selected between turning left and right randomly at each time step with a 90 percent bias toward repeating the previous action.

There were certain issues that made performing this experiment challenging. First, the Kobuki was supposed to stay in its place and only turn left and right; however, after turning left and right for a while, it would slightly slide toward one direction. Second, the orientation readings were not reliable; therefore, we would get different readings for the same orientation over time. These two issues introduced non-stationarity to the problem. To deal with this non-stationarity, we collected small batches of data and manually

resolved the non-stationarity when starting the collection of a new batch. To address the sliding problem, we moved the robot manually to the starting position. To address the unreliability of the orientation readings, we offset the readings by the value of the orientation at which the Kobuki was aligned with the charging station. By performing this offsetting, we made sure that the orientation of the robot when it was aligned with the charging station was kept at a value around 0.

We collected 80 batches of data each of size 1000 time steps by following the behavior policy. Collecting each batch took about 100 seconds; however, readjusting the robot manually made the data collection phase take much longer. Therefore, the collection of all batches took about 5 hours. We used 20 batches of data for each run, resulting in 4 runs of size 20,000 time steps. For half of the batches, the Kobuki started from the orientation at which it was aligned with the charging station and for the other half, it started from roughly the opposite orientation. After collecting the data, we applied different instances of the algorithms to learn the task offline.

A natural approach for evaluating how well an algorithm has learned a robot control task is to test the policy that it has learned on the robot and compute the return. However, to systematically compare the algorithms, lots of tests are required and running tests on robots is expensive because it requires time in the real world. For this case study, a large number of tests were required because we had many instances of each method, each corresponding to a parameter setting. For example, Greedy-GQ($\lambda$) had 72 different parameter settings. In addition, we had 4 different runs of learning data and we wanted to do multiple tests each starting from a different starting orientation. Moreover, to get an estimation of how good each algorithm had learned over time, we had to evaluate the policy that it had learned after different number of time steps. To get an estimation of how many tests are needed to evaluate Greedy-GQ let's suppose we want to do the evaluation from 4 different starting orientations and after 4 different number of time steps. We would need to run $72 \times 4 \times 4 \times 4 = 4608$ tests which is not feasible.

To reduce the high number of evaluations, we decided to select one parameter setting for each of the algorithms that seemed to be a reasonably good setting. We selected the parameter setting for each algorithm by looking at the action-value plots of them for different settings (e.g., Figure 11). These plots show the action-value function of the learned policy after 20,000 time steps, for different orientations and actions, and averaged over 4 runs.

To understand the action-value plots lets look at the two examples shown in Figure 11. The $x$ axis is the orientation and the $y$ axis is the action-value function for action $a$. Each plot contains two lines. The red one corresponding to turning right and the blue one corresponding to turning left. Given that the goal of the robot is to minimize the number of steps till aligning with the charging station and that the alignment happens at an orientation about 0, the optimal policy is to move toward orientation 0. Therefore, when the orientation of the robot is in the range $[0, 1]$, it should turn right, that is to move toward smaller values of orientation, and turn left otherwise. Moreover, the closer an orientation is to 0, the smaller is the number of steps needed till alignment. Therefore, Example 2 is a properly learned action-value function. However, Example 1 is a poorly learned action-value function: if the robot follows the greedy policy with respect to this action-value function, at orientation $-0.5$, it would turn right toward smaller values of orientation instead of turning left towards orientation 0.

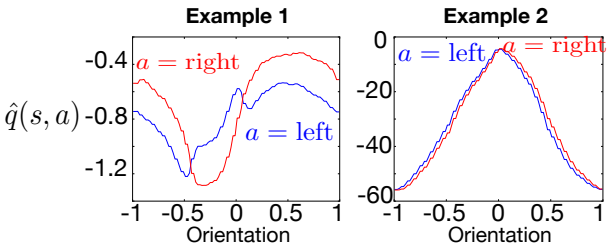


**Figure 11: Examples of action-value plots.**

We selected a reasonable parameter setting for each of the algorithms by inspecting the action-value plots. For Q-learning, the step-size was set to 0.05. For Greedy-GQ, the step-size, second step-size, and trace parameter were set to 0.05, $4 \times 0.05$, 0.5 respectively. For Greedy-ABQ, the step size and the trace parameter were set to 0.0125 and 0.5 respectively.

To evaluate the methods, we ran their target policy, the greedy policy with respect to the learned action-value functions, on the robot. We considered the action-value functions that had been learned after different number of time steps. We call these time steps, evaluation points. For this experiment the evaluation points were 1000, 5000, 10000, 20000. To evaluate the learned policy for each evaluation point and run, we ran 4 tests each starting from a different orientation and calculated the return. The length of the tests were 250 time steps. The starting orientations are shown in Figure 12. For an optimal policy, the average return from the 4 different starting orientations should be around $-30$. This whole process of evaluating the algorithms considering the manual readjustment of the robot took about 7 hours.

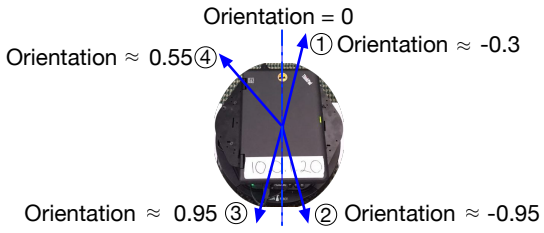


**Figure 12: The starting orientations for evaluation.**

Figure 13 summarizes the results. The $x$ axis represents the evaluation points. The $y$ axis is the average return over starting orientations, where the return is the number of steps to alignment negated. Each plot contains 5 lines with each dotted line corresponding to one run and the solid line showing the average of performance over the 4 runs. According to Figure 13, all methods learned to turn in the direction that would achieve the goal fastest within 20,000 time steps. Greedy-GQ($\lambda$), however, outperformed the other algorithms in the mean and learned the optimal policy faster.

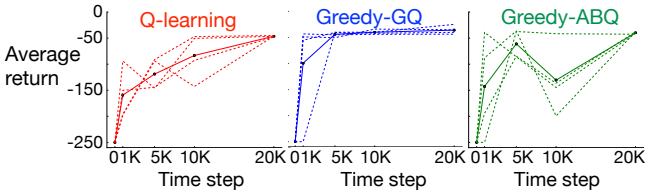


**Figure 13: Comparison of the algorithms on the Time-to-align control task. The *y* axis is the average return over starting orientations. Each dotted line corresponds to one run and the solid line shows the average over the 4 runs. Greedy-GQ outperformed the other methods.**

## 7 OFF-POLICY EVALUATION ON ROBOTS: PROPOSED METHODOLOGY

As mentioned in Section 1, conducting meaningful comparisons of off-policy algorithms on robots can be challenging. In this section, we discuss some of these challenges and the methodology that we developed in our case studies to address them. Our methodology was effective, but much more is left to know about systematic comparisons of off-policy learning in large-scale domains.

One challenge of evaluating off-policy algorithms is that when learning off-policy, the data is generated using a policy different from the policy that the agent learns about. Therefore, we cannot evaluate the algorithms using the conventional schemes of following the policy till termination and calculating the return. Moreover, in robot domains, we do not have access to the true value function and cannot use the difference between the estimated and true value function as the performance measure. To deal with these problems in the Dynamixel and Collision case studies, we collected several samples from the behavior policy and computed the return corresponding to those samples by following the target policy. All this happened prior to the learning phase. To evaluate the algorithms, we computed the squared difference between the learned predictions and previously computed returns and computed the average over all samples. Using this methodology, we efficiently estimated MSRE and effectively compared the algorithms.

Another challenge of evaluating algorithms on robots is that each time step requires time in the real world and human supervision. To conduct a fair comparison, extensive parameter sweeps are required over many independent runs. Therefore, computation and evaluation should be conducted efficiently. We developed a methodology that can be employed to efficiently evaluate policy evaluation methods on real-world problems. The methodology, however, heavily relies on following a fixed policy throughout the experiment and thus is not applicable to the control case with rapidly changing policies. To get around this issue in the control case, we proposed a systematic approach that reduced the required time for a control experiment significantly. In our proposed method, we first selected a reasonable parameter setting for each method. This removed the need to evaluate each method for various parameter settings and reduced the number of required experiments. Moreover, we proposed using only a subsample of the time steps to evaluate a method, meaning that we first select a number of time steps and evaluate the methods only at those time steps instead of evaluating them at all time steps. This produces learning curves like the ones shown in Figure 13. Although our evaluation methodology was applicable to our control problem, better practices are required for conducting comparative studies on more complicated control robot domains. A promising future direction is to use off-policy policy evaluation methods (Thomas and Brunskill, 2016; Jiang and Li, 2015) to evaluate the learned policy using the data gathered by the behavior policy.

Another challenge of evaluating off-policy learning on robots is that in off-policy learning, a system can potentially learn about thousands of GVFs in parallel; however, it is not clear how we can evaluate a large collection of GVFs each about a specific policy that is potentially different from the behavior policy and is never executed during the normal operation. The evaluation methodology that we developed for the case of policy evaluation might be effective if many of the GVFs share their policy. However, more effective evaluation techniques are required in cases which we want to learn about many different policies. Another challenge of evaluating a large collection of GVFs is that it is not clear how we can get an overall assessment of how well the entire collection was learned. Some of the GVFs may be easier to learn and some may need more data. We need evaluation methodologies that give a fair assessment of the overall performance of the system.

## 8 CONCLUSION

In this work, we presented three case studies in which we empirically compared several modern off-policy GVF learning algorithms on robot domains. Based on our results, Emphatic-TD methods exhibit an advantage over other methods both in the asymptotic performance and speed of learning when the behavior is close to on-policy. Gradient-TD methods, on the other hand, perform better than ETD when the behavior is more random. Our results also suggest that use of eligibility traces improve the performance of all prediction methods. This suggests an advantage of linear function approximation which can be easily combined with eligibility traces compared to neural network representation which due to computational reasons cannot easily be combined with eligibility traces. In GVF control, Greedy-GQ showed substantially less variance across test runs compared to Q-learning; this suggests that Greedy-GQ could be a good alternative to popular Q-learning, having both convergence guarantees and better performance. Finally, in none of our experiments, ABQ which performs off-policy learning without importance sampling showed an advantage over other methods.

We also argued that conducting fair and meaningful comparative studies of off-policy learning on robots is challenging and currently there are no well-established practices for performing such studies. We developed an evaluation methodology that was successful and applicable to relatively complicated robot domains. However, there is much left to know about how to do evaluation on large-scale real-world domains where we want to make predictions about thousands of GVFs each conditioned on a different way of behaving.

## ACKNOWLEDGMENTS

The authors gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada, Alberta Innovates—Technology Futures, and Google DeepMind.

## REFERENCES


[1] Baird, L. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings*, pp. 30–37.

[2] Boyan, J. A. (2002). Technical update: Least-squares temporal difference learning. *Machine learning*, 49(2-3), 233-246.

[3] Bradtke, S. J., Barto, A. G. (1996). Linear least-squares algorithms for temporal difference learning. *Machine learning*, 22(1-3), 33-57.

[4] Dann, C., Neumann, G., Peters, J. (2014). Policy evaluation with temporal differences: A survey and comparison. *The Journal of Machine Learning Research*, 15(1), pp. 809–883.

[5] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S. (2018). IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures. ArXiv:1802.01561.

[6] Geist, M., Scherrer, B. (2014). Off-policy learning with eligibility traces: A survey. *The Journal of Machine Learning Research, 15(1)*, 289-333.

[7] Ghiassian, S., Patterson, A., White, M., Sutton, R. S., White, A. (2018). Online Off-policy Prediction. Arxiv:1811.02597.

[8] Gu, S., Holly, E., Lillicrap, T., Levine, S. (2017, May). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396. IEEE.

[9] Hackman, L. (2012). *Faster Gradient-TD Algorithms*. MSc thesis, University of Alberta, Edmonton.

[10] Hallak, A., Tamar, A., Munos, R., Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias- variance analysis. In *AAAI*, pp. 1631–1637.

[11] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. ArXiv:1611.05397.

[12] Jiang, N., and Li, L. (2015). Doubly robust off-policy value evaluation for reinforcement learning. arXiv preprint. Arxiv:1511.03722.

[13] Littman, M. L., Sutton, R. S. (2002). Predictive representations of state. In *Advances in neural information processing systems*, pp. 1555–1561.

[14] Liu B, Liu J, Ghavamzadeh M, Mahadevan S, Petrik M (2015). Finite-Sample Analysis of Proximal Gradient TD Algorithms. In *Proceedings of the 31st International Conference on Uncertainty in Artificial Intelligence (UAI-2015)*, pp. 504–513. AUAI Press Corvallis, Oregon.

[15] Maei, H. R. (2011). *Gradient Temporal-Difference Learning Algorithms*. PhD thesis, University of Alberta, Edmonton.

[16] Maei, H. R., Szepesvari, C., Bhatnagar, S., Sutton, R. S. (2010, June). Toward off-policy learning control with function approximation. In *ICML*, pp. 719–726.

[17] Mahadevan, S., Liu, B., Thomas, P., Dabney, W., Giguere, S., Jacek, N., Gemp, I., Liu, J. (2014). Proximal reinforcement learning: A new theory of sequential decision making in primal-dual spaces. ArXiv:1405.6757.

[18] Mahmood, A. R., Yu, H., Sutton, R. S. (2017). Multi-step off-policy learning without importance sampling ratios. ArXiv:1702.03006.

[19] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature, 518(7540)*: 529–533.

[20] Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M. (2016). Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062.

[21] Precup, D., Sutton, R. S., Singh, S. (2000). Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning*, pp. 759–766. Morgan Kaufmann.

[22] Rafols, E., Koop, A., Sutton, R. S. (2006). Temporal abstraction in temporal-difference networks. In *Advances in neural information processing systems*, pp. 1313?1320.

[23] Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Van de Wiele, T., Mnih, V., Heess, N., Springenberg, J. T. (2018). Learning by Playing-Solving Sparse Reward Tasks from Scratch. ArXiv:1802.10567.

[24] Ring, M. B. (in preperation). Representing Knowledge as Predictions (and State as Knowledge).

[25] Schaul, T., Horgan, D., Gregor, K., Silver, D. (2015, June). Universal value function approximators. *In International Conference on Machine Learning*, pp. 1312–1320.

[26] Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., Degris, T. (2017). The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3191–3199. JMLR. org.

[27] Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. In *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Experiences*.

[28] Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. 2nd edition in preparation.

[29] Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, 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*, pp. 993–1000, ACM.

[30] Sutton, R. S., Mahmood, A. R., White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *Journal of Machine Learning Research 17*(73):1–29.

[31] Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., Precup, D. (2011, May). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proceeding of the tenth International Conference on Autonomous Agents and Multiagent Systems*, pp. 761–768, Taipei, Taiwan.

[32] Sutton, R. S., Tanner, B. (2005). Temporal-difference networks. In *Advances in neural information processing systems*, pp. 1377–1384.

[33] Tanner, B., and Sutton, R. S., (2005). TD($\lambda$) networks: temporal-difference networks with eligibility traces. In *Proceedings of the 22nd international conference on Machine learning*, pp. 888?895.

[34] Thomas, P., and Brunskill, E. (2016, June). Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pp. 2139–2148.

[35] Touati, A., Bacon, P. L., Precup, D., Vincent, P. (2018). Convergent tree-backup and retrace with function approximation. ArXiv:1705.09322v4.

[36] Van Hasselt, H., Doron, Y., Strub, F., Hessel, M., Sonnerat, N., Modayil, J. (2018). Deep Reinforcement Learning and the Deadly Triad. ArXiv preprint:1812.02648.

[37] Van Seijen, H., Fatemi, M., Romoff, J., Laroche, R., Barnes, T., Tsang, J. (2017). Hybrid reward architecture for reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 5392–5402.

[38] Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge.

[39] White, A. (2015). *Developing a predictive approach to knowledge*. PhD thesis, University of Alberta, Edmonton

[40] White, A., White, M. (2016, May). Investigating practical linear temporal difference learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 494–502, International Foundation for Autonomous Agents and Multiagent Systems.