Predictive Knowledge in Robots: An Empirical Comparison of Learning Algorithms

by

Banafsheh Rafiee

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Knowledge is central to intelligence. Intelligence can be thought of as the ability to acquire knowledge and apply it effectively. Despite being a subject of intense interest in artificial intelligence, it is not yet clear what the best approach is for an intelligent system to acquire and maintain a large body of knowledge. One interesting approach that we pursue in this thesis is based on the view that much of world knowledge is predictive. For example, to know that a box is heavy, is to predict that we need lots of effort to lift it. We call this predictive approach to maintaining and acquiring knowledge, the *predictive knowledge approach*. In this thesis, we implement an instance of this approach in order to explore and assess it further. To do so, we build upon the techniques and ideas of reinforcement learning. In particular, we use the idea of value functions. In conventional RL, value functions capture predictions about reward. Recently, value functions have been extended to capture more general predictions which can constitute knowledge. A value function in the extended form is called a general value function (GVF). GVFs provide a language to talk and think about predictions. More generally, we can think of GVFs as a language for representing predictive knowledge.

In this thesis, we develop the predictive knowledge view using the language of GVFs and apply it to several robot domains. Our work has three main contributions. First, we contribute to the idea of predictive knowledge by providing several new examples of it on robot domains, gaining a more substantive understanding of knowledge as predictions. Second, we perform empirical comparisons of many off-policy temporal-difference (TD) learning algorithms including gradient-TD and emphatic-TD families of methods on robot data. Third, we systematically study the learning process on robots. Such studies provide insights about how to effectively evaluate and compare algorithms on real-world systems.

Preface

Parts of one of the experiments appeared in a paper in preparation by Ghiassian et al. (2018). However, all the experiments were originally produced for this thesis. The writing of the thesis was also independent of the work by Ghiassian et al. (2018).

Acknowledgements

First and foremost, I would like to thank my supervisor, Richard Sutton. He taught me to talk and think about my work clearly. Working with him was inspiring and an honor.

I am grateful to my co-supervisor, Adam White. He taught me to make sure that I am making progress when working on a big project. He helped me get unstuck when facing difficulties working with robots.

I would like to thank the members of my examining committee, Michael Bowling and Patrick Pilarski, for taking the time to read my dissertation and improving it in many different ways.

I owe many thanks to Martha Steenstrup who taught me important things about writing effectively and helped making my writing more clearly understood.

I am also thankful to Joseph Modayil for reading my dissertation and offering comments.

I would like to thank Sina Ghiassian for sharing his experience on performing empirical studies with me and for useful discussions.

I am thankful to the members of the Robot Learning Project for developing a software for doing experiments on the Kobuki robot. I learned a lot and enjoyed my time, working with them on the Robot Learning Project.[1]

I thank all the members of RLAI lab, specifically, Dylan Ashley and Eric Graves for helping me with writing and proofreading my work.

[1]https://amiithinks.github.io/RobotLearning/

Contents

List of Figures

Chapter 1

Introduction to the Predictive Knowledge Approach

In this work, we consider learning predictions that can constitute knowledge. While the big goal is to investigate the idea of general value function (GVF) for learning predictive knowledge, the more concrete goal of this work is to empirically compare algorithms for learning predictions in the form of a GVF. In this chapter, we introduce the predictive knowledge approach that is an approach for acquiring and maintaining knowledge. We also briefly describe the idea of GVF which can be used to develop an instance of the predictive knowledge approach. In the end, we discuss the three contributions of this work.

Understanding how world knowledge should be represented, acquired, and maintained is crucial to artificial intelligence. Much of what we believe to be world knowledge can be thought of as predictions. As an example, to know that a cup of coffee is hot is to predict that if we touch it, it will burn our hands. We can make such a prediction because we have burned our hands many times in similar situations and we have noticed certain regularities when interacting with hot cups of coffee. For example, whenever a cup of coffee had steam rising from it and we touched it, it burned our hands. To know the world is to know these regularities and to be able to predict what happens.

In this thesis, we subscribe to this predictive knowledge approach that is based on the view that much of world knowledge is predictive. A great advantage of viewing knowledge as predictions is that predictive knowledge can be verified by what really happens in experience. In other words, a learning system can check the correctness of its predictions based on how well they match with what it observes.

The predictive knowledge approach creates special requirements for our predictions and learning algorithms. First, predictions need to be grounded in experience. By experience, we mean the low-level sensorimotor data acquired through interaction with the world. More specifically, experience is a temporal stream of sensations and actions. It is natural for the predictions to be translatable into statements about this stream, considering that knowledge can be thought of as regularities in experience.

Second, predictions have to be about the very own experiences of the learning system and do not need to be interpretable universally or by people. The predictive approach emphasizes the experience of the learner and encourages a subjective view of knowledge as opposed to an objective one. For example, for a robot to learn how far it is from the wall, it does not need to understand distance in terms of meters, but it can have a sense of closeness based on how many times its wheels turn until it reaches the wall. In other words, the robot can understand everything based on its own body, sensors, and actuators. This view seems natural if we think of human early years. An infant has a sense of distance not in terms of universal unit systems, but in terms of its own low-level sensorimotor experiences.

Third, predictions need to be expressive. In order to constitute knowledge, predictions have to express many aspects of the world including both low-level pieces of knowledge and high-level concepts.

Fourth, predictions should be suitable to be used by reasoning and planning procedures. We have been talking about representing knowledge as predictions, but it is also important to be able to use this knowledge. We want to represent knowledge in a form such that reasoning and planning procedures could easily use it to model the world and generate and improve behavior.

Fifth, predictions need to be both efficiently learnable and updated online. We want to learn and represent many aspects of the sensorimotor data. To do so, we need a massive number of predictions, each capturing a regularity in the data. To learn a massive number of predictions, we need algorithms that are efficient both in terms of memory and computation. Moreover, the predictions need to be continually adapted during the interaction of the learning system with the world. Therefore, we need algorithms that can use the data gathered during interaction with the world to learn, called online algorithms.

To represent and learn predictive knowledge, Sutton et al. (2011) proposed generalizing an existing idea in reinforcement learning (RL) called a value function. In conventional reinforcement learning, there is an agent interacting with its environment. The agent can perform different actions and the environment responds to the agent by emitting rewards. Value functions capture predictions about the reward signal. More specifically, value functions give an estimate of how much reward the agent receives from each state given that it behaves in a specific way. While value functions are about the reward signal, there is an extended form of value function that can be about any signal in the world and is called general value function or GVF. GVFs can be thought of as a language for representing predictive knowledge.

Reinforcement learning framework and in particular general value functions are well suited for predictive knowledge because they meet the aforementioned requirements. First, RL is based on experience and seeks to find the regularities in the sensorimotor data. Second, in RL, learning happens without supervision and based on the very own experiences of the agent. Third, the language of GVFs is capable of expressing many aspects of the world given that each prediction can be about any signal. Moreover, high-level pieces of knowledge can be constructed from low-level ones. Fourth, RL makes predictions that are commonly used for planning and reasoning procedures. Fifth, RL is equipped with temporal-difference (TD) methods that are suitable for learning a massive number of predictions efficiently and online.

There are other classical approaches addressing the challenge of representing and maintaining knowledge. Many of these approaches hold an objective view of knowledge meaning that there are certain correct statements about the world that are universally true. To be able to maintain knowledge using such approaches, the system would either require human experts or logical inference. It is evident that as the number of statements grow, maintaining knowledge using these approaches become infeasible. In other words, the classical approaches may suffer from lack of scalablility. The predictive knowledge approach, on the other hand, is potentially scalable because the correctness of the statements can be automatically verified by the learning system itself, during learning, and based on how well they match with what actually happens.

The idea of representing knowledge as predictions has been studied in other fields and is not limited to artificial intelligence. Drescher (1991) presented a predictive mechanism for building an understanding of the world. This mechanism was inspired by the work of the psychologist Jean Piaget who proposed that the new-born infant understands the world exclusively in terms of sensations and actions. The ability to make predictions about the sensory data has often been argued to have a key role in perception (Friston, 2005; Clark, 2013).

In this work, we take another step forward in investigating the suitability of the GVF language for learning predictive knowledge. Our work has three main contributions. Our first contribution is to give several new examples of learning predictive knowledge on robot domains. Providing these examples, we verify that robots can effectively use GVFs to acquire knowledge about their interaction with the world.

The second contribution of this work is an empirical comparison of many off-policy TD algorithms, learning predictive knowledge on robot domains. Off-policy learning can be explained as learning about one way of behaving while behaving in another way. Using off-policy learning, a system can learn predictions about different ways of behaving while behaving in one specific way. This property of off-policy learning is particularly suitable for the predictive knowledge approach. A handful of off-policy TD algorithms have been proposed in the past couple of years. However, there are not many studies investigating them, especially on robot data. White and White (2016) performed an empirical comparison of many off-policy TD algorithms on simulated domains, providing insights about the strengths and weaknesses of them in different situations. In this work, we perform similar empirical comparisons on robot domains. Our work is not the first to study off-policy learning on robot domains. Delp (2011) investigated GQ(λ), one of the most important off-policy algorithms, learning several optimal policies on a mobile robot. White (2015) also presented an extensive investigation of GTD(λ) and GQ(λ) algorithms on many robot domains. While these two studies focused on GTD(λ) and GQ(λ), we consider many algorithms, presenting the first empirical comparison of off-policy TD algorithms on robots.

The third contribution of this thesis is to gain a better understanding of how we can do systematic studies on robot domains. Studying the learning algorithms on robots is valuable. However, there are not many systematic studies on such domains. There are many reasons why performing such studies is challenging. First, robots are often unreliable. Second, gathering enough data so as to be able to extract meaningful results from the experiment is time-consuming. Third, there are many factors that affect the learning process, but unlike simulations, we do not have control over them. Finally, evaluating the learning process is more challenging on robots.

This thesis consists of 9 chapters. In the second chapter, we provide the background on reinforcement learning. In the third chapter, we discuss the generalized form of value functions that can be used to learn predictive knowledge. Chapter 4 and 5 discuss the robots and algorithms that we used in our experiments respectively. In Chapter 6 to 8, we present our three case studies that make up the primary contribution of this thesis. We will close the thesis in the conclusion chapter.

Chapter 2

Background on Reinforcement Learning

In this chapter, we provide background on reinforcement learning, setting the context for understanding the rest of the thesis. We start by discussing the agent-environment interaction loop and introducing basic concepts such as returns, policies, and value functions. Then we discuss linear function approximation and introduce temporal-difference learning. We close the chapter by a discussion of off-policy learning and its challenges.

Readers familiar with basic reinforcement learning concepts and off-policy learning can skip this chapter as the purpose of this chapter is to only provide the background on those topics.

2.1 The Agent-Environment Interaction Loop

Reinforcement learning (RL) is learning from interaction. The problem of RL is formalized using Markov decision processes (MDP) where there is an interaction loop between the learner and its world. The learner is called the agent and its world is called the environment. The agent and environment interact with each other continually in discrete time steps. At each time step t, the agent is in a state $S_t \in \mathcal{S}$ and performs an action $A_t \in \mathcal{A}$. The environment responds to the action by taking the agent to the next state $S_{t+1} \in \mathcal{S}$ and providing it with a reward signal $R_{t+1} \in \mathcal{R}$. This continual process produces a sequence of states, actions, and rewards:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

One main property of finite MDPs, where the sets of states, actions, and rewards are finite, is that the transition to the next state and reward only depends on the current state and action. This property is called the Markov property. According to the Markov property, the current state and action contain enough information to determine the probability of the next state and reward. Because of the Markov property, the dynamics of the environment can be fully summarized by this probability function:

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$$

2.2 Returns, Policies, and Value Functions

In RL, the agent seeks to maximize the total amount of reward it receives in the long run. More specifically, the agent wants to maximize the expectation of a weighted sum of the rewards. This weighted sum is called the return. We denote the return from time steps t by G_t:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The rewards are weighted by powers of a continuous parameter $0 \leq \gamma \leq 1$, called the discount factor. The discount factor determines how less valuable a reward signal will be if it is received in the future compared to if it was received immediately.

The expectation of the return from each state provides the agent with an estimate of how good the state is. This expectation is called the value function. Value function at each state depends on two things. First, it depends on the way that the agent will behave in the environment. Second, it depends on the reward signal that the environment emits as a response to the agent's behavior.

The agent's way of behaving is called its policy. Policies determine the probability of choosing each action at each state; they are functions of state and action and are denoted by π where $\pi : |\mathcal{S}| \times |\mathcal{A}| \rightarrow [0, 1]$.

With the notions of return and policy, we can define value functions more formally. The value of a state given a specific policy π, is defined as the expectation of the return under that policy.

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

We can define the value of a state action pair under a specific policy similar to the value of state as the expectation of return under that policy. Value of a state action pair is denoted by $q_\pi(s, a)$.

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

In RL, there are two main categories of problems that we want to solve: policy evaluation and control. In policy evaluation problems, we seek to estimate the value function for a given policy. In the control problem, we want to learn an optimal policy that maximizes return. Solving the control problem involves a policy improvement step in addition to estimating the value function of the learned policy. In this thesis, we consider both the policy evaluation and control problems.

2.3 Function Approximation

To estimate the value function, we first need to determine how to represent it over the state space. In the simplest case, we can represent each state directly. In that case, we would estimate the value function for each state individually. This kind of representation is called tabular. In many cases, estimating the value function for each state is not practical, either because the number of states is large or the state space is continuous. In these cases, we need to generalize value function over groups of states. This can be done using function approximation.

Function approximation is a method to generalize a function using samples of its input and output in order to approximate it over the entire input space. In the case of estimating the value function, the input space is the state space.

Linear function approximation is one simple and important example of function approximation. In linear function approximation the approximator is a linear function of a weight vector w and the state is represented as a vector of features $\mathbf{x} = (x_1(s), x_2(s), ..., x_d(s))^T$ where d is the number of features. In this case the estimate of the value function is $\hat{v}(s; \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s)$.

Tile coding is a feature construction mechanism that uses multiple overlapping partitions of the state space to produce features. These partitions are called tilings. Each tiling is broken into many parts each called a tile. In the simplest case, tile coding has only one tiling: a simple discretization of the state space. See left part of Figure 2.1[1]. Each state falls into a specific tile of the tiling; in other words, each state activates the feature corresponding to one tile. In this case, generalization happens only between the states that are in the same tile.

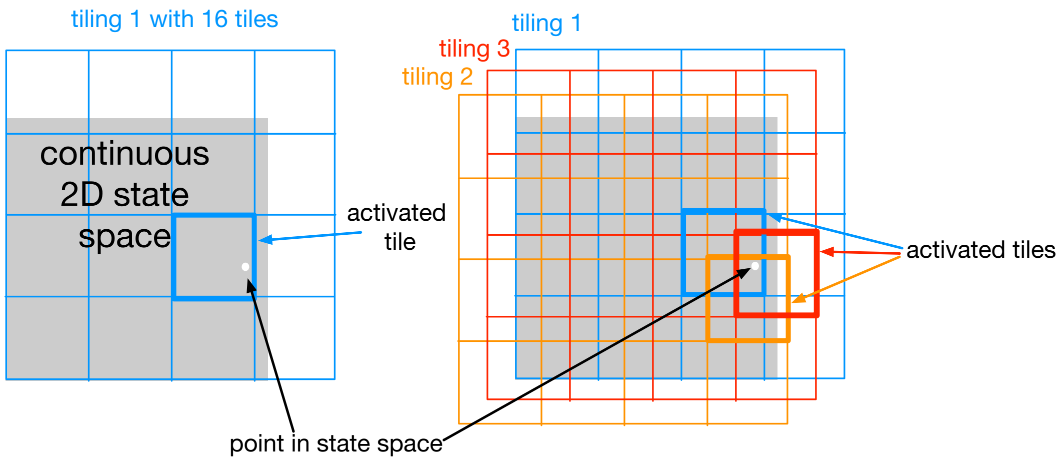


Figure 2.1: An example of how a point in a 2D state space activate different tiles of different tilings

The strength of tile coding becomes clear when multiple tilings are used. In this case, each state activates multiple tiles each belonging to one of the tilings. To better understand this case, see the example shown on the right side of Figure 2.1. In this example, we are using 3 different tilings and we have shown which tile of each tiling has been activated for a point in the state space. After finding the activated tiles for each tiling, tile coding produces

[1]This figure is adapted from Sutton and Barto (2017)

a feature vector with one feature corresponding to each tile of each tiling. Having multiple active features helps generalization happen between states with common features. For more detail on tile coding look at Sutton and Barto (2017).

2.4 Temporal-difference Learning

To estimate the value function without a model of the environment's dynamics, the learner has to directly use experience, that is sample states, actions, and rewards generated from interaction with the environment. One natural approach for estimating the value of a state directly from experience, is to sample and average returns from that state. This approach is taken in a family of methods called Monte Carlo methods. The Monte Carlo methods can only be applied to settings in which experience can be divided to subsequences and return is defined as $G_t = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$, where T is a final time step. One main drawback of Monte Carlo methods is that they need to wait till the end of each episode and then update their estimates. While Monte Carlo methods are not incremental in a step-by-step sense, there are a family of methods that are fully incremental and also estimate the value function from experience, called temporal-difference (TD) learning methods (Sutton, 1988).

Temporal-difference learning methods can update their estimates immediately after each state transition by using other learned estimates. To understand how TD methods can do this, lets take a look at the general update rule used in reinforcement learning in the case of function approximation:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha[U_t - \hat{v}(S_t, \mathbf{w}_t)]\nabla\hat{v}(S_t, \mathbf{w}_t)$$

where we denote the weight vector at time step t by $\mathbf{w}_t$. $\hat{v}(S_t, \mathbf{w}_t)$ is the learned estimate of the value of state S_t and α is the learning rate. U_t denotes the target of learning that is an estimate of the value of S_t.

The main difference between the Monte Carlo methods and TD methods can be found in their target of learning. Monte Carlo methods use the return, G_t, as the target of learning. However, TD methods use $R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t)$

as the target. The target of TD can be formed immediately after each state transition using the current estimates of the value of the next state.

The simplest TD method is called TD(0) and in the case of linear function approximation can be summarized as:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{x}(S_t) \end{aligned} \tag{2.1}$$

δ_t is called the TD error and can be thought of as an error because it is the difference between the learned value of S_t and another more accurate estimate of it that is $R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t)$.

The Monte Carlo and TD(0) methods can be unified to a more generalized class of methods by setting the target of learning to an intermediate value between the target of Monte Carlo methods, G_t, and the target of TD(0) that is $R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}, \mathbf{w}_t)$. This target is in the form of:

$$R_{t+1} + \gamma_{t+1}R_{t+2} + ... + \gamma^{n-1}\hat{v}(S_{t+n}, \mathbf{w}_{t+n-1})$$

where n denotes the number of intermediate rewards that we use directly in the target and $\hat{v}(S_{t+n}, \mathbf{w}_{t+n-1})$ is used in the target as a proxy to the remaining rewards. This target is called the n-step return. Computing the n-step return requires information that becomes available only after n steps.

In addition to setting the target of learning to an n-step return, we can set the target of learning to any average of n-step returns. Such a target is used in the TD(λ) algorithm. TD(λ) also uses a mechanism called eligibility traces that makes it possible to do updates continually and without delays. The TD(λ) algorithm can be summarized as:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\ \mathbf{z}_t &= \gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \end{aligned} \tag{2.2}$$

where $\mathbf{z}$ is a vector of the same size as the weight vector $\mathbf{w}$ and is called 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 λ. There are two extreme cases of $\lambda = 0$ and $\lambda = 1$. In the case of $\lambda = 0$, the algorithm becomes the same as TD(0). $\lambda = 0$ is also known as the case of full bootstrapping. In the case of $\lambda = 1$, the algorithm becomes equivalent to Monte Carlo methods.

2.5 Off-policy Learning

Off-policy learning is to learn about a policy using data generated from another policy. The policy that we want to learn about is called the target policy and is denoted by π and the policy that generates the data is called the behavior policy and is denoted by b. Using off-policy learning we can have a more exploratory behavior policy that chooses non-optimal actions, while we learn about an optimal policy. Moreover, off-policy learning makes it possible to behave in one specific way while learning about many different ways of behaving.

In order to learn about the target policy based on the data generated by the behavior policy, we have to correct for the difference between the two policies. To do so, we use the ratio of the probability of choosing an action under the target and behavior policy. This ratio is called the importance sampling ratio:

$$\rho_t = \frac{\pi(A_t|S_t)}{b(A_t|S_t)}$$

Adding the importance sampling ratio to the linear TD(λ), we get the off-policy linear TD(λ):

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\ \mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \qquad (2.3) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \end{aligned}$$

TD(λ) has been shown to be unstable in conjunction with linear function approximation under off-policy training (Baird,1995). Many algorithms have been proposed to solve the instability of TD learning under off-policy training. In this thesis, we consider many of these algorithms and review them in Chapter 5.

As we discussed in Section 2.2, RL problems can be divided into two main categories of prediction and control. Till this point, we have been focusing on solving the prediction problem. However, both the ideas of off-policy and TD learning can be applied to RL control problems. A well-known off-policy TD algorithm for control problems is Q-learning introduced by Watkins (1989). The Q-learning algorithm in the case of function approximation can be specified by equations 2.4:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\hat{\mathbf{x}}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{x}(S_t) \end{aligned} \tag{2.4}$$

where $\hat{\mathbf{x}}(S_{t+1}) = \mathbf{x}(S_{t+1}, A^*_{t+1})$ and $A^*_{t+1} = \text{argmax}_a \mathbf{w}_t^T\mathbf{x}(S_{t+1}, a)$.

Despite having nice results on many domains, including Atari games (Mnih et al., 2016), Q-learning does not have convergence guarantees in the case of function approximation and is potentially divergent. In this work, we consider two other off-policy TD control algorithms, in addition to Q-learning, with nice convergence properties. A discussion of these algorithms will be presented in Chapter 5.

2.6 Summary

In this chapter, we provided the background useful for understanding the rest of the thesis. We started by discussing basic concepts in reinforcement learning. Then, we covered topics of linear function approximation and temporal-difference learning. We closed the chapter by a discussion on off-policy learning.

Chapter 3

Pursuing the Predictive Knowledge Approach Using General Value Functions

As we discussed in Chapter 1, we subscribe to the predictive knowledge approach. This approach is based on the view that knowledge is regularities in one's interaction with the world and to know the world is to know these regularities and predict them. In this work, we pursue the predictive knowledge approach using general value functions (GVFs).

In this chapter, we discuss how general value functions are related to conventional value functions and how they are formulated using reinforcement learning ideas. We will also provide some reasons why we think implementing the predictive knowledge approach using GVFs is promising. Finally, we will review some of the previous work on GVFs.

3.1 From Value Functions to General Value Functions

As we said, knowledge can be well thought of as regularities in one's interaction with the world. To model this interaction, let's suppose we have an agent and environment interacting with each other at discrete time steps. At each time step t, the agent receives sensory signals from the environment and performs an action. The environment receives the action and responds to the agent with new sensory signals. We denote the observation at time t with $O_t \in \mathcal{O}$

and the action at time step t with $A_t \in \mathcal{A}$. The experience of the agent in its environment can be expressed as the sequence of the observations and actions. We call this sequence sensorimotor data:

$$O_0, A_0, O_1, A_1, O_2, A_2, ...$$

As the agent and environment interacts, certain regularities emerge in the sensorimotor data. The regularities in the sensorimotor data are of the same kind as the sensorimotor data itself; they are in terms of observations, actions, and time steps. As an example of a regularity, we can think of the sensorimotor data of a child lying on his back. The child is looking at the ceiling. He starts moving his arm randomly and notices changes in his visual input. Whenever he moves his arm in a specific manner, he sees his arm. We can think of this as a pattern in the child's sensorimotor data. Whenever certain stream of actions are performed, certain sensors receive certain values.

To express the regularities in the sensorimotor data, we can use ideas from reinforcement learning, in particular value functions. Value functions capture certain regularities in the interaction between the agent and environment. More specifically, there is an interaction loop between the agent and the environment very similar to the one described in the previous paragraph. At each time step, t, the agent is in a state $S_t \in \mathcal{S}$, performs an action $A_t \in \mathcal{A}$, and the environment responds to the agent by generating a reward signal $R_{t+1} \in \mathcal{R}$ and taking the agent to the next state $S_{t+1} \in \mathcal{S}$. This procedure generates the following stream of states, actions, and rewards:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$$

In reinforcement learning, we seek to learn a specific pattern in the stream of states, actions, and rewards. This pattern is called a value function. As we discussed in Chapter 2, a value function is the expected sum of discounted reward given our actions come from a policy π where $\pi : \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$.

$$\begin{aligned} v_\pi(s) &= E[G_t | S_t = s, A_{t:\infty} \sim \pi] \\ G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \end{aligned} \tag{3.1}$$

where we denote the value function and return with v_π and G_t respectively.

Value functions can be extended to a more general form such that they can capture a broader range of patterns in the data and be more expressive. One such general form has been proposed by Sutton et al. (2011). A value function in this extended form is not limited to the reward signal and is called a general value function (GVF). In the next section, we discuss GVFs in more detail.

3.2 General Value Function Formulation

General value functions are a language for expressing regularities in the data. Formulation of a GVF consists of specifying a question part and an answer part. The question part defines the target that we want to predict. We can think of the target as a summary of the future that we are interested in. The answer part is the process that estimates the target. In what follows, we discuss the question and answer part in more detail.

The question part defines the regularity in the data that we are interested in and we want to predict. The regularity is a scalar target similar to the case of conventional value functions. As we mentioned before, the regularity is in terms of observations, actions, and time steps. The question part has three main elements, each capturing one of these three aspects of a regularity: 1- Target policy 2- continuation function 3- cumulant.

The target policy is the policy that we want to know about. In other words, the target policy generates the sequence of actions that we want to ask a question about. We denote the target policy with $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$.

The continuation function specifies the time scale or horizon of the regularity. As we discussed earlier, the regularity is a scalar target that is a summary of the future; at each time step, we care about what happens in the future only till some specific point. The continuation function specifies that point. We denote the continuation function by $\gamma : \mathcal{S} \rightarrow [0, 1]$. The continuation function is similar to the discount factor in conventional reinforcement learning. The discount factor specifies how much weight an immediate reward gets compared to the one that happens in the far future. There is an interpretation of the discount factor that is similar to the role of a continuation function. The interpretation suggests that we can think of the discount factor as a probability of termination or continuing. At each time step, the probability of terminating is $1-\gamma$ and the probability of continuing is γ. If the task is episodic, the discount factor is one during an episode and becomes zero when we reach the end of the episode. This is called hard termination. If the task is continuing, the discount factor is constant and there is a constant probability of terminating at each time step. This is called soft termination. In this case the expected number of steps till termination will be $T = \frac{1}{1-\gamma}$. The same interpretation is true for the continuation function in GVF language. However, the continuation function is a generalized form of the discount factor that is a function of the state and can use the information in the state to specify the point of termination. Therefore, it adds to the expressiveness of the GVF language.

The cumulant is the information that is available at each time step to be accumulated to construct the target. The cumulant has a very similar role to the reward signal in conventional reinforcement learning. However, it has been introduced as a more general signal so that the learning system can make predictions about many signals in the world and not limited to the reward signal. These signals can be some sensory observations or any function about any signal in the world. The cumulant is a function of states and is denoted by $c : \mathcal{S} \rightarrow \mathcal{R}$.

These three functions – target policy, continuation function, and cumulant – define the target of a GVF. Similar to conventional RL, we will have a notion of return or outcome that is a weighted sum of the cumulant. However, this formulation of return uses a state dependent discounting $\gamma(S_t)$; therefore, the cumulants are weighted by products of $\gamma(S_t)$ instead of powers of γ.

$$\begin{aligned} G_t &= C_{t+1} + \gamma_{t+1} C_{t+2} + \gamma_{t+1}\gamma_{t+2} C_{t+3} + ... \\ &= \sum_{k=0}^{\infty} (\prod_{j=1}^{k} \gamma_{t+j}) C_{t+k+1} \end{aligned} \tag{3.2}$$

Where C_t and γ_t are $c(S_t)$ and $\gamma(S_t)$ respectively.

The value function is defined as the expectation of outcome or return, similar to conventional RL. We subscript $v(s)$ by the target policy, continuation function, and cumulant:

$$v_{\pi,\gamma,c}(s) = v(s; \pi, \gamma, c) = E[G_t | S_t = s, A_{t:\infty} \sim \pi]$$

The answer part of a GVF is the mechanism that is used to approximate the answer to the GVF. In order to approximate the answer, we have to make certain choices about the mechanism. First, we should specify what function approximator we want to use and set the parameters of the function approximator. For example, we might want to use linear function approximation with tile coded features. Second, we should decide which algorithm we want to use to estimate the answer to the GVF and specify the parameters of the algorithm such as step-size. Finally, we should choose the behavior policy that would generate the data.

3.3 Why GVFs are a Promising Approach for Representing Knowledge

Understanding how knowledge can be acquired, represented, and maintained is a big challenge. We believe GVFs provide us with a path to understand how we can approach this challenge. In this section, we discuss certain characteristics of GVFs that makes it a proper language for representing knowledge. First, GVF predictions are in terms of observations, actions, and time steps. Therefore, the GVF language produces pieces of knowledge that are grounded in experience.

Second, acquiring and maintaining GVF predictions does not rely on human intervention. The correctness of a prediction can be verified by what actually happens. Therefore, acquisition of predictive knowledge may be scalable with computation and data.

Third, a GVF prediction is specified by a target policy, continuation function, and cumulant function, all of which are functions of state. Therefore, a GVF prediction has access to any information that is included in the state.

This information could be low level sensorimotor information, or some functions of the sensorimotor data, or other GVF predictions. Therefore, GVFs have the potential to be an expressive language.

Fourth, the GVF language produces pieces of knowledge that have continuous values. Therefore, GVF predictions can easily be used by different procedures such as reasoning and planning.

Fifth, we need the same mechanism and algorithms for learning GVF predictions as we use for conventional value functions. Therefore, we can take advantage of the computational power of conventional reinforcement learning to learn GVF predictions efficiently and online.

3.4 Related Work

GVFs were first introduced as a shift from value functions to knowledge in Sutton et al. (2011). In that work, an architecture for acquiring knowledge was proposed, called Horde. The architecture consisted of many sub-agents each trying to answer a predictive question about the environment in the form of a GVF. Each sub-agent had its own target policy, but learned about the environment from experience generated by the same behaviour policy.

The power of GVFs in representing a psychological class of predictions called nexting was shown in Modayil et al. (2014). Nexting is defined as the tendency of people and many animals to continually make predictions about what will happen next. Modayil et al. (2014) showed that thousands of GVF predictions can be learned online, on a robot, and using conventional computational resources. While Modayil et al. focused on on-policy prediction learning, White (2015) further investigated the effectiveness of the GVF language providing experiments on both on-policy and off-policy prediction learning.

Ring (2014) provided a thought experiment showing how knowledge can be built layer-by-layer from low-level understanding of the sensorimotor data to high-level concepts, using GVFs. To construct knowledge, Ring proposed putting a set of GVFs into a layer-by-layer architecture where each layer was built upon lower-level layers

The GVF language is not the only predictive approach addressing the challenge of grounding knowledge in experience. Two other approaches are predictive state representations (PSR) and temporal-difference networks. PSRs represent the state of a system in terms of predictions about the stream of observations and actions (Littman et al., 2002). While the GVF predictions are the accumulative value of a cumulant signal, the PSR predictions are the probability of different sequences of observations happening.

Temporal-difference networks are another framework for representing and learning regularities in the interaction of an agent with its world (Sutton and Tanner, 2015; Tanner and Sutton, 2005; Rafols et al., 2006; Rafols, 2006; Sutton, 2009). These regularities are in the form of predictions about the consequences of performing an action and are compositional in the sense that each prediction uses the output from other predictions. In TD nets, similar to GVFs, we want to estimate a collection of scalar predictions. However, in TD nets these scalar predictions are arranged in an interrelated network. The network has some primitive nodes that are predictions about some observation signals; the non-primitive nodes are predictions about the value of other nodes given that some specific action is performed. Similar to GVFs, TD nets have a notion of a question and answer language that are represented as networks. The question network determines how the predictions are connected to each other; the answer network specifies how we should approximate the value of one prediction from our estimate of another prediction. There are two main differences between TD nets and GVFs. First, TD nets predictions are predictions about the value of a specific observation signal, given that a specific stream of actions is followed. However, GVF predictions are predictions about the accumulative value of a signal. Second, the state representation is more complicated in TD nets compared to GVFs.

3.5 Summary

In this chapter, we discussed general value functions. We reviewed how conventional value functions can be generalized to be more expressive and be able to capture a wider range of regularities in the sensorimotor data. We provided some reasons why GVFs are a proper language for acquiring and representing knowledge. The chapter closed with a review of the previous work on GVFs and some other related predictive approaches.

Chapter 4

The Robots Used in this Thesis

In this chapter, we introduce the robots that we used for our experiments. The main goal of this chapter is to take a close look at the sensors and actuators of the robots and give a sense of what the sensorimotor data looks like.

We used two robots for our experiments. One is a simple robot consisting of just two servo actuators. We call this robot the Dynamixel robot. The second robot is a more complicated mobile robot designed for research on robotics, called Kobuki. In what follows we provide details about the sensors and actuators of the robots and how the sensorimotor information is communicated between the robots and the computer on which the learning system runs.

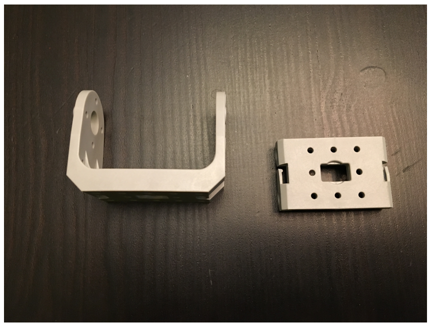
4.1 The Dynamixel Robot

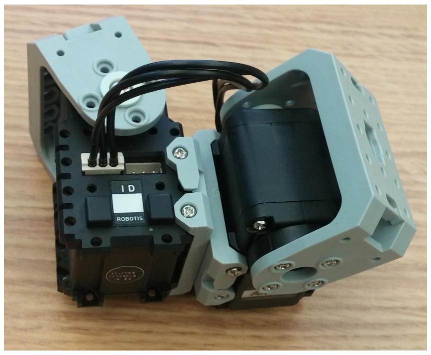
We made the Dynamixel robot with the purpose of performing empirical studies on robot prediction tasks. The robot consists of two Dynamixel AX-12 servo actuators, two OF-12S frames, and two OF-12SH frames. See Figure 4.1a and 4.1b. The servo actuators are connected to one another by OF-12S frame. In addition, an OF-12SH frame is installed on each of the AX-12 as shown in Figure 4.1c. [1]

The Dynamixel robot communicates with a computer on which the learning system runs, through a USB2Dynamixel and a connector that we made. The connector also connects the robot to the power supply. See Figure 4.2. The

[1]Figures 4.1a, 4.1b, 4.2a, and 4.2c were provided in CMPUT 607 (Applied Reinforcement Learning course), University of Alberta.

(a) Dynamixel AX-12 actuators


(b) OF-12SH and OF-12S frames are in left and right of the figure respectively

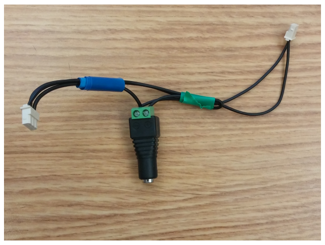


(c) The body of the robot

Figure 4.1: The components of the Dynamixel robot.

sensorimotor data gets transferred between the robot and the computer using a library created by Georgia Tech Research Corporation, called lib_robotis_hack.

(a) The USB2Dynamixel


(b) The connector

(c) The power supply

Figure 4.2: The USB2Dynamixel, connector, and power supply of the Dynamixel robot.

There is four sensory information regarding each servo actuator: angle, load, temperature, and voltage. We can also control the movements of the servo actuators by sending a command telling the actuator to move to a specific angle.

4.2 The Kobuki

The Kobuki is a robotic base designed for research on robotics. There are several manufacturers that make Kobukis; we got one Kobuki from Clearpath Robotics. The Kobuki is well suited for research because it can run for multiple hours, has reliable sensors and driving system, and is physically robust. For our experiments, we used a Kobuki as the base of the robot, added an Orbbec Astra Pro Sensor to it, and put a netbook on top of the base. See Figure 4.3.

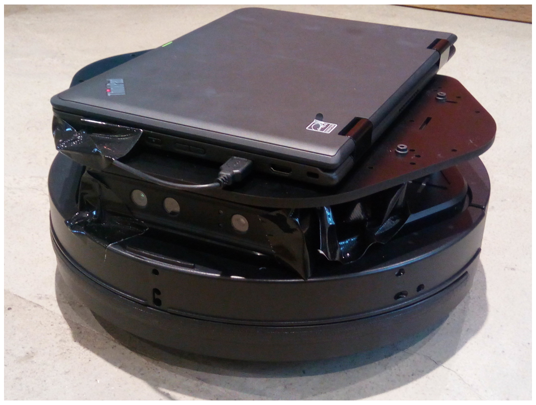


Figure 4.3: The Kobuki with the 3D camera and netbook.

The Kobuki has multiple sensors including bumpers detecting when the robot bumps into something, wheel drop sensors detecting drops, and cliff sensors that are infrared emitters detecting edges and stairs. The Kobuki is also equipped with a gyroscope providing information about the orientation, angular velocity, and angular acceleration. There are also some sensors providing information about the motors and battery such as motors' current, battery level, and charger state. We have also attached an Orbecc Astra 3D camera to the Kobuki. The resolution of the camera is 480×640.

Another main sensing ability of the Kobuki is provided by three infrared receivers detecting the position of the robot with respect to its charging station. The charging station has 3 infrared emitters each covering the left, center, and right regions in front of it. The regions are also divided to near and far subfields. The three infrared receivers of the robot are located in front of it and each receiver can detect in which region it is if it is facing toward the charging station. Each infrared receiver produces a unique reading that is based on the region in which it is.

For reading the sensory information and controlling the robot we use the Robot Operating System (ROS). ROS is an open-source collection of libraries for making robot software. The sensorimotor information gets communicated between the Kobuki and the netbook via a USB cable. The sensory data of most of the sensors are available at a rate of 50 Hz. The camera data are available at a rate of 30 Hz. The sensory information of each sensor gets stored in a queue and will be ready for the learning system to access at some specific timescale. For different experiments, we have set the timescale to different values. At each time step, the learning system has access to the data stored in all of the queues. It should summarize the data of each queue to one value so that in each time step it would have only one value for each sensor. There are multiple methods for summarizing the data of the queues. For most of the sensors, we use the most recent element of the queue. For both the bump sensor and infrared sensors, we use bitwise or; however for the former we apply bitwise or on all the elements of the bump queue and for the latter we apply it only to the last 10 elements of the infrared queue.

The driving system of the Kobuki allows it to go forward and backward and turn left and right. The Kobuki has two powered wheels and two non-powered wheels. To make the Kobuki move smoothly, we made a process that sends action commands to the Kobuki at a rate of 40 Hz. This process is called the action manager. The learning system determines the action commands that the action manager sends to the Kobuki. The rate of sending action commands from the learning system is determined by its timescale and is different from the rate of sending action commands from the action manager to the Kobuki. The action manager resends the previous action command to the Kobuki until it receives a new action command from the learning system.

To use ROS libraries, a process called ROS Master should run. ROS master manages the communication between different ROS processes. In our implementation, ROS master was run on the netbook. We also ran the learning system on the netbook by connecting to it via SSH from another computer.[2]

4.3 Summary

In this chapter, we discussed the robots that we used for our experiments. The first robot is a simple robot with a limited number of sensors. Although this robot is simple, performing empirical studies on it is a good first step for studying the language of GVFs for representing predictive knowledge. This is mainly because it is easier to understand the data stream and there are fewer factors that affect the learning process which we do not have control over. In Chapter 6, we present a case study involving the Dynamixel robot. While the first robot consists of just two actuators, the second robot is much more complicated and consists of a Kobuki, a 3D camera, and a netbook. We present the case studies that use the Kobuki in Chapter 7 and 8.

[2]To do experiments on the Kobuki, we used software provided as a result of the Robot Learning Project initiated in the University of Alberta: https://amiithinks.github.io/RobotLearning/

Chapter 5

The Off-policy Algorithms Used in this Thesis

In this chapter, we discuss the reinforcement learning (RL) algorithms that we used for our case studies. The algorithms that we considered belong to the family of temporal-difference (TD) learning methods. We use them to learn off-policy. These considered 10 prediction algorithms and 3 control algorithms, all with per step computation linear in the number of function approximation parameters. In what follows, first we discuss the prediction algorithms then we talk about the three algorithms that we used for solving our control problem.[1]

5.1 The Algorithms for Solving the Policy Evaluation Problem

In this section, we discuss the algorithms that we used in our case studies for estimating the value function for a given policy. The first algorithm is off-policy TD(λ). A description of TD(λ) can be found in Chapter 2. As we discussed in Chapter 2, off-policy TD(λ) has been shown to be unstable in conjunction with linear function approximation.

We considered several algorithms from the gradient-TD family of methods. Gradient-TD methods are one of the most important family of methods with nice convergence guarantees under off-policy training. The first algorithm that we considered from this family was GTD(λ); this algorithms applies stochastic

[1]We would like to thank Sina Ghiassian for useful discussions on different algorithms and in particular for providing the variant of ABQ(ζ) algorithm for learning state values.

gradient descent (SGD) to an objective function called mean squared projected bellman error (MSPBE) and take advantage of the robust convergence properties of SGD (Sutton et al., 2009; Maei and Sutton, 2010; Maei, 2011). GTD(λ) can be specified by equations 5.1. This algorithm has two weight vectors, $\mathbf{w}$ and $\mathbf{h}$, and two step-sizes, α and α_h. The update to the main weight vector, $\mathbf{w}$, for GTD(λ) is similar to the one for TD(λ). However, there is an additional term in the update of GTD(λ). This additional term, $\gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_t)\mathbf{x}_{t+1}$, is called the gradient correction. This term is added to correct the TD update so that the update follows the gradient of MSPBE.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[\delta_t\mathbf{z}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_t)\mathbf{x}_{t+1}] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t]
\end{aligned} \tag{5.1}$$

We also considered GTD2(λ) from the gradient-TD family of methods. GTD2(λ), similar to GTD(λ), applies stochastic gradient descent to MSPBE and is convergent under off-policy training. The main difference between GTD2(λ) and GTD(λ) algorithms is that they use different techniques for computing an estimate of the gradient of MSPBE. The GTD2(λ) algorithm can be specified by equations 5.2. Similar to GTD(λ), GTD2(λ) has two weight vectors and two step-sizes.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[(\mathbf{h}_t^T\mathbf{x}_t)\mathbf{x}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_t)\mathbf{x}_{t+1}] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t]
\end{aligned} \tag{5.2}$$

Another gradient-TD algorithm that we considered is hybrid TD or HTD(λ) that can be thought of as a combination of GTD(λ) and TD(λ) methods (Hackman, 2012; White and White, 2016). HTD(λ) applies updates similar to GTD(λ) when the target and behavior policies are different and behaves similarly to TD(λ) otherwise. This algorithm takes advantage of both GTD(λ) robustness and TD(λ) sample efficiency. Equations 5.3 summarizes the HTD(λ) algorithm. Similar to GTD(λ), HTD(λ) has two weight vectors. HTD(λ) has an extra eligibility trace vector, $\mathbf{z}_t^\mu$, which is an on-policy trace.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{z}_t^\mu &= \gamma_t\lambda\mathbf{z}_{t-1}^\mu + \mathbf{x}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[\delta_t\mathbf{z}_t - (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(\mathbf{z}_t - \mathbf{z}_t^\mu)^T h_t] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t - \gamma_{t+1}\mathbf{x}_{t+1})(h_t^T\mathbf{z}_t^\mu)]
\end{aligned} \tag{5.3}$$

We also used two other gradient-TD methods in our case studies that use proximal methods to achieve acceleration. The first method is proximal GTD(λ) (Mahadevan et al., 2014; Liu et al., 2015) and can be specified by equations 5.4. The equations for proximal GTD(λ) are similar to those of GTD(λ); however, the proximal one has an additional half step update.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{h}_{t+\frac{1}{2}} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t] \\
\mathbf{w}_{t+\frac{1}{2}} &= \mathbf{w}_t + \alpha[\delta_t\mathbf{z}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_t)\mathbf{x}_{t+1}] \\
\delta_{t+\frac{1}{2}} &= R_{t+1} + \gamma_{t+1}\mathbf{w}_{t+\frac{1}{2}}^T\mathbf{x}_{t+1} - \mathbf{w}_{t+\frac{1}{2}}^T\mathbf{x}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[\delta_{t+\frac{1}{2}}\mathbf{z}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_{t+\frac{1}{2}})\mathbf{x}_{t+1}] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_{t+\frac{1}{2}}\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_{t+\frac{1}{2}})\mathbf{x}_t]
\end{aligned} \tag{5.4}$$

The second proximal method that we considered is proximal GTD2(λ). The equations for proximal GTD2(λ) are provided below. The updates to the weight vectors for proximal GTD2(λ) are similar to those of GTD2(λ). However, proximal GTD2(λ), similar to GTD(λ), had an additional half step update.

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{h}_{t+\frac{1}{2}} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t] \\
\mathbf{w}_{t+\frac{1}{2}} &= \mathbf{w}_t + \alpha[(\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_t)\mathbf{x}_{t+1}] \\
\delta_{t+\frac{1}{2}} &= R_{t+1} + \gamma_{t+1}\mathbf{w}_{t+\frac{1}{2}}^T\mathbf{x}_{t+1} - \mathbf{w}_{t+\frac{1}{2}}^T\mathbf{x}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[(\mathbf{h}_{t+\frac{1}{2}}^T\mathbf{x}_t)\mathbf{x}_t - \gamma_{t+1}(1-\lambda)(\mathbf{z}_t^T\mathbf{h}_{t+\frac{1}{2}})\mathbf{x}_{t+1}] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_{t+\frac{1}{2}}\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_{t+\frac{1}{2}})\mathbf{x}_t]
\end{aligned}
\tag{5.5}
$$

Another important method with appealing convergence properties under off-policy training that we used is emphatic-TD(λ) or ETD(λ) (Sutton et al., 2016). This method achieves stability by ensuring that the distribution of the updates are on-policy meaning that all state transitions are based on the target policy. ETD(λ) can be specified by equations 5.6. ETD(λ) has an extra trace, F, which changes the distribution of the updates by affecting the eligibility traces. F is a scalar and is called the followon trace.

$$
\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
F_t &= \rho_{t-1}\gamma_t F_{t-1} + I_t, \quad \text{with } F_{-1} = 0 \\
M_t &= \lambda I_t + (1-\lambda)F_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + M_t\mathbf{x}_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t
\end{aligned}
\tag{5.6}
$$

We also considered a generalized form of ETD(λ) introduced by Hallak et al. (2016), denoted by ETD(λ, β). This algorithm introduces an additional parameter, β, working as a bias-variance knob. Adding β was proposed to control the high variance of conventional ETD(λ) by adding bias to it. The equations of ETD(λ, β) are provided below. The difference between ETD(λ) and ETD(λ, β) is in the update of the followon trace. ETD(λ, β) uses β instead of γ when updating the followon trace. The original ETD(λ, β) algorithm was proposed for the case of constant discount factor. In the problems that we considered the discount factor was state dependent and either had a non-zero value less than 1 or was 0. To use the ETD(λ, β) algorithm, we used the same equations but whenever the discount factor was zero, the followon trace, F_t, became equal to I_t.

$$\begin{aligned}\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\ F_t &= \rho_{t-1}\beta F_{t-1} + I_t, \quad \text{with } F_{-1} = 0 \\ M_t &= \lambda I_t + (1-\lambda)F_t \\ \mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + M_t\mathbf{x}_t) \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t\end{aligned} \tag{5.7}$$

Off-policy learning suffers from high variance introduced by the importance sampling ratio. There exists methods that control the effect of importance sampling ratio on updates, not allowing high values of it to affect the trace. One such algorithm for learning action values is ABQ(ζ) introduced by Mahmood et al. (2017) with ζ being a trace parameter similar to λ. This algorithm also uses gradient-based TD updates, achieving stability under off-policy training. In this work, we used a variant of ABQ(ζ) that is used for learning state values, called ABTD(ζ) (personal communication with Sina Ghiassian). For our case studies, we implemented a simpler version of ABTD(ζ) that did not involve gradient corrections. The simpler version of ABTD(ζ) that we used can be summarized by equations 5.6. These equations are similar to the equations of off-policy TD. However, instead of $\lambda\rho$ in the update of the eligibility traces, ABTD(ζ) uses a new term $\nu_\zeta\pi$. This new term controls the variance of the updates.

$$\begin{aligned}\delta_t &= \rho_t[R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t] \\ \mathbf{z}_t &= \gamma_t\nu_{\zeta,t-1}\pi_{t-1}\mathbf{z}_{t-1} + \mathbf{x}_t \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t\end{aligned} \tag{5.8}$$

We also considered one algorithm that has convergence guarantees only in the tabular case, called provisional TD or PTD. This algorithm is interesting because it is the first off-policy algorithm that is equivalent to Monte Carlo algorithms when $\lambda = 1$ (Sutton et al., 2014). The equations for PTD(λ) are provided below. In the case of on-policy training, PTD(λ) is exactly the same as TD(λ). However, in the off-policy case, an additional weight vector, $\mathbf{h}$, affects the updates to the weight vector $\mathbf{w}$.

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t \\
\mathbf{z}_t &= \rho_t(\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t) \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t + (\rho_t - 1)\mathbf{h}_t \\
\mathbf{h}_{t+1} &= \gamma_{t+1}\lambda(\rho_t\mathbf{h}_t + \alpha\bar{\delta}_t\mathbf{z}_t) \\
\bar{\delta}_t &= R_{t+1} + \mathbf{w}_t^T\mathbf{x}_{t+1} - \mathbf{w}_t^T\mathbf{x}_t
\end{aligned} \tag{5.9}$$

5.2 The Algorithms for Solving the Control Problem

We considered three control algorithms in our work. The first one is the well-known Q-learning algorithm which we discussed in Chapter 2. As we discussed in Chapter 2, Q-learning is potentially divergent in the case of function approximation.

The second algorithm that we considered belongs to the family of gradient-TD algorithms. In the previous section, we discussed some of the algorithms in this family such as GTD(λ). Maei and Sutton (2010) introduced an extension of GTD(λ) for estimating action-value functions called GQ(λ). To extend GQ(λ) to control domains, Maei et al. (2010) presented an algorithm called greedy GQ(λ). The extension of GQ(λ) to greedy GQ(λ) is similar to the extension of TD(0) to Q-learning. Greedy GQ(λ) is the second algorithm that we considered in our studies. This algorithm can be specified by the following equations:

$$\begin{aligned}
\delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\hat{\mathbf{x}}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\
\mathbf{z}_t &= \rho_t\gamma_t\lambda\mathbf{z}_{t-1} + \mathbf{x}_t \\
\mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha[\delta_t\mathbf{z}_t - \gamma_{t+1}(1-\lambda)(\mathbf{x}_t^T\mathbf{h}_t)\hat{\mathbf{x}}_{t+1}] \\
\mathbf{h}_{t+1} &= \mathbf{h}_t + \alpha_h[\delta_t\mathbf{z}_t - (\mathbf{x}_t^T\mathbf{h}_t)\mathbf{x}_t]
\end{aligned} \tag{5.10}$$

where $\hat{\mathbf{x}}(S_{t+1}) = \mathbf{x}(S_{t+1}, A^*_{t+1})$ and $A^*_{t+1} = \text{argmax}_a \mathbf{w}_t^T \mathbf{x}(S_{t+1}, a)$. If $A_t = A^*_t$, $\rho_t = \frac{1}{b(A^*_t|S_t)}$; otherwise $\rho_t = 0$.

The third off-policy control algorithm that we considered is a variant of ABQ(ζ) for policy learning which we call greedy ABQ(ζ). We derived this algorithm from ABQ using the same greedifying method that was used for extending GQ(λ) to greedy GQ(λ). The greedy ABQ algorithm that we considered did not apply gradient corrections and can be specified by the following equations:

$$\begin{aligned} \delta_t &= R_{t+1} + \gamma_{t+1}\mathbf{w}_t^T\hat{\mathbf{x}}(S_{t+1}) - \mathbf{w}_t^T\mathbf{x}(S_t) \\ \mathbf{z}_t &= \gamma_t \nu_{\zeta,t}\pi_t \mathbf{z}_{t-1} + \mathbf{x}_t \\ \mathbf{w}_{t+1} &= \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \end{aligned} \tag{5.11}$$

where $\hat{\mathbf{x}}(S_{t+1}) = \mathbf{x}(S_{t+1}, A^*_{t+1})$ and $A^*_{t+1} = \text{argmax}_a \mathbf{w}_t^T \mathbf{x}(S_{t+1}, a)$. If $A_t = A^*_t$, $\pi_t = 1$; otherwise $\pi_t = 0$.

5.3 Summary

In this chapter, we reviewed the RL algorithms that we considered in our case studies. First, we discussed the 10 off-policy algorithms that we used for solving the policy evaluation problem. Then, we briefly reviewed the 3 off-policy algorithms that we used for solving the control problem. Reviewing our robots in the previous chapter and discussing our algorithms in this one, we are ready to present our first case study in the next chapter.

Chapter 6

The Dynamixel Case Study

Having introduced our robots and algorithms, in this chapter we present the first of our case studies that make up the primary contributions of this thesis. In this first case study, we perform an empirical assessment of many algorithms on a relatively simple robot prediction task. We decided to start with a simple task because interpretation of the results is more straightforward in this case. In the subsequent chapters, we present studies on more complicated tasks.

As we discussed in Chapter 1, we pursue the predictive knowledge approach using reinforcement learning (RL) ideas such as general value functions (GVFs) to learn predictive knowledge. In this thesis, we study how effective RL algorithms are at learning GVF predictions. This helps us in the bigger project of investigating the utility of GVFs as a language for representing knowledge. In the past couple of years many RL algorithms have been proposed; however, there are not any studies comparing these algorithms on real-world systems. We aim to provide a better understanding of RL algorithms by studying their application on robot data.

6.1 The Dynamixel Prediction Task

In this case study, we use the Dynamixel robot, which was described in Chapter 4, learning a prediction task. The Dynamixel prediction task is 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. See Figure 6.1. The limiting angles have the values of 0 and 1.5 radians and at each time step, the motor moves for 0.05 radians. It takes about 60 steps for the motor to go from one end to the other and come back.

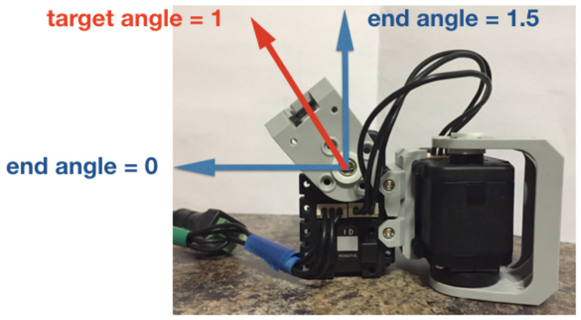


Figure 6.1: The Dynamixel robot

6.2 The GVF Formulation of the Dynamixel Prediction Task

We formulate the Dynamixel prediction task as a single GVF. As we discussed in Chapter 3, to formulate a GVF, we use the language of questions and answers.

For the question part, as we discussed in Chapter 3, we have to specify the target policy, the continuation function, and the cumulant function. For this prediction task, the target policy is to move back and forth between two limiting angles of 0 and 1.5. The continuation function always returns 0.9 except for the time when the distance between the current angle and the target angle is less than 0.05 in which case it returns 0. The cumulant function returns 1 when the distance between the current angle and the target angle is less that 0.05; otherwise, it returns 0.

The answer part is the mechanism that approximates the answer to the question and depends on the state representation, the learning algorithm, and the behavior policy. In this study, our answer part is a linear function with a tile coded representation. A description of tile coding and the learning algorithms is given in Chapter 2 and 5 respectively. We provide the details about the parameters that we used for the tile coding and learning algorithms, together with a description of the behavior policy in the next section.

6.3 The Dynamixel Prediction Experiment

To construct the feature vector, we used two signals from the motor: angle and velocity. Angle is a continuous value that can be between -0.1 to 1.6. We computed the velocity from the difference between the current angle and the previous angle. It is a discrete number that has a value of 1 whenever the difference is positive and has a value of -1 otherwise. To produce the features, the angle position was tile coded using 8 tilings each with 4 tiles. Tile coding the position resulted in a binary vector of size 8×4. The final feature vector was of size $2 \times 8 \times 4$ where each of the 2 parts corresponded to one of the values that velocity could have.

We applied the 10 algorithms discussed in Section 5.1 to this prediction task. For each algorithm, we made many instances of it with different values of the parameters. The step-size was in the form $\frac{\alpha}{\text{number of tilings}}$ where $\alpha \in \{0.1 \times 2^x | x = -13, -12, ..., 3\}$. For the gradient based algorithms the second step-size was in the following form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{2^x | x = -13, -12, ..., 2\}$. The ζ parameter for the ABTD algorithm and the trace parameter λ for all the other algorithms was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1\}$. The β parameter for ETD(λ, β) was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 1\}$

We used a behavior policy that was the same as the target policy plus ten percent randomness. Ten percent of the time, the motor would move in a direction opposite to the direction that it was supposed to go. The importance sampling ratio would have values of 0 and 1.11. We generated 30 runs of sensorimotor data. Each run consisted of 20000 steps and took approximately 100 minutes. Learning happened offline.

6.4 Performance Measure

Evaluating the learning algorithms on a robot can be challenging. In this robot domain, like all other robot domains, we do not have access to the value function. Therefore, we cannot evaluate the algorithms by looking at the difference between the learned estimations and the values. One natural choice for evaluating the algorithms in such settings is to calculate the return from the data and compare the predictions with the calculated returns. However, in this experiment, the algorithms learn the task off-policy and the data is generated following the behavior policy. Therefore, we have to correct the return calculated from the data using the importance sampling ratio. See equation 6.1. Unfortunately, the importance sampling ratio introduces variance to the return and the performance measure. Therefore, evaluating the learning process using the data produced by the behavior policy is not desirable.

$$G_t = \rho_t C_{t+1} + \rho_t \rho_{t+1} \gamma_{t+1} C_{t+2} + \rho_t \rho_{t+1} \rho_{t+2} \gamma_{t+1} \gamma_{t+2} C_{t+3} + ... \tag{6.1}$$

To deal with these difficulties of evaluating a learning process on a robot domain, we collected a number of sample states following the behavior policy and calculated their return following the target policy. To collect samples from the behavior policy, we followed the behavior policy for a random number of time steps coming from the uniform distribution of $U(70, 140)$. After sampling a specific time step, we followed the target policy for 70 time steps to compute the return corresponding to that time step. The reason that we used number 70 is that it takes about 60 steps for the motor to complete a cycle following the target policy; therefore, 70 seemed to be a big enough number to be used for this prediction task. Following this procedure multiple times, we collected 40 sample states and their corresponding return. During the learning time, we used these samples to compute a measure of how well the algorithms have performed. To evaluate the algorithms, we computed the following expression and got its root:

$$\widehat{\text{MSRE}}(\mathbf{w}) \doteq \frac{1}{|D|} \sum_{(s,G)\in D} (\mathbf{w}^T \mathbf{x}(s) - G)^2$$

where D includes the sample states and their corresponding returns and $\mathbf{w}$ is the weight vector. We denote this measure by $\widehat{\text{MSRE}}(\mathbf{w})$ because it can be thought of as an estimation of the mean square return error (MSRE):

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

where $d_b(s)$ denotes the probability of state s under the behavior policy.

6.5 Results

The learning curves of all the algorithms for the case of $\lambda = 0$ is shown in Figure 6.2. We decided to first show the results for the case with full bootstrapping because in this case the difference between the algorithms was more evident and as we increased the value of the trace parameter, the algorithms performed more similarly. All the other parameters were set to values resulting in the lowest asymptotic performance. To estimate the asymptotic performance, we computed the average of error over the last 0.0025 percent of each run and averaged over 30 runs.

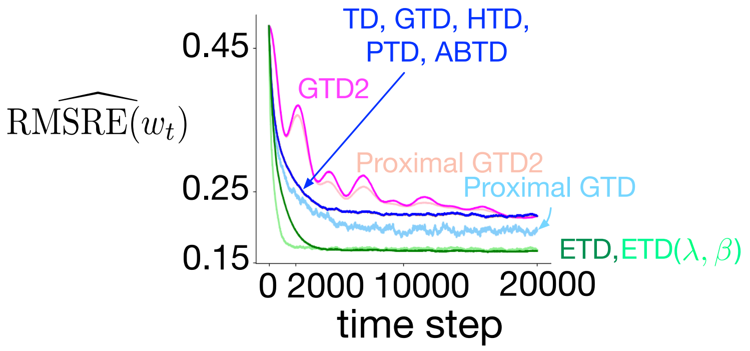


Figure 6.2: Learning curves for the case of $\lambda = 0$ for the Dynamixel prediction task. All the parameters are set to values resulting in the best asymptotic performance.

According to the learning curves, ETD(λ) and ETD(λ, β) reached a significantly lower level of asymptotic error compared to the other algorithms. Moreover, they converged faster than the other algorithms with ETD(λ, β) being the fastest (See Figure 6.2).

The parameter studies of the asymptotic performance of the algorithms over the step-size for the case of $\lambda = 0$ are shown in Figure 6.3. Based on the parameter studies, the two emphatic-TD algorithms reached the lowest

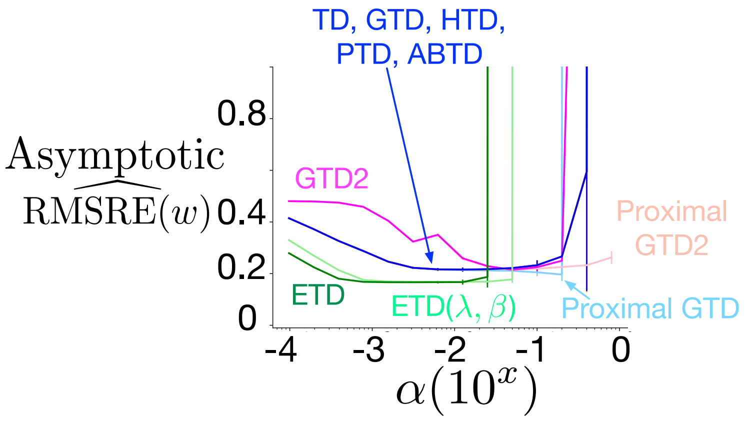


Figure 6.3: Parameter studies of the asymptotic performance over the step-size for the case of $\lambda = 0$ for the Dynamixel prediction task. All the parameters are set to values resulting in the best asymptotic performance.

level of asymptotic error; however, they converged for a smaller range of the step-size compared to the other algorithms.

The learning curves of the algorithms for the case with $\lambda = 0.95$ is shown in Figure 6.4. Based on this figure, the algorithms performed similarly for $\lambda = 0.95$ with PTD(λ) being slower compared to the other algorithms.

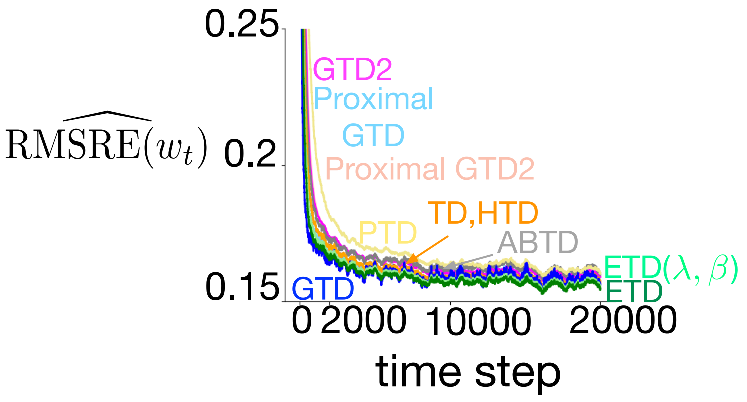


Figure 6.4: Learning curves for the case of $\lambda = 0.95$ for the Dynamixel prediction task. All the parameters are set to values resulting in the best asymptotic performance.

An interesting observation is that ETD(λ) and ETD(λ, β) in the case of full bootstrapping are as good as all the other algorithms with $\lambda = 0.95$. See Figure 6.2 and Figure 6.4. In other words, using higher values of λ improves the performance of all the other algorithms except for ETD(λ) and ETD(λ, β).

The parameter studies of the algorithms for the case with $\lambda = 0.95$ is shown in Figure 6.5. The parameter studies support the result of the learning curves that all the algorithms perform similarly for $\lambda = 0.95$.

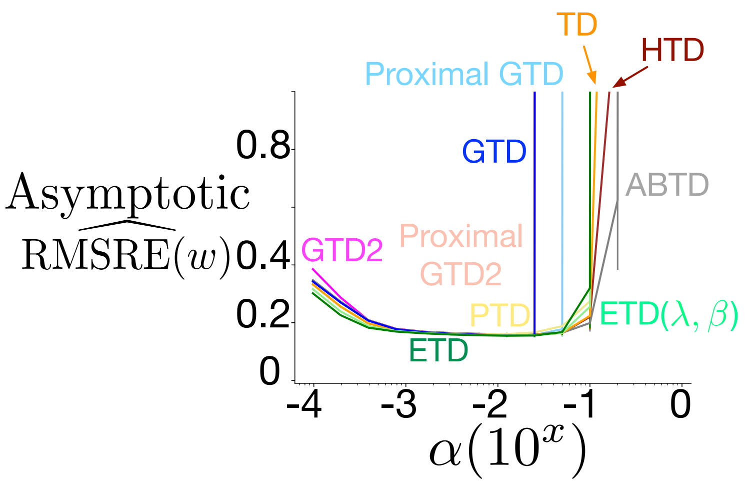


Figure 6.5: Parameter studies of the asymptotic performance over the step-size parameter for the case of $\lambda = 0.95$ for the Dynamixel prediction task. All the parameters are set to values resulting in the best asymptotic performance.

In addition to the results regarding the comparison between the algorithms, there were some interesting observations in the parameter studies of each of them. To understand these parameter studies, lets look at the plot for TD(λ) in Figure 6.6 as an example. This plot shows the effect of step-size on the asymptotic performance for different values of λ. Each line in this plot corresponds to one value of λ. All of the lines have a U-shape, with the ones corresponding to the higher values of λ having smaller asymptotic error. The lines corresponding to the lower values of λ, on the other hand, converge for a wider range of step-sizes.

Discussing an example of a parameter study in the previous paragraph, we are ready to discuss some interesting observations about the parameter sensitivity of the algorithms. First, ETD(λ), unlike the other algorithms, was not sensitive to the value of λ (See Figure 6.6). Second, GTD(λ), proximal GTD(λ), and HTD(λ) were not much sensitive to the value of the second step-size and their sensitivity reduced as the trace parameter got bigger (See Figure

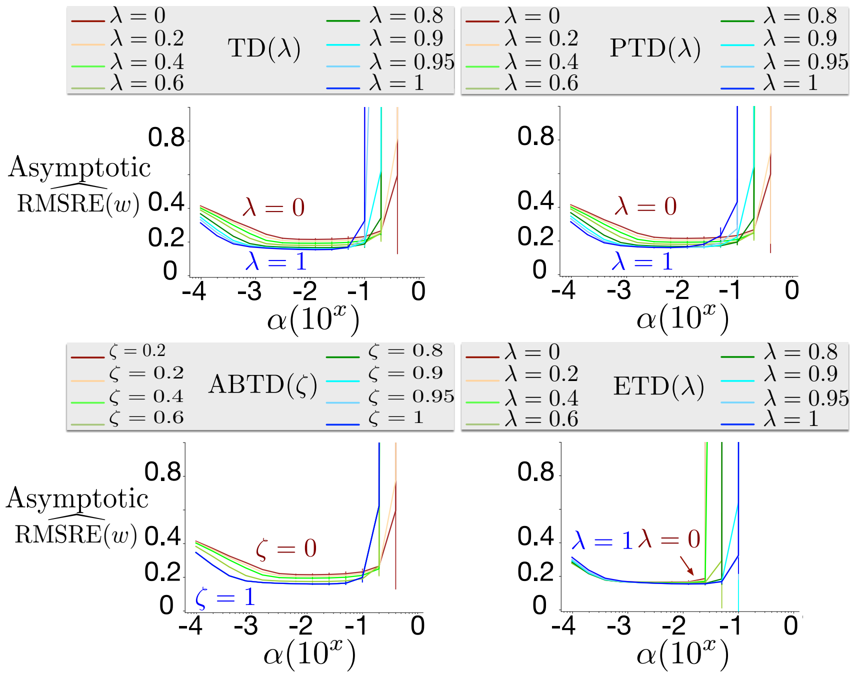


Figure 6.6: The parameter studies over the step-size parameter for ABTD(ζ), TD(λ), PTD(λ), and ETD(λ) for the Dynamixel prediction task.

6.7). Third, the asymptotic performance of TD(λ), PTD(λ), and ABTD(ζ) were very similar (See Figure 6.6). Fourth, for ETD(λ, β) as β got bigger, the range of step-size for which the algorithm converged got smaller. Moreover, for higher values of λ, β did not much affect the asymptotic performance.

6.6 Summary and Conclusions

In this chapter, we presented our first case study. We performed an empirical comparison of 10 off-policy temporal-difference learning algorithms on a simple prediction robot task. Our results suggest that several RL algorithms seem useful for learning predictions represented as GVFs, within the computational and sample complexity constraints of a simple robot. Based on our empirical comparison, ETD(λ) and ETD(λ, β) outperform the other algorithms both in terms of asymptotic performance and speed in the Dynamixel experiment. However, they converge for a shorter range of step-sizes compared to the other algorithms.

$$\eta = \frac{\alpha_w}{\alpha}$$

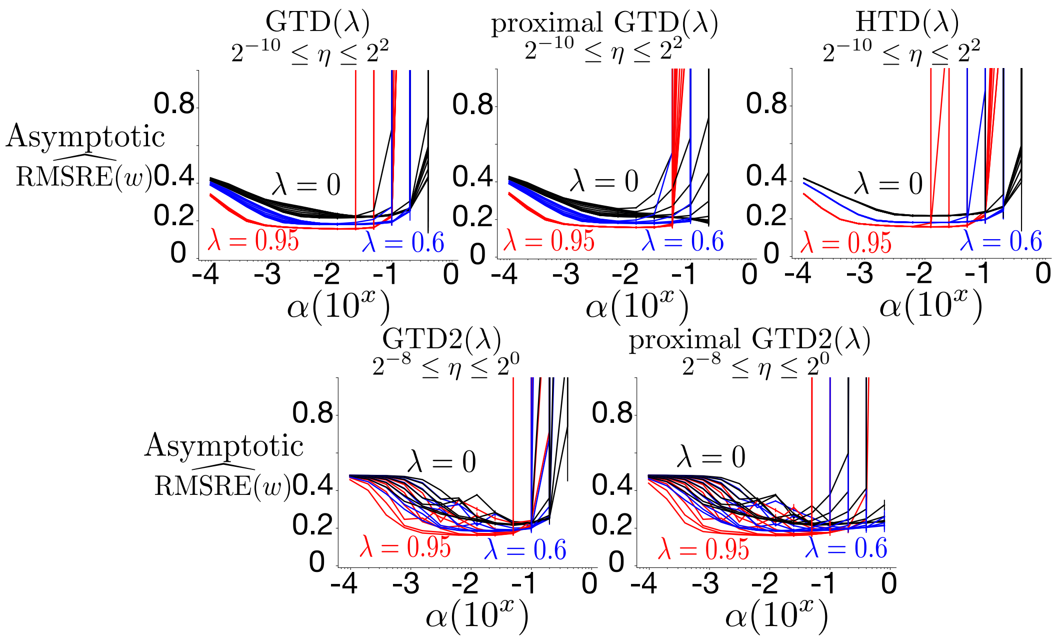


Figure 6.7: The parameter studies over the step-size parameter for gradient based algorithms for the Dynamixel prediction task. Each curve of each plot is for a specific value of λ and η, where all the curves corresponding to $\lambda = 0$, $\lambda = 0.6$, and $\lambda = 0.95$ are shown in red, blue, and black respectively.

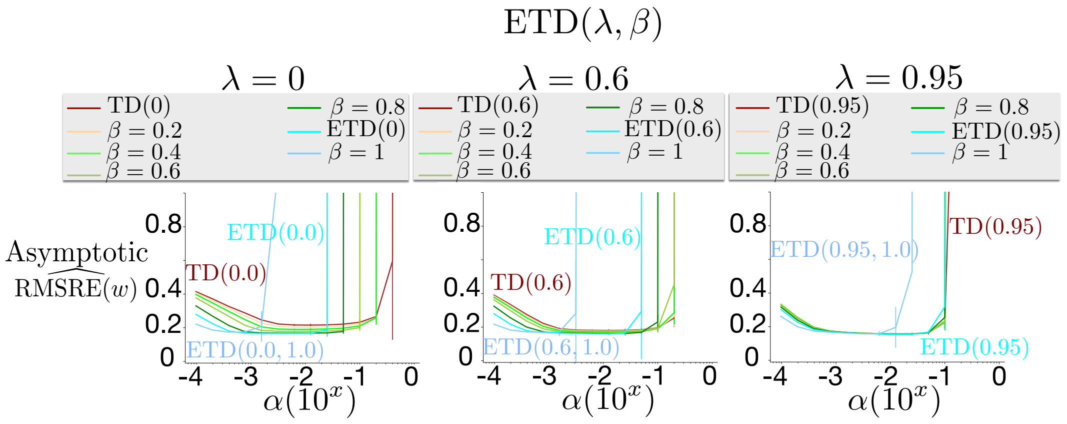


Figure 6.8: The parameter studies over the step-size parameter for ETD(λ, β) for the Dynamixel prediction task. Each plot corresponds to one value of λ and contains multiple lines each corresponding to different values of β, where $\beta = 0$ and $\beta = 0.97$ are equivalent to TD(λ) and ETD(λ) respectively.

We can also conclude from this case study that the methodology that we developed for doing empirical comparisons on a simple robot domain seems to be effective. Using this methodology, we could do parameter studies of many algorithms on robot domains and for many runs.

Chapter 7

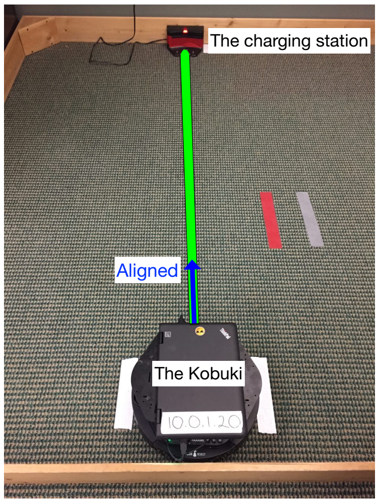
The Time-to-align Case Study

Having presented our first case study on a simple robot prediction task, in this chapter we present the second of our three case studies on a more complicated robot, the Kobuki. In this case study, similar to the previous one, we perform an empirical comparison of several algorithms. However, instead of focusing on a robot prediction task, we consider a robot control task.

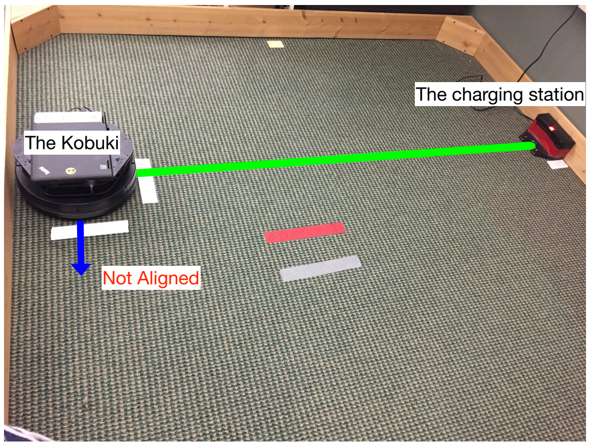
As we discussed in Chapter 6, investigating how well RL algorithms learn GVF predictions can benefit us in the project of examining the suitability of GVFs as a language for representing predictive knowledge. Reinforcement learning is enriched with control algorithms; however, there is not any study comparing these algorithms on real-world control tasks. In this chapter, we perform an empirical comparison of several RL algorithms learning a goal-directed GVF.

7.1 The Time-to-align Control Task

For this case study, we use the Kobuki which was described in Chapter 4 and use it to learn a control task. The control task is to learn to align with the charging station as fast as possible. To do this task the Kobuki can turn left and right in place. See Figure 7.1. We call this task the time-to-align control task. There are two sensors available for the robot to learn this control task. One is the infrared receiver at the front of the Kobuki that produces a unique reading whenever the Kobuki is aligned with the charging station. The other is the Kobuki's gyroscope that provides the orientation of the robot; turning left increases the orientation. It takes about 100 steps for the robot to complete a cycle with the velocity at which it is turning.



(a) The Kobuki is aligned with the charging station



(b) The Kobuki is NOT aligned with the charging station

Figure 7.1: The Kobuki alignment with the charging station.

7.2 The GVF Formulation of the Time-to-align Control Task

We formulate this control task as a single GVF. As we discussed in Chapter 3, formulating a GVF is a process of specifying a question part and an answer part.

For the question part of this GVF, we use a target policy that is greedy with respect to the action-value function where the action set consists of turning left and turning right. The continuation function returns 0 whenever the Kobuki is aligned with the charging station, otherwise, it returns 1. Finally, the cumulant function always returns -1.

The answer part of this GVF is a linear function with tile coded representation. The things that influence the answer part are the state representation, learning algorithm, and behavior policy. We provide more details about how we set these things for learning this GVF in the next section.

7.3 The Time-to-align Control Experiment

To construct the feature vector, we used the tile coding software.[1] The input to tile coding was the orientation of the Kobuki which was a real value between -1 and 1 that wraps around 1. We fed the orientation to a wrap tile-coder with 16 tilings each with 4 tiles. The tile coding software uses an indexed hash table; for this experiment we set the size of the hash table to 4096.

We applied the 3 algorithms of Q-learning, greedy GQ(λ), and greedy ABQ(ζ) which we discussed in Chapter 5 to this control task. We made several instances of each algorithm each corresponding to a parameter setting. The step-size was in the form $\frac{\alpha}{\text{number of tilings}}$ where $\alpha \in \{0.1 \times 2^x | x = -4, -3, ..., 1\}$. λ and ζ parameters were a number in $\{0, 0.5, 0.9\}$ for the greedy GQ and ABQ algorithms. The second step-size was in the following form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{0.0625, 0.25, 1, 4\}$.

The behavior policy selected between turning left and right randomly at each time step with a 90 percent bias toward repeating the previous action. The time scale for this experiment was 0.1 seconds.

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 a 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 each batch. For the sliding problem, we moved the robot manually to the starting position. For 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 could make 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 following

[1]Here is a link to the tile coding software: http://incompleteideas.net/tiles/tiles3.html

the behavior policy. Collecting each batch took about 100 seconds; however because of manually readjusting the robot, the whole process of collecting the 80 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.

7.4 Performance Measure

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. For this case study, however, doing the evaluation on the robot could be expensive because we needed to run lots of tests. A large number of tests were needed because we had many instances of each of the three algorithms, each corresponding to a parameter setting. For example, GQ had 72 different parameter settings. In addition to the high number of instances for each of the algorithms, we had 4 different runs of learning data. Moreover, we wanted to do multiple tests each starting from a different starting orientation. Finally, in order to get an estimation of how good each algorithm had learned over time, we had to evaluate the policy that they had learned after different number of time steps. To get an estimation of how many tests are needed to evaluate 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. 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 of Figure 7.2. The x axis of the plots is the orientation and the y axis is the action-value function for action a. Each plot contains two lines with the red one corresponding to the turning right action and the blue one corresponding to the turning left action. 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 range $[0, 1]$, it should turn right, that is 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.

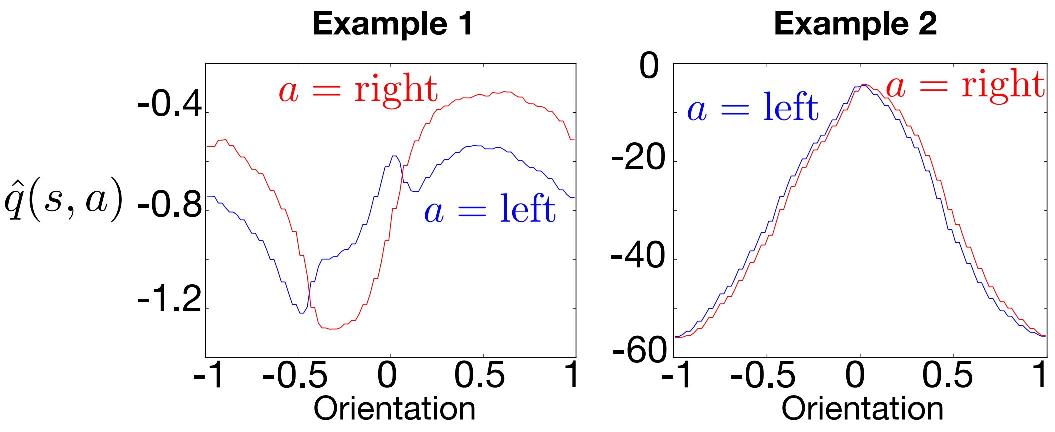


Figure 7.2: Examples of action-value plots.

We selected a reasonable parameter setting for each of the algorithms by looking at 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, 4×0.05, 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 algorithms over time, we considered the greedy policy with respect to the learned action-value functions after different number of time steps. We call these time steps, evaluation points. For this experiment the evaluation points were 1000, 5000, 10000, 20000 times steps. 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 7.3. 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.

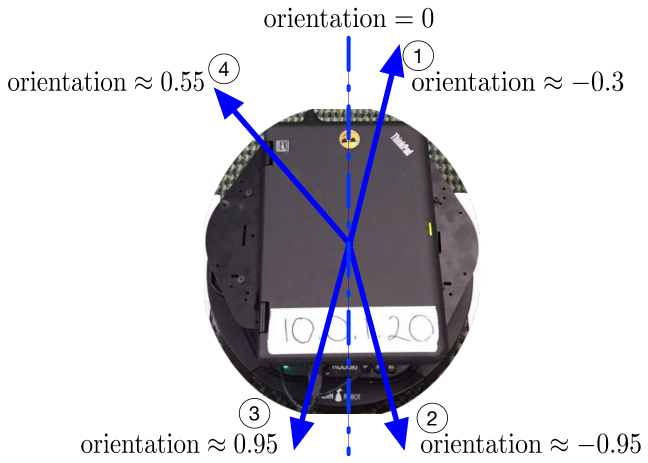


Figure 7.3: The four different starting orientations for the evaluation.

7.5 Results

The performance of the three algorithms of Q-learning, greedy GQ, and greedy ABQ is shown in Figure 7.4. The x axis is the evaluation point. The y axis is the average return over runs and tests where the return is the number of steps to alignment negated.

According to Figure 7.4, all of the algorithms learned to turn in the direction that would achieve the goal fastest within $20,000$ time steps. Greedy GQ(λ) outperformed the other algorithms in the mean; however, further work is required to establish significance.

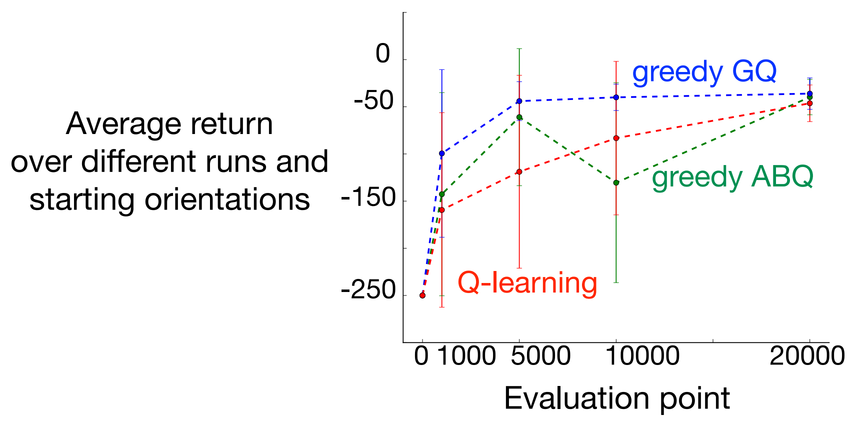


Figure 7.4: Comparison of the algorithms on the time-to-align control task.

7.6 Summary and Conclusions

In this chapter, we presented our second case study. In this case study, we considered a robot control task performed on the Kobuki and assessed the performance of the three algorithms of Q-learning, greedy GQ, and greedy ABQ on the task. Based on our results, all of the three algorithms are useful for learning a simple control task in a real-world setting and under off-policy training. Our results about greedy GQ is consistent with previous work on this algorithm, suggesting that it is a reasonable choice to be used in real-world systems as well as simulated domains.

We found performing empirical comparisons of control algorithms on robots quite difficult. We could only make the comparison for a few runs and were not able to draw meaningful conclusions about the superiority of one algorithm over the others. One important conclusion of this study is that more efficient and effective evaluation methodologies are required to do empirical studies on robot domains.

Chapter 8

The Collision Case Study

In this chapter, we present the last of our three case studies. In the two previous chapters, we looked at two case studies; one considering a relatively simple prediction task and one considering a control task. In both of these studies, we focused on tasks with low-dimensional sensory information. This last case study is in particular interesting because it considers a more complicated prediction task in which a robot wanders in a relatively large space trying to predict when it is going to bump into something using its camera.

Assessing how effective reinforcement learning (RL) algorithms learn general value function (GVF) predictions is valuable. As we discussed in Chapter 6, such assessments is an step towards investigating the suitability of GVFs as a language for representing knowledge. In this chapter, we further assess the applicability of RL algorithms for learning GVF predictions using a relatively complicated robot prediction task.

8.1 The Collision Prediction Task

The collision experiment task is about a Kobuki robot wandering in a wooden pen, asking itself the question "If I go forward, how close am I to bumping into something?". A picture of the Kobuki in the pen is shown in Figure 8.1 and the description of the Kobuki is provided in Chapter 4. The sensors that are available for the learning system to answer this question are the Kobuki's camera and bump sensors. The learning system tries to make a prediction about the bump sensor based on the visual input from the camera.

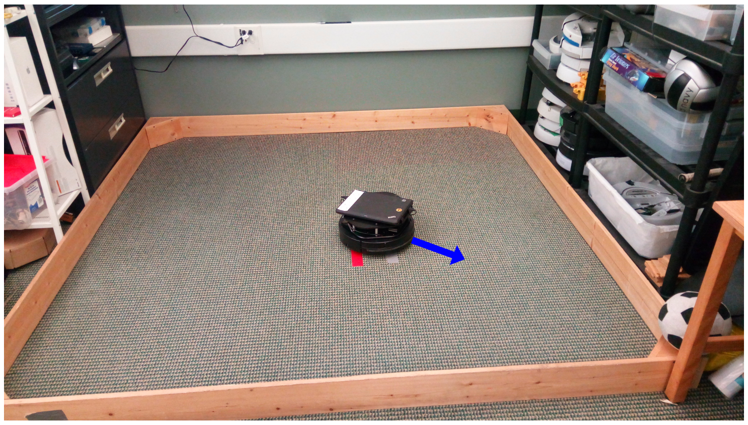


Figure 8.1: The Kobuki in the wooden pen.

8.2 The GVF Formulation of the Collision Prediction Task

We formulate this prediction task as a single GVF. Like the other two case studies, we formulate our prediction task using the language of questions and answers.

Similar to the other case studies, to specify the question part, we have to specify the target policy, continuation function, and cumulant function. For this prediction task, the target policy picks the forward action in all states. The continuation function returns zero whenever the robot bumps into something and returns 0.97 otherwise. The cumulant function returns a binary value that becomes 1 whenever either of the bump sensors are on.

The answer part of this GVF, like the other two case studies, is a linear function with tile coded features. The answer part approximates the answer to the question and this approximation depends on the state representation, learning algorithm, and behavior policy. In the next section, we discuss how we set these things for this GVF in more detail.

8.3 The Collision Prediction Experiment

To construct the feature vector we used the tile coding software. 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 of the 150 numbers separately using 8 tilings and 4 tiles in each tiling. The tile coding software uses an index hash table. For this experiment, we set the size of the hash table to 9600.

We applied the 10 algorithms discussed in Section 5.1 to this prediction task. We made many instances of the algorithms each corresponding to a specific parameter setting. The step-size was in the form $\frac{\alpha}{\text{number of tilings}}$ where $\alpha \in \{0.1 \times 2^x | x = -13, -12, ..., 3\}$. For the gradient based algorithms the second step-size was in the following form: $\alpha_w = \eta \times \frac{\alpha}{\text{number of tilings}}$ where $\eta \in \{2^x | x = -5, -4, ..., 1\}$. The ζ parameter for the ABTD algorithm and the trace parameter λ for all the other algorithms was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.95, 1\}$. The β parameter for ETD(λ, β) was a number in $\{0, 0.2, 0.4, 0.6, 0.8, 0.9, 0.97, 1\}$

We used a behavior policy which would go forward with probability 0.9 and turn left with probability 0.1.

8.4 Performance Measure

As we discussed in Chapter 6, evaluating the learning algorithms on robot domains can be challenging, especially when learning is happening off-policy. In the first case study, we used a performance measure that could be thought of as an estimation of mean square return error (MSRE). To compute the performance measure, we sampled many states from the behavior policy and computed their returns by following the target policy. We call the pairs of state and return that is used for evaluation, the evaluation data. During learning, we used the evaluation data to calculate an estimation of MSRE. In this case study, we did something similar. 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. See Figure 8.2. 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 and the target policy was followed for 100 time steps at the excursion phase. We selected the number 100 to make sure enough time is given for the robot to bump into something. At the end of each excursion phase, we recorded the return and the observations for the state at which the excursion was executed. This information were 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. In the recovery phase, the behavior policy was followed for a random number of time steps coming from the uniform distribution $U(100, 200)$. Following this procedure, we collected 30 runs of learning and evaluation data. Each run contained 150 excursions. Therefore, each run contained 150 streams of learning data and an evaluation data consisting of 150 pairs of state and return.

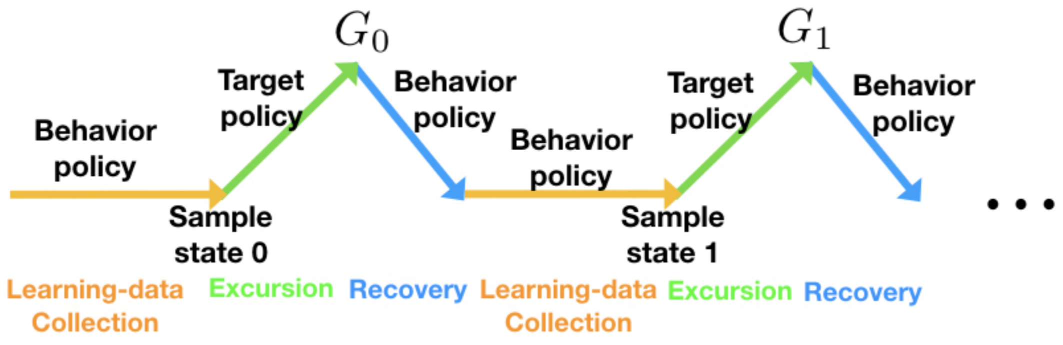


Figure 8.2: The process of collecting the learning and evaluation data for the collision prediction task.

After collecting the learning and evaluation data, we did learning and evaluation offline. For each run, we concatenated the 150 streams of learning data and applied the algorithms to it to learn the predictions. To evaluate the learned predictions, we used the evaluation data collected from the 150 excursions. We computed the difference between the learned prediction, $\mathbf{w}^T\mathbf{x}(s)$, and the return G and got the average over the evaluation data D:

$$\widehat{\text{MSRE}}(\mathbf{w}) = \frac{1}{|D|}\sum_{(s,G)\in D}(\mathbf{w}^T\mathbf{x}(s) - G)^2$$

where D includes all the evaluation data collected by executing the 150 excursions of a run.

We computed this value at each time step to evaluate the algorithms over time. We could use all the evaluation data of a run to compute an estimation of MSRE because we did learning and evaluation offline. If learning was happening online, at each time step we could only use the evaluation data that was collected till that point.

8.5 Results

A comparison between the performance of the algorithms for the case with $\lambda = 0$ is shown in figure 8.3. All the other 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. The parameter studies of the asymptotic performance of the algorithms over the step-size is shown in Figure 8.4.

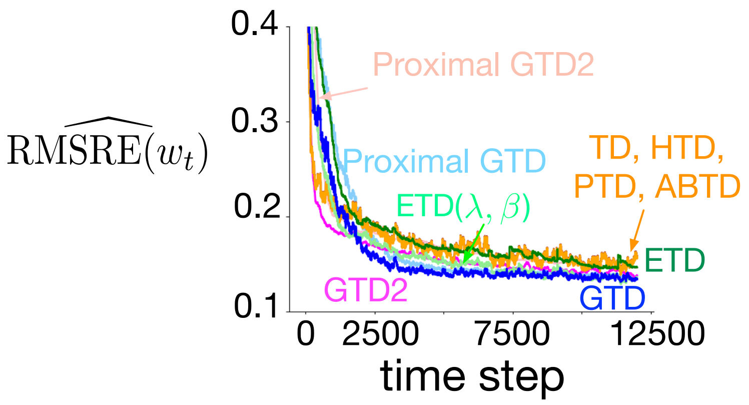


Figure 8.3: Learning curves for the case of $\lambda = 0$ for the collision prediction task. All the parameters are set to values resulting in the best asymptotic performance.

According to the learning curves and parameter studies, the algorithms of GTD(λ), ETD(λ, β), and proximal GTD(λ) reached a lower level of asymptotic error compared to the other algorithms. To establish significance, more runs would be needed.

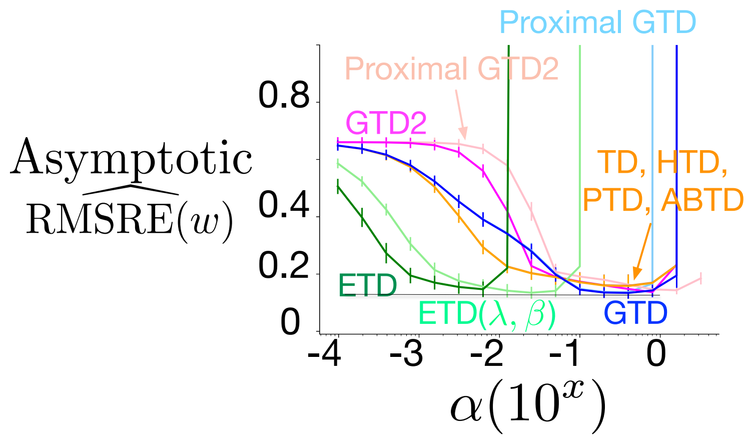


Figure 8.4: Parameter studies of the asymptotic performance over the step-size for the case of $\lambda = 0$ for the collision prediction task. All the parameters are set to values resulting in the best asymptotic performance.

The learning curves and parameter studies of the algorithms for the case of $\lambda = 0.95$ is shown in Figure 8.5 and Figure 8.6 respectively. In this case, ABTD(ζ) and GTD(λ) outperformed the other algorithms with the former being faster.

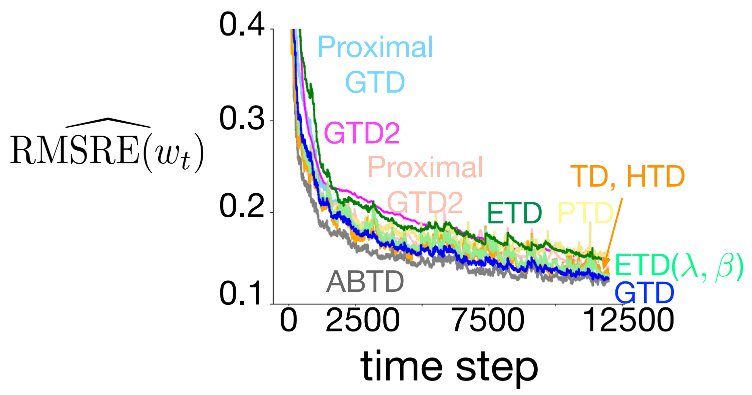


Figure 8.5: Learning curves for the case of $\lambda = 0.95$ for the collision prediction task. All the parameters are set to values resulting in the best asymptotic performance.

In addition to the results regarding the comparison between the algorithms, there were some observations about some of them that were interesting. These observations are also compatible with the results of our first case study. First, ETD(λ), unlike the other algorithms, was not sensitive to the value of λ (See

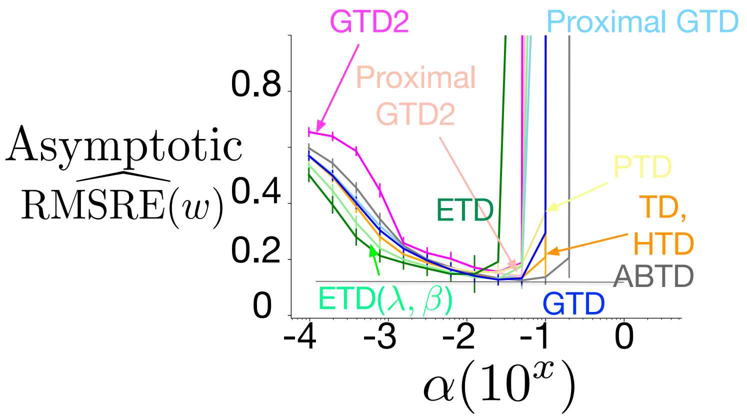


Figure 8.6: Parameter studies of the asymptotic performance over the step-size for the case of $\lambda = 0.95$ for the collision prediction task. All the parameters are set to values resulting in the best asymptotic performance.

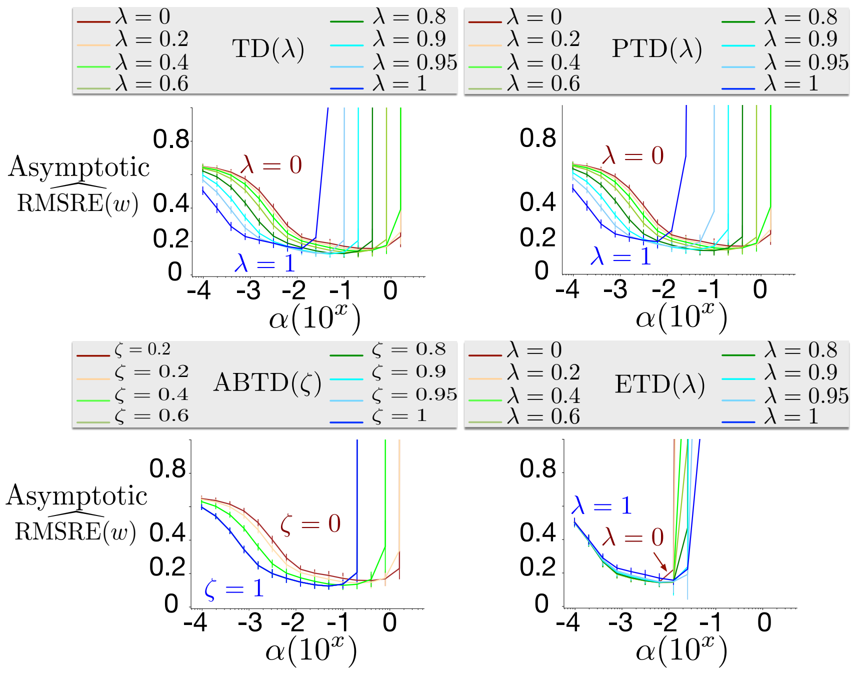


Figure 8.7: The parameter study over the step size parameter for ABTD(ζ), TD(λ), PTD(λ), and ETD(λ)

Figure 8.7). Second, for GTD(λ), proximal GTD(λ), and HTD(λ), as the trace parameter got bigger, the sensitivity to the second step-size reduced (See Figure 8.8). Third, for smaller values of the trace parameter, ABTD(ζ) and TD(λ) performed similarly. However, for larger values of the trace parameter,

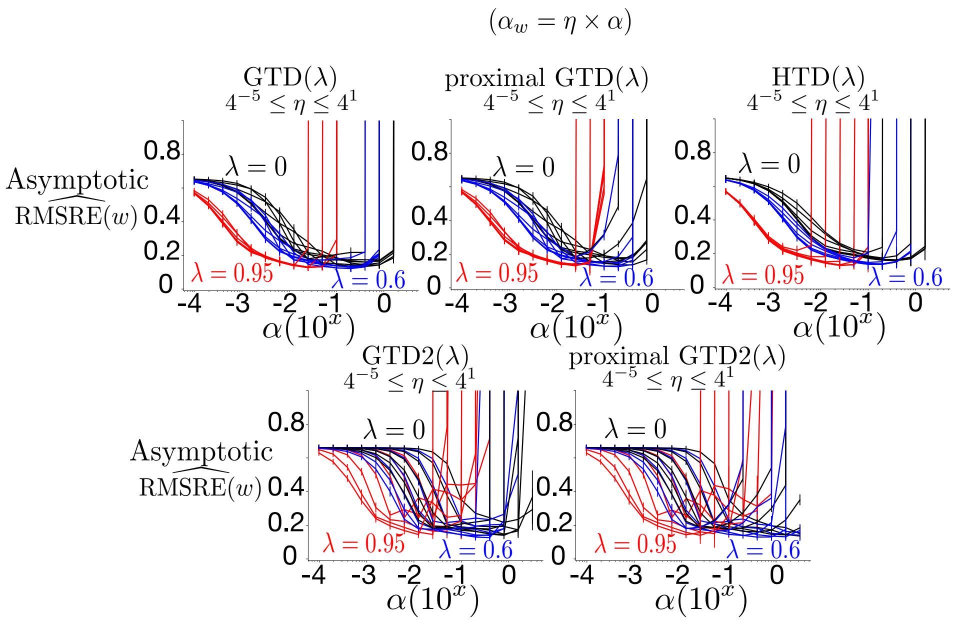


Figure 8.8: The parameter study over the step size parameter for gradient based algorithms. Each curve of each plot is for a specific value of λ and η where all the curves corresponding to $\lambda = 0$, $\lambda = 0.6$, and $\lambda = 0.95$ are shown in red, blue, and black respectively.

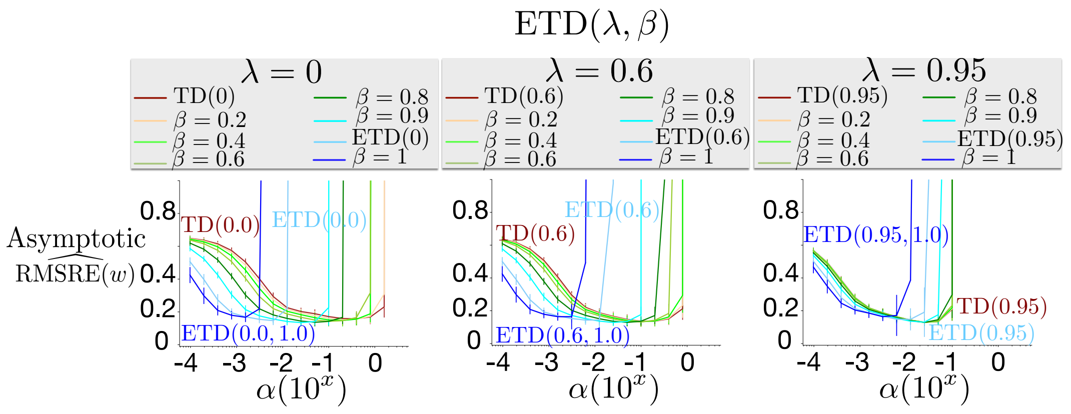


Figure 8.9: Parameter study over the step size parameter for ETD(λ, β). Each plot corresponds to one value of λ and contains multiple lines each corresponding to different values of β where $\beta = 0$ and $\beta = 0.97$ are equivalent to TD(λ) and ETD(λ) respectively.

ABTD(ζ) converged for a much larger range of step-size compared to TD(λ) (See Figure 8.7). Fourth, for ETD(λ, β) as β got bigger, the range of step-size for which the algorithm converged got smaller (See Figure 8.9).

8.6 Summary and Conclusions

In this chapter, we presented the last of our three case studies. We assessed the performance of 10 different temporal-difference learning methods on a relatively complicated robot prediction task. The results of this case study supports those of the first case study, suggesting that many RL algorithms can effectively learn predictions represented as GVFs on robot domains. We can conclude from our empirical comparisons that GTD(λ) is superior to the other algorithms in terms of asymptotic performance in the collision prediction task. Based on our results, ETD(λ, β) also seems to be a promising algorithm, reaching the lowest level of asymptotic error alongside GTD(λ).

Our results also suggest that the evaluation methodology that we used was successful and applicable to a relatively complicated robot domain. Using this methodology, we could compare many RL algorithms systematically, performing parameter studies of the algorithms for many runs.

Chapter 9

Conclusion

In this work, we took small steps toward answering the grand question of how an intelligent system can acquire and maintain a large body of knowledge. We pursued the predictive knowledge approach by using the reinforcement learning (RL) idea of general value functions (GVFs). We presented three case studies in which a robot learned a predictive question in form of a GVF about its world. Our case studies made up the contributions of this thesis. Our first contribution was providing several new examples of learning predictive knowledge using GVFs on robots. Our second contribution was performing empirical comparisons of many different off-policy temporal-difference (TD) learning algorithms on robot domains. Our third contribution was performing systematic studies of the learning process on robots, gaining a better understanding of how such studies should be done in real-world systems. In the next sections, we discuss these contributions in detail and draw some general conclusions regarding each of them. We will close the chapter by a discussion on the limitations of this work and future directions.

9.1 Contribution 1: Providing Several New Examples of Predictive Knowledge on Robots

In this work, we contributed to the idea of predictive knowledge by implementing an instance of it and applying it to several robot domains. A handful of examples of learning predictive knowledge on robots exist in previous work. One of the most important examples is the work by Modayil et al. (2014)

which studied learning thousands of GVF predictions on a robot. That work focused on learning on-policy. However, we considered cases of learning GVFs off-policy on robots. White (2015) also performed an extensive investigation of learning both prediction and goal-directed GVFs off-policy on robots. Our work adds to that work by considering a more complicated task involving visual data. We also did a more systematic study, performing lots of runs of data for each experiment. Our work confirms the result from Modayil et al. (2014) and White (2015) that robots can learn pieces of knowledge in the form of GVF predictions using RL algorithms. These predictions provide the robot with an understanding of how different courses of action affect its sensory readings and can constitute knowledge.

9.2 Contribution 2: Empirical Comparison of Many Off-policy TD algorithms

The second contribution of this thesis was the empirical comparison of many RL algorithms on robot domains. These empirical comparisons were presented in chapters 6 to 8. We focused on off-policy TD algorithms. We also restricted attention to algorithms with linear complexity. Our experiments concerned two prediction problems and one control problem. On the prediction problems, we compared 10 different TD algorithms, including gradient-TD methods and emphatic-TD methods. On the control problem, we compared the three algorithms of Q-learning, greedy GQ, and greedy ABQ.

The empirical comparisons presented in this work can be considered as the most thorough comparisons that have been done on robots. We compared many algorithms and considered a wide range of parameters. Moreover, we did the comparison for a large number of runs. It is unusual to do such systematic comparisons on robots; as a result, we had to do the comparison on relatively small robot domains.

From the empirical comparisons, we reached fairly general conclusions about how different algorithms compare to each other and which algorithms are suitable for learning predictive knowledge. The gradient-TD method of

GTD(λ) and its variant for control, greedy GQ(λ), are good choices for learning predictive and goal-directed GVF questions on robots. In our experiments, GTD(λ) achieved a fairly good level of asymptotic performance in both the Dynamixel and Collision prediction tasks. Moreover, it was robust to the first and second step-sizes. In the time-to-align control task, greedy GQ(λ) learned the optimal policy after a reasonable number of time steps.

Our empirical results suggest that the relationship between GTD(λ) and its state-of-the-art counterpart, ETD(λ), is subtle. We were not able to draw a conclusion about the strengths of ETD(λ) over GTD(λ) for learning predictive knowledge. In the Dynamixel prediction task, ETD(λ) performed substantially better than GTD(λ). However, GTD(λ) was superior to ETD(λ) in the Collision prediction task. The clear conclusion was that ETD(λ) converged for much smaller values of the step-size compared to GTD(λ).

In our experiments, ETD(λ, β) performed well. ETD(λ, β) provides another dimension of flexibility to ETD(λ) controlling the bias-variance trade-off. Based on our empirical results, this additional parameter seems to be useful. In both of our prediction experiments, ETD(λ, β) performs well, reaching the same level of error as the superior ETD(λ) in the Dynamixel prediction task and performing as well as GTD(λ) in the collision prediction task.

9.3 Contribution 3: Systematic Study of the Learning Process on Robots

The second contribution of this work was developing a good methodology for systematically comparing many algorithms on robots. Robots are a great framework for studying predictive knowledge because they are a rich source of data and working with them involve looking closely at the low-level sensorimotor data and understanding the world from the perspective of the robot. While robots are a great framework for investigating the predictive knowledge approach, as we discussed in Chapter 1, performing systematic studies on them entails certain difficulties. In what follows, we provide some general conclusions and suggestions with regard to doing such studies.

First, a good strategy for evaluating off-policy algorithms, learning a prediction task is to collect a reasonable number of sample states from the behavior policy and compute their returns following the target policy. These sample states and their corresponding returns can be stored to be later used to compute an estimation of mean square return error.

Second, evaluating control algorithms on robot domains is challenging. One natural approach for doing evaluation in such settings is to test the learned policy on the robot and compute the return or outcome. This approach is quite expensive for many reasons. First, to evaluate the performance of the learning system over time, we need to evaluate the policy learned after different number of time steps. Second, we need to run tests starting from different points in the state space and probably multiple tests are needed for each point. Finally, we might want to evaluate multiple algorithms each with different parameter settings. To reduce the number of tests in a problem with a small state space, one might be able select the parameter setting for the algorithms using action-value plots. However, for a more complicated task, one might need to use off-policy policy evaluation methods such as the one introduced by Thomas and Brunskill 2016 and Jiang and Li 2015.

Third, we found it essential to examine the data closely when working with robots, including using different visualization tools. Examining the data is crucial for understanding what is happening and verifying whether our expectations of the sensor readings match with the actual readings. Moreover, we need to find whether there is any source of stochasticity in the data and detect unreliable sensors.

9.4 Limitations and Future Directions

Many more studies are needed to investigate how effective RL algorithms are at learning predictions in form of GVFs. In this section, we discuss some of the limitations of our work and possible future studies that could give us more insights for pursuing the predictive knowledge approach.

In our studies we only considered tasks in which the target policy and the behavior policy were similar. To really get a sense of how the algorithms compare to each other, empirical comparisons are needed in cases where the target policy and the behavior policy are much different. In these cases, the importance sampling ratio would be large which could make some of the algorithms unstable.

In all of our experiments, the algorithms learned only one GVF question. A good next step would be to make algorithms learn lots of GVF questions with different target policies, and study the overall performance of them over all these questions.

In all of our case studies, our performance measure was based on following the target policy. It would be valuable to explore different methods of evaluation on robots. For example, there are methods that use data generated by an arbitrary behavior policy to do the evaluation. Example of such methods are off-policy policy evaluation methods.

References

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

Clark, A. (2013). Whatever next? Predictive brains, situated agents, and the future of cognitive science. *Behavioral and brain sciences*, 36(3), pp. 181–204.

Delp, M. (2011). *Experiments in off-policy reinforcement learning with the GQ(lambda) algorithm.* MSc thesis, University of Alberta, Edmonton.

Friston, K. (2005). A theory of cortical responses. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 360(1456), pp.815–836.

Ghiassian, S., Yu, H., Rafiee, B., and Sutton R. S. (2018). Two geometric input transformation methods for fast online reinforcement learning with neural nets. Arxiv:1805.07476.

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

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

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

Littman, M. L., Sutton, R. S., and Singh, S. (2002). Predictive representations of state. In *Advances in Neural Information Processing Systems 14 (NIPS 2001)*, pp. 1555–1561. MIT Press, Cambridge, MA.

Liu B, Liu J, Ghavamzadeh M, Mahadevan S, and 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.

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

Maei, H. R., and 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,* pp. 91–96.

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

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

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

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

Modayil, J., White, A., and Sutton, R. S. (2014). Multi-timescale nexting in a reinforcement learning robot. *Adaptive Behavior*, 22(2):146–160.

Rafols, E. (2006). *Temporal Abstraction in Temporal-difference Networks.* MSc thesis, University of Alberta, Edmonton.

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

Ring, M. (2014). *Representing Knowledge as Predictions (and State as Knowledge).* Unpublished manuscript.

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

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.*

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

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., and 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.

Sutton, R. S., Mahmood, A. R., Precup, D., and van Hasselt, H. (2014). A new Q(λ) with interim forward view and Monte Carlo equivalence. In Proceedings of the 31st International Conference on Machine Learning. JMLR W&CP 32(2).

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

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and 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.

Sutton, R. S., and Tanner, B. (2015). Temporal-Difference Networks. Arxiv: 1504.05539.

Tanner, B., and Sutton, R. S., (2005). TD(λ) networks: temporal-difference networks with eligibility traces. In *Proceedings of the 22nd international conference on Machine learning*, pp. 888–895.

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.

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

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

Adam, A., and 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.