Letting the Agent Take the Wheel: Principles for Constructive and Predictive Knowledge

by

Alexandra Kearney

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

Doctor of Philosophy

Department of Computing Science

University of Alberta

Abstract

Of all the capabilities of natural intelligence, one of the most exceptional is the ability to expand upon and refine knowledge of the world through subjective experience. Therefore, a longstanding goal of Artificial Intelligence has been to replicate this success: to enable artificial agents to construct knowledge of the world through subjective experiences.

This thesis explores how an agent can come to know its world through its experiences by making many predictions about its future sensations, referred to as *Predictive Knowledge*. Specifically, I consider predictions expressed as *General Value Functions* (GVFs): expected accumulations of future sensations conditioned on a particular behaviour. While it has been suggested that GVFs could express all of an agent's knowledge of the world, few examples of applications of GVFs exist. Many present examples of GVF applications require hand-coded relationships between the predictive inputs and an agent's decision-making. I argue that two key challenges must be addressed in order for Predictive Knowledge agents to achieve their potential: enabling agents to determine both *what* their predictions are about and *how* predictions are learned.

In this thesis, I provide one particular solution to both challenges. First, I generalize Incremental Delta-Bar-Delta to be used with temporal difference learning, which I name TIDBD. TIDBD allows Predictive Knowledge agents to modify both the rate at which they learn and the weighting of their features independent of designer intervention during learning. I empirically evaluate the performance of TIDBD in synthetic and real-world robotics prediction tasks.

Having provided agents with a means of modifying *how* they learn a prediction, I then explore how an agent might choose *what* prediction questions to ask. I argue that predictions should be chosen not based solely on their accuracy with respect to some true value but rather with respect to their usefulness in decision-making. Through a series of examples, I demonstrate that selecting predictions based solely on strict measures of accuracy can lead to poor model construction. I show with a worked example how poor model choices can lead to catastrophic performance when model estimates are used in further decision-making. I propose a heuristic for assessing GVF estimates that combines both the accuracy of the prediction and the usefulness of the input features.

Further exploring usefulness in the construction of knowledge, I provide a meta-gradient method that adapts what predictions an agent learns based on feedback from the control learner. I demonstrate that by using meta-gradient descent, an agent can find predictions that resolve partial observability when the control learner uses prediction estimates as additional inputs.

In total, this thesis provides a new perspective on the importance of predictions: prioritizing an artificial agent's use of predictions over a prediction's representational accuracy. In the process of developing this perspective, I introduce new algorithms that enable Predictive Knowledge agents to be applied more broadly with less designer intervention.

Preface

The contents of this thesis are drawn from a number of peer-reviewed publications that were submitted during the course of my studies.

Chapter 3 draws from the following works:

1. **A. Kearney**, V. Veeriah, J. Travnik, R. Sutton, P. M. Pilarski, "Every step you take: Vectorized Adaptive Step-sizes for Temporal-Difference Learning," *3rd Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2017. (Poster and abstract.)

2. **A. Kearney**, V. Veeriah, J. B. Travnik, P. M. Pilarski, R. S. Sutton, "Learning Feature Relevance Through Step Size Adaptation in Temporal-Difference Learning," arXiv:1903.03252, 38 pages, 2019.

3. J. Günther, **A. Kearney**, N. M. Ady, M. R. Dawson, P. M. Pilarski, "Meta-learning for Predictive Knowledge Architectures: A Case Study Using TIDBD on a Sensor-rich Robotic Arm," *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems)*, 2019, pp. 1967–1969. (Extended abstract and poster.)

4. J. Günther, N. M. Ady, **A. Kearney**, M. R. Dawson, P. M. Pilarski, "Examining the Use of Temporal-Difference Incremental Delta-Bar-Delta for Real-World Predictive Knowledge Architectures," *Frontiers in Robotics and AI*, vol. 7, no. 34, 2020, DOI: 10.3389/frobt.2020.00034.

For papers 1 and 2, I developed the experimental design in consultation with Patrick Pilarski. I generalised IDBD and AutoStep to the policy evaluation case independently. Jaden Travnik and Vivek Veeriah provided feedback on experimental design. Jaden Travnik co-wrote the grid-world experiments, Vivek Veeriah wrote the original feature relevance experiments presented in the first manuscript. All prediction experiments involving prosthetic data were solely developed by me. Patrick, Jaden, and Vivek were consulted during the analysis of experimental results. I contributed the majority of the writing for both manuscripts. Editing suggestions were provided by Rich Sutton and Patrick Pilarski.

For manuscripts 2, and 3, I contributed code and I contributed to the experimental design and analysis of both papers. I contributed substantially to the writing of manuscript 2; this manuscript was further expanded on for manuscript 3—a journal paper.

Chapter 4 draws from a series of publications:

1. **A. Kearney**, A. Koop, C. Sherstan, Günther, R. S. Sutton, P. M. Pilarski, M. E. Taylor, "Evaluating Predictive Knowledge," *AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, 2018, pp. 43–46.

2. **A. Kearney**, P. M. Pilarski, "When is a prediction knowledge?," *4th Multidisciplinary Conference on Reinforcement Learning and Decision Making*, 2019, pp. 231-235.

3. **A. Kearney**, A. Koop, P. M. Pilarski, "What's a Good Prediction? Challenges in evaluating an agent's knowledge," *invited paper to the ICLR NERL21 workshop: A Roadmap to Never-ending Reinforcement Learning*, 2021, arXiv:2001.08823

4. **A. Kearney**, A. J. Koop, P. M. Pilarski, "What's a good prediction? Challenges in evaluating an agent's knowledge,"*Adaptive Behavior*, 14 pages, 2022.

The central idea of these papers was developed with Anna Koop. I was the sole designer of the code and primary designer of the experiments presented. In the first iteration of the paper, Johannes Günther and Matt Taylor provided supervision throughout the project. The line of argumentation was developed with both Anna Koop and Patrick Pilarski. I was the primary writer of all manuscripts. Patrick and Anna both provided editorial feedback on various versions of the paper.

Chapter 5 draws from:

1. **A. Kearney**, A. Koop, J. Günther, P. M. Pilarski, "Finding Useful Predictions by Meta-gradient Descent to Improve Decision-making,"*NeurIPS 2021 Workshop: Self-Supervised Learning - Theory and Practice*, Dec. 14, 2021.

2. **A. Kearney**, A. Koop, J Günther, P. M. Pilarski, " What Should I Know? Using Meta-Gradient Descent for Predictive Feature Discovery in a Single Stream of Experience," *Proc. of the 1st Conference on Life Long Learning Agents*, 2022.

I developed the meta-descent method which is evaluated in the preceding manuscripts. Experimental design was developed in consultation with Anna Koop, Johannes Günther, and Patrick Pilarski. All code for this project was written by me. Writing of these papers was joint effort between myself and Anna Koop, with editorial feedback from Patrick Pilarski and Johannes Günther.

Ethics approval was not needed for this thesis.

For Oliver Oxton & Anna Koop

Nihil est in intellectu quod non prius fuerit in sensu.

Acknowledgements

It takes a village to raise a researcher. I am profoundly grateful for the research community and the individuals who supported me in my studies.

- Anna Koop, thank you for your unwavering support throughout my studies. Your mentorship and nurturing made all the difference. I owe a debt of gratitude to you.
- Johannes Günther, thank you for standing by me throughout my studies. During difficult times, your guidance and perspective brought me to creative solutions.
- Matthew Schlegel, thank you for your friendship. You were an invaluable source of discussion, and an anchor I could rely on.
- Brian Tanner, thank you for being a sounding board. Your openness to exploring new ideas with me was a source of energy and motivation.
- Thank you to Jaden Travnik, Dylan Brenneis, and Vivek Veeriah for being exceptional lab-mates. The hours spent at the whiteboard exploring new ideas with you all were everything that I hoped graduate studies would be.

Finally, I would like to thank my supervisory committee. Thank you, Joseph, for your careful feedback and patience. Rich, thank you for giving me licence to explore big ideas. Patrick, thank you for your enthusiasm, positivity, and creativity. Your encouragement throughout the long years helped me become the researcher I am today.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Knowledge is a fundamental aspect of intelligence, both biological and artificial. The ability of humans and animals to conceptualize the world around them is integral to their success and survival. Therefore, a central theme of Artificial Intelligence research is how agents might develop knowledge of their world (McCarthy & Hayes, 1981).

This thesis explores knowledge within the context of Computational Reinforcement Learning, where a growing collection of work seeks to express all of an agent's knowledge of the world as a collection of predictions of future sensations (White, 2015). This approach, referred to as *Predictive Knowledge*, promises to enable agents to build knowledge of their world through interaction in a completely subjective, autonomous way by learning to estimate many predictions of future sensations (Ring, 2021; Sutton, 2009; Sutton et al., 2011; White, 2015), without human labelling of data or instruction. Predictive Knowledge differentiates itself from prior constructivist works (Becker, 1973; Cunningham, 1972; Drescher, 1991), prior continual learning efforts (Ring, 1997), and prior work on predictive state representations (Littman et al., 2002) by taking an epistemic stance: more than a collection of machine learning methods, Predictive Knowledge positions itself as a theory of machine knowledge.

Predictions are a sensible starting point for constructing knowledge and are of interest outside of AI research to those focused on biological intelligence. Humans and many animals build predictive sensorimotor models of the world, and these predictions of experience form the basis of perception (Rao & Ballard, 1999; Wolpert et al., 1995).

A principled and well-understood way of making temporally extended predictions in computational Reinforcement Learning is by estimating value functions (Sutton & Barto, 2018). Value functions estimate the expected long-term accumulation of a signal available to the agent given its current sensations. While commonly used to estimate the discounted sum of future reward, value functions can estimate the accumulation of any stimulus available to an agent via its senses (White, 2015).

Predictive Knowledge agents can be self-supervised (meaning that they do not require input from human designers to learn estimates), fully incremental (learning as data and experience become available to the agent), and computationally efficient (as the complexity of learning is independent of the time horizon over which estimates are learned). In short, Predictive Knowledge provides a plausible approach for artificial agents to conceptualize the world in an ever-growing way, independent of human coaching and instruction.

However, for us to realize the full potential of Predictive Knowledge, two key challenges must be addressed: how do agents learn their predictions, and how do agents decide what to predict? This thesis presents an approach to these challenges.

1.1 Statement of Purpose

Predictive Knowledge agents construct their knowledge of the world by learning to estimate many predictions about their sensations. One challenge Predictive Knowledge agents face is determining what to predict: of the infinitely many predictions an agent could make, *what* predictions are most useful in supporting decision-making? What predictions help an agent make better decisions? I call the problem of determining what to learn *the what* problem.

Machine learning methods are often governed by hyperparameters that specify how learning occurs: parameters that must be tuned for each individual learner. Given that Predictive Knowledge agents learn to estimate many predictions of their world, another challenge is determining *how* an agent should learn by selecting its hyperparameters. I call the problem of modifying how learning occurs as *the how* problem.

In this thesis, I develop learning methods that:

1. Enable a computational agent to decide *what* to learn about by modifying the parameters that specify a prediction.

2. Enable a computational agent to decide *how* to learn a prediction by modifying the parameters that govern learning.

In the following sections, I elaborate on the challenges of determining *what* to learn and *how* to learn and how this thesis presents an approach to solving these challenges.

1.1.1 The *what*

I focus on agents that construct knowledge of their world by learning many predictions about the sensorimotor inputs available to them. In particular, I consider predictions encoded as General Value Functions (GVFs) (White, 2015). General Value Functions have three components that determine *what* a prediction is about: 1) the *cumulant*, or signal of interest being predicted; 2) the *discount function* that determines how far into the future the signal of interest should be attended to; and 3) the *policy* that determines what behaviour must be undertaken by the agent to realise the signal-of-interest's value.

An agent could make infinitely many predictions about its environment; however, not all predictions are useful for informing decision-making. In this thesis, I develop a method that enables an agent to specify the parameters that determine what a prediction is about, independent of human selection based on the agent's subjective experience. In particular, I develop a meta-descent

method that adapts what a GVF is about based on its alignment with the control task an agent is attempting to solve.

This pragmatic approach to selecting predictions contrasts with methods that choose predictions based on assessing GVFs based on how well they represent the environment. I argue against representation as the focus of knowledge and expand on this in a series of worked examples. In particular, I show that evaluating predictions in terms of their error—how well they represent some *true* underlying value over a set of states—does not imply that their learned estimates will help inform further decision-making. Having empirically explored the limitations of representation, I discuss how an agent can learn general value functions that are useful for decision-making. This thesis empirically demonstrates that through meta-descent, an agent can learn predictions that can be used as input features to improve task performance.

1.1.2 The *how*

Machine learning methods are often governed by parameters that modify how learning occurs. A particular parameter that is common in many machine learning methods is the *step size*, or *learning rate*. Learning methods can be sensitive to the step size parameter, where the parameter is carefully tuned for each new learning problem. For ordinary learning agents, this means that for each individual learning task (e.g., a classic control problem vs a video game) the step size is manually selected to ensure reasonable performance. There exist Machine Learning methods that do not have a learning rate. The assumptions made in these methods are not suited to the Predictive Knowledge setting. For example, Least-squares TD learning (Boyan, 2002) is the most data efficient form of linear TD learning, and eliminates the need for a step-size parameter; however, to achieve this data efficiency LSTD learns offline, and has a higher computational and memory cost: traits not suited to the Predictive Knowledge setting where an agent incrementally and continually updates many prediction estimates online and incrementally. Moreover, because LSTD does not use a step-size, it never forgets past experiences. In stationary environments where the world and the agent's policy do not change over time, this persistence of

past experiences may not be a concern. However, in the Predictive Knowledge setting, where an agent must continually learn online in a non-stationary environment, LSTD's elimination of the step-size is a liability, rather than an advantage. Incremental learning methods that have a deterministic step-size schedules that are effectively parameter-free exist, such as HL(λ) (Hutter & Legg, 2008) and AlphaBound (Dabney & Barto, 2012). In both cases, it assumed that the environment is stationary. The Predictive Knowledge regime considers an agent learning continually in its environment, ideally the real-world. Much preliminary work in both designing and applying predictive knowledge agents has been set in the real-world (Edwards, Dawson, Hebert, Sherstan, et al., 2016; Modayil et al., 2014; Sutton et al., 2011): an inherently non-stationary environment. For these reasons, existing parameter-free step-size adaptation methods are not well suited for the Predictive Knowledge setting.

In this thesis, I provide a method of adapting step sizes for temporal-difference learning, the learning method that underpins the estimation of General Value Functions. In particular, I develop a step-size adaptation method suited for real-world, never-ending learning contexts. Predictive Knowledge as a branch of research aims to design learning methods that enable agents to learn continually for the duration of their lives: constructing and refining knowledge of the world as an ongoing practice the agent is engaged in.

Step sizes can be assigned per feature, enabling the update to a weight to be tuned based on the corresponding input. Adapting step sizes on a per-feature basis provides a basic representation learning method. In this thesis, I bring Incremental Delta-Bar-Delta (Sutton, 1992) and AutoStep (Mahmood et al., 2012) to policy evaluation. The goal of adapting these meta-learning methods to suit General Value Functions is to provide a robust way of enabling agents in continual learning settings to both continuously adjust the step size values that govern temporal-difference learning and also perform simple representation learning by weighting features independently in updates. I generalise these methods for on-policy prediction and empirically assess their performance on various synthetic and real-world robotics prediction tasks.

1.2 Contribution 1: How Should an Agent Learn? Adapting Step Sizes by Meta-gradient Descent

This thesis first starts by addressing the *how*: this thesis contributes a new step-size adaptation algorithm for temporal difference learning, which I call Temporal Difference Incremental Delta-Bar-Delta (TIDBD). TIDBD extends Incremental Delta-Bar-Delta (Sutton, 1992) from the supervised learning setting to TD learning (Sutton, 1988). TIDBD outperforms TD on a conventional prediction problem, achieving lower prediction error. To adapt the step sizes, TIDBD introduces an additional meta step size parameter. TIDBD is sensitive to its meta step size in the same way that TD is sensitive to its ordinary step size. To ameliorate this sensitivity, I incorporate AutoStep's (Mahmood et al., 2012) normalisation into TIDBD. I name TIDBD with auto normalisation AutoTIDBD. This thesis' experiments show that on a non-stationary prediction problem, AutoTIDBD outperforms the existing step-size adaptation methods for policy evaluation that were compared against in this thesis. On the same non-stationary prediction problem, TIDBD performs representation learning by assigning small step sizes to noisy features not relevant to the prediction task.

1.3 Contribution 2: What's a good prediction? New Directions for Evaluating an Agent's Knowledge

Having proposed a method of step-size adaptation for TD learning, this thesis turns its attention to the problem of *what* to learn. To do so, we critically examine existing methods of evaluating prediction estimates suited for Predictive Knowledge agents. A challenge for Predictive Knowledge agents is determining whether to rely on a learned estimate for decision-making. A common practice in the community is to evaluate a prediction based on its error. The basis of this belief stems from Predictive Knowledge's perspective on truth: that an

agent's beliefs about the world—its predictions—are true if they can be verified by comparing the estimated value to what is observed (Ring, 2021; Sutton et al., 2011).

This is hardly a surprising stance—similar assumptions are made throughout machine intelligence research. Researchers often use accuracy as a means of determining the quality or usefulness of a learned estimate (Bengio et al., 2017; Russell & Norvig, 2010; Sutton & Barto, 2018). Throughout AI, much research is driven by accuracy, and by how well learned estimates beat the state-of-the-art on benchmarks (Bellemare et al., 2013; Deng, 2012; Krizhevsky & Hinton, 2009; Panayotov et al., 2015). Time has shown that when state-of-the-art-systems are deployed, and are finally evaluated through their use, unforeseen practical effects are brought to bear. Patches applied to real-world scenes can catastrophically disrupt image classification systems (Brown et al., 2017), imperceptible permutations to auditory inputs can cause classification to speech recognition systems (Qin et al., 2019), and perturbations to inputs can disrupt learned policies of goal-seeking agents (Huang et al., 2017).

Over a series of worked examples, I empirically demonstrate that relying solely on strict measures of prediction error to build an agent's knowledge can be misleading. In particular, I demonstrate that when predictions are put to use, accuracy does not always reflect utility, or usefulness in decision-making. First, on a simple prediction task, I demonstrate how existing online evaluation methods for GVFs do not always rank predictions effectively. A core motivation of GVFs is their proposed use as predictive features, or as signals of interest for more complex predictions. Using an example drawn from prior work (Ring, 2021), I demonstrate that relying on existing error metrics alone for GVF selection can negatively impact the learning progress of agents which depend on estimates as inputs. Having explored the pitfalls of evaluating by error alone, I demonstrate that by analysing the relevance of input features and prediction error, it is possible to differentiate between useful and non-useful estimates. To facilitate this analysis, I generalise AutoTIDBD to the off-policy prediction setting. This chapter presents an argument against verificationism (Goldman, 1976) as a basis for truth in machine knowledge.

1.4 Contribution 3: What Should an Agent Know? Online Discovery of Useful Predictions

Finally, this thesis presents an approach to determining *what* to predict. A core challenge for Predictive Knowledge agents is determining of all the possible predictions that could be learned, which can support decision-making. This thesis contributes a method to learn control learner feedback the parameters that specify what a GVF question is about. In experiments on a small, partially observable problem, an agent could learn to specify predictions that—when used as input features—enabled the agent to solve the control problem. The agent learned to predict aspects of the environment that were different from what experts constructing GVFs for the agent selected: the agent found an alternate solution to the problems. In a sparse reward problem, an agent that selected its GVFs was able to outperform an agent using expert-chosen predictions on a sparse reward task. Similar to the first domain, GVFs chosen by the agent are different from expert-chosen GVFs in this sparse reward setting.

1.5 Thesis Adjacent Contributions:

This thesis is about developing algorithms that further enable applications and refinement of Predictive Knowledge agents. I build on prior work that proposes an agent's knowledge of the world could be well thought of as a collection of value estimates (Koop, 2008; White, 2015). However, the notion of *Predictive Knowledge* not yet fully developed. To better understand how to develop algorithms for Predictive Knowledge, I also sought to refine the definition, commitments, and philosophical stance of Predictive Knowledge. While important to the project of Predictive Knowledge, these works were omitted to keep the thesis concise and the narrative focused. While the following papers have been omitted, they may be of interest to the reader:

A. Kearney, P. M. Pilarski, "When is a Prediction Knowledge?", *4th Multidisciplinary Conference On Reinforcement Learning and Decision Making* (RLDM), July 7-10th, Montreal, Quebec, Canada, 2019.

A. Kearney, O. Oxton, "Making Meaning: Semiotics Within Predictive Knowledge Architectures" *4th Multidisciplinary Conference On Reinforcement Learning and Decision Making* (RLDM), July 7-10th, Montreal, Quebec, Canada, 2019.

A. Kearney, J. Gunther, P. M. Pilarski, "Prediction, Knowledge, And Explainability: Examining The Use of General Value Functions in Machine Knowledge", *Frontiers in Artificial Intelligence* 2022

J. Gunther, **A. Kearney**, M. R. Dawson, C. Sherstan, P. M.Pilarski, "Predictions, Surprise, and Predictions of Surprise in General Value Function Architectures," *Proceedings of the AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, Arlington, Virginia, October 18-20, 2018, pp. 22–29.

1.6 Structure of the Thesis

This thesis is six chapters long. The second chapter outlines a history of Predictive Knowledge and related work. The third chapter presents a method of automatically adapting a learner's step sizes on a per-feature basis: enabling agents to modify *how* a prediction is learned by modifying their learning parameters. The fourth chapter challenges commonly held beliefs about evaluation of Predictive Knowledge—I demonstrate that existing only evaluation methods do not reliably differentiate between useful and useless predictions. The fifth chapter presents a method enabling Predictive Knowledge agents to specify *what* to predict by stochastic meta-descent. The sixth chapter concludes the thesis by summarising the contributions of this work and discussing how future work might further develop Predictive Knowledge agents following in the steps of this thesis.

Chapter 2

Background

In this chapter, I present the general context of this thesis. I survey the ideas and literature which this thesis builds upon. Details about each contribution will be discussed in their respective chapter.

2.1 A Brief History of Predictive Knowledge

This thesis is concerned with *Predictive Knowledge* (Koop, 2008; Rafiee, 2018; Ring, 2021; Sutton, 2009; White, 2015): an approach to machine knowledge that is influenced by *constructivism* (Piaget, 1954; Piaget & Duckworth, 1970). Constructivism argues that knowledge is continuously expanded upon and refined through an individual's experimentation with the world (Piaget & Cook, 1952). Importantly, knowledge is not external to the individual (as an empiricist might claim), or that a child innately understands the world from birth (as a Nativist might claim). Rather, intelligent agents continuously integrate their subjective experiences to conceptualise the world around them from(Piaget & Cook, 1952).

Predictive Knowledge is strongly influenced by constructivist theories of learning: specifically, that knowledge of the world should be grounded in subjective experience. However, Predictive Knowledge agents do not seek to explicitly replicate the physical and biological learning systems from which they draw inspiration. For instance, Predictive Knowledge agents do not enforce strict adherence to Piaget's proposed stages of development (c.f. Piaget and Cook, 1952). Outside Predictive Knowledge, there exists constructivist

approaches to machine knowledge that more closely approximate Piaget's theories by learning schemas (Chaput et al., 2003; Drescher, 1991; Guerin & Starckey, 2009; Kansky et al., 2017). Moreover, there are works that take a developmental approach to guide how agents incrementally construct knowledge of their world through interactive learning: methods that enable agents to learn hierarchical spatial representations of the environment (Pierce & Kuipers, 1997), build representations of objects from an agent's experience (Modayil & Kuipers, 2008), and enable agents to postulate of sub-tasks to learn skills (Mugan & Kuipers, 2008). From the perspective of knowledge acquisition, prior work has explored how to drive an agent's behaviour to construct knowledge based on principles from developmental learning (Oudeyer et al., 2005). Such works have shown that developmental approaches to exploration can enable agents to learn more efficiently than simply exploring the sensor-space (Baranes & Oudeyer, 2013), and that by taking an intrinsically motivated approach to exploration an agent can learn complex behaviours (Forestier et al., 2022).

Following in the footsteps of these prior lines of work: instead of directly replicating biological learning, Predictive Knowledge agents learn to estimate predictions about future subjective sensations conditioned on the agent's behaviour. These predictions may be interrelated hierarchically (Ring, 2021; Schlegel et al., 2021) to express abstractions about the agent's world using only the agent's subjective experiences. In this sense, Predictive Knowledge is another branch in the tradition of building agents that learn to build their conceptualizations of the world in a way that is grounded in subjective experience.

Constructivism's influence on Predictive Knowledge is best seen through a series of perspectives that guide the development of learning methods:

Learning is subjective: Predictive Knowledge research asserts that an agent cannot rely on labelled examples, as is the case in supervised learning (Russell & Norvig, 2010); and that an agent cannot rely on hand-coded predicates and inferential relationships, as is the case in expert systems (Russell & Norvig, 2010). For instance, to conceptualise a cat, an agent may *not* receive labelled data that indicates whether a cat is present in an image; such information comes from human annotation and not the agent's senses. Once

all labelled data has been exhausted, there is no possible way for an agent to continue to learn about the concept *cat* without further annotated examples. There is no way for an agent to expand its categorisation beyond what has been given via supervision, and there is no way for an agent to expand its understanding of the world by forming new categories (Sellars, 1956). An agent's self-supervision based on its experiences is often referred to in the Predictive Knowledge literature as *grounding*: all of an agent's beliefs are grounded exclusively in terms of their sensations (Sutton, 2009; Sutton et al., 2011; White, 2015).

Learning is interactive and action conditional: Drawing from insights gained through the study of biological intelligence, it is evident that an agent's perception of the world does not arise solely from the sensations it receives but also from how those sensations change as a result of its actions (Nöe, 2004). In much the same way, Predictive Knowledge agents must acquire knowledge through their interactions with the world (Sutton, 2009), and this knowledge must be contingent upon the agent's behaviour (Ring, 2021).

Learner capacity should increase: Constructive accounts of knowledge are predicated on the ability of an individual to expand what they know about the world over time (Piaget & Duckworth, 1970). An infant starts with a limited understanding of its world. Through the stages of development, a child gradually refines and expands their understanding of the world (Piaget, 1954). Similarly, a Predictive Knowledge agent should be able to expand its understanding of the world over time: a Predictive Knowledge agent should be able to add and learn new predictions of the world over time to expand its understanding of the world. This requirement is not yet met by the field. Recent work has explored how an agent may allocate fixed capacity (Schlegel et al., 2018; Veeriah et al., 2019), but there are few examples exploring how capacity might expand over time (e.g., Makino and Takagi, 2008).

Learning is continual: In order for an individual to progress through the stages of development, they must be continually integrating new experiences and learning from that experience (Piaget, 1954). Drawing from continual learning (Ring, 1997), a Predictive Knowledge agent must be able to refine their beliefs

about the world for the duration of their lives: there is no distinct training, testing, and deployment phase for Predictive Knowledge agents (White, 2015). Rather, a prediction is learned, evaluated, and used simultaneously.

2.2 Reinforcement Learning & Predictive Knowledge

Just as constructive accounts of knowledge begin with a child exploring their environment, this thesis' account of machine knowledge starts with an artificial agent interacting with its environment. I specifically consider *Reinforcement Learning* agents that interact with their environment through trial-and-error to maximise the discounted sum of future signals from the environment: the *reward* signal (Sutton & Barto, 2018). The relationship between an agent and its environment is often phrased as a loop, where an agent takes an action, which may influence its environment, and then observes the environment.

An agent's sequential interaction with the world is described as discrete time-steps. At each new time-step t, the agent observes the environment—encoded as an observation vector $\mathbf{o}_t$—and takes an action a_{t+1}, which describes how it uses its effectors. The agent then observes how the environment changed o_{t+1} and receives a reward signal r_{t+1}. The *policy* an agent is following is a mapping of observations to actions, where $\pi(a_t|\mathbf{o}_t)$ is the probability of taking action a_t given the agent's observations $\mathbf{o}_t$. After taking an action, the time-step increments, and the agent observes the environment on this new time-step $\mathbf{o}_{t+1}$. The endless continuing loop of observations and actions over time is the agent's subjective stream of experience. The primary focus of this thesis is on how an agent develops relationships between its observations and actions—from its experience—to understand the world". In particular, I consider how an agent might conceptualise its environment by estimating many value functions.

In computational Reinforcement Learning (Sutton & Barto, 2018), value functions describe the expected discounted sum of a reward signal received by the artificial agent. By estimating the expected sum of future reward, or *return*, an agent can learn which action a_t is expected to yield the greatest return

given what it presently senses from its observations. Value functions can be generalised to *General Value Functions* (GVFs) that can express not only the accumulation of reward but the sum of any future signal available to the agent via its sensors (White, 2015). Three parameters define the discounted return a GVF estimates: 1) the policy the prediction is conditioned on π; 2) the signal of interest being predicted, called the *cumulant* c; 3) and the amount by which the return is discounted, called the *discount* γ. Having defined these three parameters, the expected return is $\mathbb{E}_\pi[G_t] = \mathbb{E}_\pi[\sum_{k=0}^{b}(\prod_{j=1}^{k}(\gamma_{t+j}))c_{t+k+1})]$.

Learning General Value Functions can provide important insights into the environment: learned estimates can be useful predictive features that the agent can leverage for decision-making (Jaderberg et al., 2017; Pilarski, Dawson, Degris, Carey, Chan, et al., 2013). For example, by learning to estimate the future value of the quality of a weld, an intelligent laser welding system could adapt its controls in anticipation of quality changes (Günther et al., 2016), enabling the control agent to achieve its goals.

Algorithm 1 TD for learning the accumulation of a cumulant c with linear function approximation.

initialise:
  Initialise vectors $\mathbf{w} \in \mathbb{R}^n$. Choose a function approximator ϕ that transforms the observations into a feature vector of size n. Choose a step size $\alpha > 0$. Choose a discount $\gamma > 0$.
begin:
  Observe initial stimulus from the environment $\mathbf{o}$.
repeat interaction with environment:
  Take action a, observe next stimulus $\mathbf{o}'$ and cumulant c.
  $\delta \leftarrow c + \gamma \mathbf{w}^\top \phi(\mathbf{o}') - \mathbf{w}^\top \phi(\mathbf{o})$
  $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\alpha} \delta \phi(\mathbf{o})$
  $\mathbf{o} \leftarrow \mathbf{o}'$
until termination

Value functions can be estimated online and learned incrementally using temporal difference learning (Sutton, 1988). Temporal difference learning is a method of iteratively updating the value estimate by taking the difference between the observed value and the previous estimate. In Algorithm 1, I describe TD learning of General Value Functions. The value is estimated by

updating a set of weights $\mathbf{w}$ incrementally (line 9), which, when combined with a transformation of the observations $\phi(\mathbf{o})$, produces the estimated value $v(\mathbf{o}) = \mathbf{w}^\top \phi(\mathbf{o})$. The weights are updated according to a temporal difference error δ (line 8). The TD error is a *bootstrapped* error: the difference between the estimated value $v(\mathbf{o}_t)$ and the observed value $c + \gamma v(\mathbf{o}_{t+1})$ depends on the estimate itself. The target value that the error δ is taken with respect to, in part, is determined by the estimate on the following step: $v(\mathbf{o}') = \mathbf{w}^\top \phi(\mathbf{o}')$. The weight updates on each learning step are moderated by a *step size* α, or a *learning-rate*: a small value that determines the magnitude of the weight update at each time step to reduce the observed error.

TD methods are often online learning methods where the agent can learn while interacting with its environment (Degris & Modayil, 2012; Sutton, 1988; White, 2015). Applications of GVFs have been successful in real-time learning problems, where the agent receives observations at a high frequency (Edwards, Dawson, Hebert, Sherstan, et al., 2016; Modayil & Sutton, 2014; Modayil et al., 2014). Moreover, TD methods are often incremental: when the agent observes the environment $\mathbf{o}_t$, it performs a learning update, and then the observation is discarded. Observations need not be stored in a buffer for extended periods, as with replay buffers (Lin, 1993; Schaul et al., 2016). Finally, one of the most compelling features is that learning is entirely self-supervised[1] (White, 2015): while a designer may choose the parameters that govern learning, the agent does not rely on information provided by an expert for learning. It does not require a training set to learn a model— the agent's own experience stream provides all the data needed for learning. Such desirable traits have made GVFs a compelling foundation for developing machine knowledge.

Predictive Knowledge describes a sub-field of research in Reinforcement Learning which seeks to build constructive intelligence through general value functions (White, 2015). An agent learns a collection of value functions that estimate the future value of many aspects of the environment simultaneously

[1]Self-supervised learning can refer to multiple sub-fields of Machine Learning. In this thesis, I use self-supervised learning to mean learning where no human-annotated labels are provided.

(Modayil et al., 2014). Predictions need not only be about the immediate sensations available to an agent; GVFs can also estimate the future return of other estimates (Schlegel et al., 2021) or internal values generated by learning itself (Günther et al., 2020; Günther et al., 2018; Sherstan et al., 2016). By layering predictions, an agent can learn ever-more complex predictions of their sensations (Ring, 2021; Schlegel & White, 2022). The basis of Predictive Knowledge research is the claim that such interrelated forecasts could form the basis for an agent's knowledge of the world (Ring, 2021; Sutton, 2009; White, 2015).

I divide progress in Predictive Knowledge research into four categories: 1) conceptual progress that further expands understanding of the expressivity of predictions, 2) architectural progress that proposes new ways of structuring Predictive Knowledge agents, 3) algorithmic progress that furthers the stability of learning methods used by Predictive Knowledge agents, and 4) applications of Predictive Knowledge agents in real-world settings.

2.2.1 Conceptual progress in developing Predictive Knowledge

The idea that an agent could build a model of its world grounded only in the agent's own experience through TD learning is as old as TD itself (Sutton, 1988). The idea was later refined with the introduction of General Value Functions (Sutton et al., 2011; White, 2015) as a generalisation from the prediction of reward to predictions of an agent's stimuli.

Since the introduction of TD learning, there has been a progression of work seeking to expand and explore what predictive estimates can express. An example of such lines of work is *introspective agents*. Sherstan et al. examines how an agent might predict signals generated by its own learning processes—how an agent might introspect by making predictions about its own learning (Sherstan et al., 2016). One such internal signal generated during learning is the prediction error. Günther et al. suggests that the unexplained error an agent experiences is a signal useful to predict (Günther et al., 2018). By making predictions of error, an agent could anticipate regular unexplained

disturbances in its environment (Günther et al., 2018).

Other work explores how different ways of learning TD estimates can increase the expressivity of predictions. Drawing from the successor features framework (Dayan, 1993), recent work has explored how decoupling the environment dynamics from the reward function enables an agent to learn more quickly when the reward changes (Barreto et al., 2017). Because an agent learns the structure of the environment independently, using successor features, agents can learn new predictions about diverse aspects of the environment more quickly (Sherstan et al., 2018). Similarly, Universal Function Approximators parameterise value functions over not just states but also goals (Schaul et al., 2015). This additional input parameter enables value estimates to generalise to new and unseen goals. Universal Successor Feature Approximators (USFAs) (Borsa et al., 2019; Ma et al., 2018) combine UVFAs and SFs to further improve inference to new tasks.

Further work on expanding expressiveness examines how value estimates can generalise over time scales. It is not always possible to know which time scales are useful for agents to learn about. One can imagine that it might be advantageous to reason over multiple time scales —as biological agents do (Tanaka et al., 2016). Γ-nets describe a method that enables agents to generalise learned predictions over arbitrary discount values γ (Sherstan et al., 2020). Extending prior works that use learned embeddings to generalise over goals (Schaul et al., 2015), and works that use γ as an input parameter (Xu et al., 2018) to a control learner, Γ-nets train over many possible discount values γ. By enabling predictions to generalise from fixed temporal horizons to multiple time-scales, Γ-nets improve the expressivity of learned predictions.

The aforementioned works add to the literature by proposing new predictions or new learning methods; another line of work seeks to unify many existing lines of work in computational Reinforcement Learning under the framework of General Value Functions and Predictive Knowledge, including policy gradient methods, successor features (Dayan, 1993), and option models (Precup, 2000) with GVFs (Comanici et al., 2018).

Not all the works described explicitly mention Predictive Knowledge as

their motivation. However, each of these works further expands what can be expressed using predictions and can be understood through the lens of predictive knowledge.

2.2.2 Architectural developments in defining the structure of Predictive Knowledge agents

Another category of research explores how Predictive Knowledge agents might be structured. If an agent is learning many predictions in parallel, how are they organised? Are they arranged as one flat structure, or are there layers of many predictions? How does a control agent use the predictions in decision-making? TD networks describe how collections of predictions can be hierarchically structured and learned using temporal difference methods (Silver, 2013; Sutton & Tanner, 2004). A natural question is how one might expand the predictions learned by an agent: how additional predictions learners might be added to the network so that an agent could know more about its surroundings over time. One proposal was to incrementally add nodes to a TD network based on residual prediction error (Makino & Takagi, 2008). If a prediction in the TD network has unexplained residual error, a node is added to the network. General Value Function Networks (GVFNs) (Schlegel et al., 2021) generalises the idea of learning networks of predictions to General Value Functions.

Another important decision in the structuring of agents is how learned predictions are used in decision-making. One option is to use the predictions to drive fixed responses in anticipation of some stimuli: to give the agent reflexes (Modayil & Sutton, 2014). A straightforward yet relatively unexplored choice is to use the predictions as additional input features to a control agent. Another option is to use the predictions as an auxiliary task (Jaderberg et al., 2017). An auxiliary task is an additional learning constraint placed on an agent in addition to learning the value of a given state (e.g., predictions of pixel values in a video game environment). While the agent does not use the forecasts themselves, the predictions act as a regulariser by contributing to the loss function. By learning to estimate predictions, the agent learns features useful not only for the control task at hand but also in the modelling of different

aspects of the environment (Jaderberg et al., 2017).

2.2.3 Algorithmic developments for Predictive Knowledge agents

A foundational component of Predictive Knowledge agents is the mechanism by which they learn their predictions. Recent years have seen the development of numerous off-policy TD learning methods (Ghiassian et al., 2018; Hackman, 2012; Hallak et al., 2016; Liu et al., 2016; Maei, 2011; Mahmood, 2017), each with their strengths and weaknesses. New work has empirically assessed off-policy learning algorithms to better understand their performance characteristics in synthetic (Ghiassian & Sutton, 2021) and real-world robotics domains (Rafiee, 2018) with a focus on their usefulness as a learning mechanism for Predictive Knowledge agents.

When an agent learns about many aspects of the environment, how does the agent behave? How does an agent balance the needs of many independent learning processes? By taking into account the learning progress of several sub-learners, an agent can better explore the environment, generating interesting behaviour (Linke et al., 2020).

2.2.4 Applications of Predictive Knowledge

Finally, I discuss how predictions have been used to inform decision-making in real-world agents. A mature line of work applying Predictive Knowledge to control and decision-making has explored the use of predictions in adapting the control interfaces of myoelectrically controlled bionic limbs.(Pilarski et al., 2011). By learning value estimates, an agent can anticipate many signals produced in the operation of a bionic limb, including the control commands of a subject who uses a bionic limb to complete a task (Pilarski et al., 2012; Pilarski, Dick, et al., 2013). Using these estimates, the control interface can be modified so that a user can complete tasks more quickly (Edwards et al., 2014). In addition to modifying the control interface, agents can also cooperatively take actions using the limb to reduce the burden of control on a user (Edwards, Dawson, Hebert, Sherstan, et al., 2016; Sherstan et al., 2015). Such work can be viewed

as an introduction of Predictive Knowledge agents as intelligent assistants that support human users in their decision-making—collaborative agents that support human decision-makers. Further work has explored how assistive agents can track important events in a temporally complex environment and provide signals to a user to alert them about approaching events (Brenneis et al., 2021; Butcher et al., 2022; Pilarski et al., 2022). In total, Predictive Knowledge agents are effective in learning about the world and providing information to human collaborators about oncoming events.

Each of the aforementioned applications are in settings where an agent is assisting or collaborating with a human. Other works have explored how an agent might be able to use predictions to better inform their own decisions. In industrial laser welding, maintaining a high-quality weld is integral to the strength of the manufactured component (Günther et al., 2016). Pavlovian control has been used to produce over-ground walking in animal models of hemisectional spinal cord injuries (Dalrymple et al., 2020). The learned reflexes of the pavlovian controller demonstrated the ability to learn quickly, personalise and adapt to new users, and recover from induced mistakes during walking (Dalrymple et al., 2020).

2.3 Modifying Learning by Adapting Step Sizes

The foundation of Predictive Knowledge agents are their predictions: the value functions an agent estimates to make sense of its world. How an agent learns each of these predictions is an important choice. The choice of *learning parameters*—values that govern learning–has an important role in determining how an agent learns to estimate its value functions. How well an agent can estimate a specific value function is in part determined by the learning parameters chosen. Of particular interest to this thesis is the *step size* parameter: a small constant that determines how much a value estimated is updated on each time-step.

Given the impact of the step size chosen on the performance of a learner, it is common for experiment designers to sweep over many parameter settings

to find a value for the step size α that results in adequate performance in offline machine learning(Bergstra et al., 2011; Bergstra & Bengio, 2012). To sweep over parameter settings, multiple instantiations of an agent are run on a target task—or set of tasks—for a period of time. After each experiment terminates, the performance of an agent is analysed—e.g, a ranking based on value error—so that the best combination of parameters can be identified.

Finding step size values via parameter sweeps is not often suited for Predictive Knowledge agents. In simulated environments, it may be possible to run many agents in parallel to find appropriate learning parameters; however, it is impractical to perform parameter sweeps in the long-lived continual learning settings that Predictive Knowledge focuses on: e.g., real-world robotics settings where there may only be access to a single physical robot. Moreover, a goal of predictive and constructive approaches to machine knowledge is the construction of knowledge over time: an agent's structure may not be fixed over time. For instance, a network of predictions might be modified over time as an agent learns more about its environment: predictions might be added over time to increase the agent's capacity to learn about its world (Makino & Takagi, 2008). In such cases, halting an agent's learning to test a variety of parameter settings is impractical at best.

If it is impractical to test a broad sampling of step sizes ahead of time, an alternative approach is for an agent to modify its step size over time. It is important for Predictive Knowledge agents that a step-size adaptation method can adapt to changes in the environment. Online continual learning in the real-world—a setting Predictive Knowledge research prioritises—is often a non-stationary learning setting. What determines optimal behaviour may change based on environmental changes (Milan et al., 2016), changes in the behaviour of other agents (Bowling & Veloso, 2002), or changes in the task an agent is tackling (Finn et al., 2017). In non-stationary learning problems, the agent may need to adjust its estimates over time as the environment changes.

In the following sections, I discuss different ways that step-sizes are chosen. First, I discuss Stochastic Gradient Descent optimisers originally proposed for scaling the gradients when training large Artificial Neural Nets in the

Supervised Learning setting. Following this, I turn my attention to the step-size adaptation methods that have been proposed in Reinforcement Learning. I will then discuss one focus of this thesis, a step-size adaptation method which I will generalise from supervised learning to policy evaluation.

2.4 Stochastic Gradient Descent Optimisers

Adaptive step size methods that update based on learning progress are ubiquitous in applications of Artificial Neural Nets. *Optimisers* describe a set of approaches designed to scale the gradients in Stochastic Gradient Descent (SGD). Optimisers such as Adagrad (Lydia & Francis, 2019), Adadelta (Zeiler, 2012), RMSPROP (Tieleman & Hinton, 2012), and Adam (Kingma & Ba, 2015) are designed for applications of SGD. These optimisers counteract some challenges of training deep neural nets: for instance, vanishing and exploding gradients. The principles underlying many optimisers are similar. For brevity, I explain one of the most common optimisers currently deployed: Adam.

Adam scales a gradient via two moving averages: an average of the gradient and another of the squared gradient. Each exponential moving average is governed by a weighting, β_1 and β_2 that determines how much to weight recent gradients g_t for an estimate.

$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{2.1}$$

$$n_t \leftarrow \beta_2 n_{t-1} + (1-\beta_2) g_t^2 \tag{2.2}$$

The first and second moment estimates correspond to the mean and variance of the gradient. The update to the weights is then the mean divided by the standard deviation of the gradients.

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \frac{m}{1-\beta_1} / (\sqrt{\frac{n}{1-\beta_2}} + \epsilon) \tag{2.3}$$

While Adam provides a robust way of scaling the gradients for SGD, it is not parameter-free. Adam introduces three additional parameters that are

tuned: two step sizes, β_1 and β_2; and a small constant ϵ that ensures the denominator is always non-zero. While α, β_1, β_2, and ϵ can all be tuned, it has been argued that for the default parameters Adam performs well across a variety of different network structures and learning problems (Kingma & Ba, 2015). Adam is computationally efficient, requiring $O(n)$ memory, where n is the size of the gradient g, and has linear computational complexity $O(n)$. Adam's estimates of the first and second moments can be learned online and incrementally.

The lesson from optimisers is that by scaling updates based on learning progress, the underlying learning (in this case SGD) method can benefit and performance can be improved. This improvement is drawn from two sources: first, the difficulty of choosing a learning rate is reduced; second, weight updates are scaled on a per-feature basis, acting as a form of representation learning.

In this thesis, I focus on Predictive Knowledge agents that learn many GVFs. While optimisers have had great success in deep learning, whether they are suited to applications of Reinforcement Learning without artificial neural nets is not as clear: a topic broached in Chapter 3 through empirical comparisons.

2.5 Step-size Adaptation for Reinforcement Learning

In the following sections, I discuss a variety of step-size adaptation methods that have been developed for the Reinforcement Learning setting. There are approaches to step-size adaptation that are independent of the batch machine learning literature, and designed specifically with RL in mind. Such methods include HL(λ) (Hutter & Legg, 2008), AlphaBound (Dabney & Barto, 2012), and the many examples explored by Dabney (Dabney, 2014). A common hindrance of most of these adaptive learning methods is that they introduce additional meta learning parameters: parameters that govern the process by which the step size is adapted. In this way, adaptive step sizes do not eliminate the influence of learning parameters on performance, but attempt to reduce

sensitivity to these parameters.

2.5.1 HL(λ)

HL is an approach to setting step sizes where the step size is determined by the eligibility traces, and a visitation count of the present state transition. For a state visitation-count c, eligibility traces e, and a discount γ, HL(λ) defines a scalar step size value $\alpha(s_t, s_{t+1}) = \frac{1}{c(s_{t+1}) - \gamma e(s_{t+1})} \frac{c(s_{t+1})}{c(s_t)}$. Intuitively, HL($\lambda$) scales the step size based on the visitation count between two states. The second term $\frac{c(s_{t+1})}{c(s_t)}$ could be interpreted as increasing the step size when s_{t+1} has been visited more frequently than the present state s_t. In such cases, it might be expected that $v(s_{t+1})$ is better estimated than $v(s_t)$ meaning that the TD error should be weighted more, as the bootstrapped error might be more accurate to the true underlying value. The first term $\frac{1}{c(s_{t+1}) - \gamma e(s_{t+1})}$ is an ever-decreasing fraction: as the state visitation increases, the step size decreases. While HL(λ) removes the step size parameter, it requires a state-visitation count c, incurring $O(n)$ memory cost, where n is the number of states. Because of this state visitation count, HL is not well suited for real-world applications where the agent observes stimuli from the environment, rather than states.

2.5.2 AlphaBound

AlphaBound is a step-size adaptation method designed for online learning with function approximation. AlphaBound calculates an upper bound for the step size value to guarantee that divergence does not occur. Using this upper bound as a heuristic, the step size is adapted over time. Given eligibility trace e, discount γ, a function approximator ϕ, and observations o the step sizes are adapted as follows: the initial step size is $\alpha_0 = 1.0$, and subsequent step sizes are $\alpha_t = \min(\alpha_{t-1}, |e_t^\top(\gamma\phi(\mathbf{o}_{t+1}) - \phi(\mathbf{o}_t))|)$. Alpha bound has a linear $O(n)$ computational complexity per step, where n is the number of features in $\phi(o)$, and does not require additional memory.

2.5.3 The step-size adaptation method explored in this thesis: Incremental Delta-Bar-Delta

Having discussed some of the numerous step-size adaptation methods in supervised learning and Reinforcement Learning, let us turn our attention back to the supervised learning setting. In particular, I discuss the adaptation method that I generalise to the TD learning setting: Incremental Delta-bar Delta (IDBD) (Sutton, 1992). IDBD is a step-size adaptation for supervised learning that adapts step sizes on a per-feature basis by performing stochastic meta-descent. Through meta-gradient descent, IDBD minimises the squared error δ^2 with respect to the meta-weights $\boldsymbol{\beta}$, $\frac{\partial \delta^2}{\partial \boldsymbol{\beta}}$; δ is the difference between the target value and the estimated value, and $\boldsymbol{\beta}$ are meta weights that specify the step sizes $\boldsymbol{\alpha} = \exp(\boldsymbol{\beta})$. As IDBD is at its heart stochastic gradient descent, it requires an additional *meta* step-size parameter θ to govern its own meta gradient descent process that adapts the underlying supervised learning process' step sizes. For n features, the inputs are $\mathbf{x} \in \mathbb{R}^n$, the meta-weights are given by $\boldsymbol{\beta} \in \mathbb{R}^n$ and an additional memory vector $\mathbf{h} \in \mathbb{R}^n$. IDBD's gradient steps are scaled by a meta step size $\theta > 0$. The updates for IDBD are as follows:

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \theta \delta \mathbf{x} \mathbf{h} \tag{2.4}$$

$$\boldsymbol{\alpha} \leftarrow \exp(\boldsymbol{\beta}) \tag{2.5}$$

$$\mathbf{h} \leftarrow \mathbf{h} \, \text{relu}(1 - \boldsymbol{\alpha} \mathbf{x}^2) + \boldsymbol{\alpha} \delta \mathbf{x} \tag{2.6}$$

An intuition of IDBD is that features which are correlated with the error should be weighted more, and features that are not correlated with the error, should contribute less to the learned estimate. In this regard, IDBD is solving two problems at once: 1) how to choose step size parameters, and 2) how to weight input features. IDBD is relevant to this thesis for three properties:

1. IDBD is an efficient algorithm that is linear in terms of memory and computation in the number of features.

2. IDBD is an online, incremental algorithm.

3. IDBD adapts step sizes on a per-feature basis.

Because of these properties, IDBD is a suitable candidate for generalisation to TD learning for use in Predictive Knowledge agents where many predictions are learned, and ideally designers are not tasked with extensive parameter tuning and feature selection prior to learning and deployment.

There exist two other works that generalise IDBD to step-size adaptation to value-based RL methods. First, a generalization of nl-IDBD (Koop, 2008) for self play in episodic games, also called nl-IDBD (Bagheri et al., 2016; Thill, 2015). Second, an examination of online step-size adaptation methods for episodic control tasks: SID, NOSID, and AutoSID (Dabney, 2014). The variety of existing approaches reflects the differing motivations and use-cases of the original works. Both bodies of work make different choices when bringing IDBD from supervised learning to bootstrapped RL methods.

One choice is how the gradient of TD error is taken with respect to the learned weights $\nabla_{\mathbf{w}}\delta$. The TD error $\delta = c + \gamma v(\phi(\mathbf{o}')) - v(\phi(\mathbf{o}))$ can be broken into two components:

Target: $c + \gamma v(\phi(\mathbf{o}'))$ what error is calculated with respect to.

Estimate: $v(\phi(\mathbf{o}))$ the value estimate that error is calculated for.

Because the target depends on the value estimate at the following time-step $v(\phi(\mathbf{o}'))$, there is a question of whether the estimate at $\mathbf{o}'$ should be included in the gradient calculation. If included, the gradient is a *residual gradient* (Baird, 1995), or *ordinary-gradient*. If excluded, then the gradient is a *semi-gradient* (Sutton & Barto, 2018). Both existing generalisations to RL take differing approaches: nl-IDBD examines a semi-gradient approach, and Dabney's methods all use an ordinary-gradient. In this thesis, we derive

algorithms for both, and compare their properties to better understand the consequences of each choice.

Another choice is how to construct the objective function: with respect to what value should the step-sizes be modified? One option is to use the one-step TD error, another is to use the λ-return. Multi-step learning methods that use the λ-return enable a more general approach and offer more efficient learning (Bhandari et al., 2018). In some cases, methods that use eligibility traces are susceptible to divergence, particularly in the non-linear function approximation case (Seijen, 2016); however, this is not a general rule. Multi-step methods that utilise eligibility traces can yield improvements over one-step methods in the non-linear function approximation setting (Harb & Precup, 2017). In this thesis, I consider the one-step setting when generalising IDBD to TD learning.

While each existing generalisation of IDBD to RL is distinct, there are some commonalities: all existing generalisations are tested on problems that are episodic, stationary, and fully observable. In contrast, this thesis is concerned with continual learning problems that exhibit both non-stationarity and partially observability. How well existing generalisations of IDBD perform in these conditions remains to be seen. This discussion and comparison is elaborated on in Chapters 3 and 4, where I introduce a new generalisation: TIDBD.

Algorithm	Gradient	Objective	# of Step sizes
SID, NoSID, AutoSID	Ordinary Gradient	λ-return	Scalar
nl-IDBD (TD)	Semi-gradient	λ-return	Vector
TIDBD, AutoTIDBD	Both	δ-error	Both

Table 2.1: IDBD variants for Reinforcement Learning and their defining features.

2.6 Open Questions in Predictive Knowledge

Predictive Knowledge is an open area of study, with a broad frontier of research. In this section, I highlight a few of these open areas of study.

2.6.1 How does an agent learn to predict?

Predictions are at the core of a Predictive Knowledge system. Naturally, an important decision in the design of Predictive Knowledge agents is how an agent learns to make a prediction. In this thesis, I consider predictions phrased as General Value Functions: predictions of the accumulation of a stimulus. Choosing how to learn a prediction in this case is choosing 1) the learning method (e.g., gradient TD vs. TDRC) and 2) the hyperparameters used in learning (e.g., should the step size be smaller).

There is no total ordering of learning methods. Different TD learning methods will perform better than others or worse depending on many factors, including the choice of environment; there is no *best* TD learning method (Ghiassian & Sutton, 2021)—there is a trade-off between different algorithms.

Common amongst online TD learning methods are their hyperparameters. TD learning methods are sensitive to the selection of learning parameters: given the same underlying conditions, two instantiations of the same learning method may perform differently given the selection of their learning parameters. In the Predictive Knowledge setting where an agent may pose and modify its own predictive questions without designer input, it is impractical to carefully perform a parameter study for each of the many thousands of predictions an agent might make. For this reason, developing algorithms that are well suited to a broad variety of learning problems, and are relatively insensitive to hyperparameters is critical for Predictive Knowledge agents.

2.6.2 How does an agent use predictions?

Having learned to predict aspects of their environment, how does an agent use these predictions to improve decision-making? One way an agent can use their estimates is as predictive input features. In addition to observations, an

agent uses its learned estimates to choose which action to take. In this way, the *agent-state*—the state of the world from the agent's perspective—or the agent's perception—how an agent processes inputs into features—is hopefully better than operating using the environmental inputs alone.

The most common use of GVFs is not as input features, but as auxiliary tasks (Jaderberg et al., 2017). An auxiliary task is a learning objective that an agent must solve in addition to its main task. For example, in addition to learning to solve a particular control problem, an agent must also learn to solve a prediction task. The central motivation of auxiliary tasks is that when learning to solve sparse reward problems with a Deep Neural Net, it might be beneficial to find features that enable the prediction task to be solved as an intermediary sub-task that helps the agent learn in the absence of reward.

2.6.3 How does an agent structure its predictions?

How does an agent effectively organise its predictions? In the most simple setting, predictions can be simply about the observations available to an agent (Edwards, Dawson, Hebert, Sherstan, et al., 2016). Or, predictions can be hierarchically organised, so that the signal of interest of a prediction is from another estimate (Ring, 2021; Schlegel et al., 2021). Many existing applications of Predictive Knowledge use a fixed set of predictions that are organised as a single layer (Edwards, Dawson, Hebert, Sherstan, et al., 2016; Günther et al., 2020). Hierarchical collections of predictions have been suggested to enable agents to learn more abstract aspects of their environment (Ring, 2021). How an agent might structure large collections of predictions, including how predictions might be hierarchically related, is an open area of research in Predictive Knowledge agents.

2.6.4 How does an agent determine what to learn about?

Given an agent's inputs, possible behaviours, and temporal horizons, there are infinitely many possible predictions that an agent might choose to make about the world. A core challenge is then determining of all the possible predictions an agent might make, which will best inform decision-making. In

most applications, what a GVF is about is determined by a human designer (Dalrymple et al., 2020; Edwards, Dawson, Hebert, Sherstan, et al., 2016; Kearney et al., 2018; Modayil & Sutton, 2014; Sutton et al., 2011).

If an agent is to use predictions to form the basis of their conceptualisation of the world, requiring a human designer to choose what to predict is a serious limitation. If an agent makes many thousands of estimates, having a human designer select each is impractical: it is a constraint that limits the development of Predictive Knowledge.

An alternative is for agents to autonomously specify what to learn. What a General Value Function is about is determined by meta-parameters that specify the signal of interest being predicted, the horizon over which the prediction is being made, and the behaviour that the agent undertakes to realise the prediction. By enabling an agent to select or modify these meta-parameters, an agent can choose which aspects of the environment to model.

One way to enable an agent to select their parameters is via generate and test (Schlegel et al., 2018). Recently, work has explored how through meta-descent an agent might choose to modify the parameters that specify auxiliary tasks (Veeriah et al., 2019). These are both relatively new lines of research; how an agent selects what to learn remains a largely open problem.

Chapter 3

How should an agent learn? Adapting Step Sizes by Stochastic Meta-descent

Contributions of this chapter.

1. A generalisation of Incremental Delta-Bar-Delta and AutoStep to on-policy TD learning.
2. An empirical comparison of AutoTIDBD with existing step-size adaptation methods in both a synthetic stationary problem and multiple real-world non-stationary problems.
3. A comparison of prediction error and initialisation sensitivity across different step-size adaptation strategies.

I begin by examining how an agent chooses to learn by adapting the parameters that govern learning through experience. Central to many machine learning methods are their *learning parameters* or hyperparameters: values that modify the underlying learning process an agent uses. How an agent learns an estimate–and the agent's performance on a given task–is in part determined by the selection of learning parameter values. To achieve acceptable performance on a given task, designers may systematically search through the space of possible values and examine performance over multiple independent trials (Bergstra et al., 2011; Bergstra & Bengio, 2012).

For long-lived agents learning many independent predictions, such parameter searches are infeasible. How might an agent determine how to learn through experience?

In this chapter, I will consider how an agent might adapt a particular kind of learning parameter, its step-size, during the course of learning. In particular, I generalize the Incremental Delta-Bar-Delta (IDBD) (Sutton, 1992) method of learning step sizes to Temporal-difference (TD) learning, that I call TIDBD.

TIDBD is an algorithm for learning the relevance of features of a linear function approximator by adapting the individual step sizes for each feature. Using TIDBD, a feature that is relevant to the task will be assigned a large step-size, and an irrelevant feature will be assigned a correspondingly small step size. By modifying step sizes for each feature independently, TIDBD can be seen as a rudimentary form of representation learning. How much an update influences each weight depends on the relevance of its corresponding feature.

3.1 Gradient Methods for Meta-learning

The performance of Machine Learning (ML) methods depends greatly on the inputs they use, and how those inputs are transformed into a feature vector. Moreover, the choice of input features can be the difference between a successful application and one which is unable to learn. One way to find useful features is to hand-select them using an expert's knowledge of the problem (Shapiro & Stockman, 2001). The effectiveness of hand-constructed features is limited, as their design requires substantial knowledge of both the environment and the problem being solved: knowledge a designer may not have prior to the deployment of an agent in a real-world setting in which the agent is expected to learn continually. Moreover, features that are appropriate for a given task are not necessarily transferable to different environments and problems. For each new problem and environment, the engineer must assess and possibly re-design the representation used. Deciding how to process signals from the

environment can itself be learned through a second-order learning process—sometimes described as representation learning, or more broadly as *learning to learn* (Andrychowicz et al., 2016) or *meta-learning* (Finn et al., 2017). Meta-learning methods learn to modify the inputs or the parameters of the underlying machine learning method by a higher-order learning process. Meta-learning can be used to perform representation learning by constructing new features, or learning to shape an existing representation by weighting given features (Sutton, 1992; Veeriah et al., 2017). In this chapter, I focus on systems which perform meta-learning to weight a given set of inputs by identifying relevant features.

The simplest method of representation learning is to learn the relevance of given features. Whether constructed by hand or learned, the features available to an agent will not be equally relevant to the task at hand. Some features will be more relevant, and it is desirable for an agent to generalize over these relevant features more than others. By learning feature relevance, a system can weight the influence of input features on the model being learnt.

The problem of identifying relevant features through meta-learning has roots in both animal and human learning. Humans and animals learn to discern which aspects of the environment are relevant to the task at hand. Work in neuroscience has assessed how humans perform representation learning by identifying relevant stimuli (Wilson & Niv, 2012); in cognitive science, research has examined how children can generalize from just a few examples by forming appropriate inductive biases (Colunga & Smith, 2005). Animals learn over which features to generalize their learning to new examples–they learn the salience of the signals. In doing so, humans and animals are performing representation learning by identifying the relevance of stimuli.

One method of assigning feature relevance in Machine Learning is through adapting many step sizes: the hyper parameters that scale updates made to a learned model. By assigning step sizes on a per-feature basis, weight updates can be scaled based on the relevance of input features large step sizes may be assigned to relevant features and small step sizes to irrelevant features (Sutton, 1992). In this chapter, I focus on methods which use linear function-

approximation. In the linear setting, there is a single step size per feature and feature-relevance directly corresponds with step sizes. In particular, I explore learning feature relevance through Stochastic Meta-descent (SMD): a form of gradient descent which takes the gradient of the error on the main-task with respect to specific hyper-parameters that govern learning.

3.2 Stochastic Meta-descent for Learning Feature Relevance

SMD was first introduced for online learning in the linear case as Incremental Delta-Bar-Delta (IDBD) (Sutton, 1992) which was later extended to non-linear mappings (Schraudolph, 1999). Most recently, SMD has been used for temporal predictions and sequential problems, such as MAML (Finn et al., 2017): a method for finding initial weight settings to enable better generalisation in settings with multiple tasks. Xu et al. use SMD to adapt the return of a reinforcement learning problem (Xu et al., 2018), and Crossprop uses SMD to learn weightings of inputs to learn representations which generalize across tasks (Veeriah et al., 2017). Outside of machine learning, IDBD has been extended to a biologically plausible version for modelling neural metaplasticity (Schweighofer & Arbib, 1998).

3.2.1 Incremental Delta-Bar-Delta

As introduced in section 2.5.3 IDBD (Sutton, 1992) is a meta-learning algorithm which learns a bias through experience by maintaining a vector of learned step sizes. An intuition behind IDBD is that features which are correlated with the prediction task (and error) should have larger step sizes, while features which are irrelevant to the prediction task should have smaller step sizes.

IDBD is a meta-learning method for supervised learning, where weights $\mathbf{w} \in \mathbb{R}^n$ are adapted by stochastic gradient descent to estimate $\hat{y} \in \mathbb{R}$, such that the dot-product of the observations and the weight vector produce an estimate $y = \mathbf{w}^\top \mathbf{o}$. IDBD amends traditional supervised learning by learning to weight features by adapting a vector of step sizes $\boldsymbol{\alpha}$, such that each weight $\mathbf{w}_i$ has

its own step size $\boldsymbol{\alpha}_i$. IDBD learns many step sizes online and incrementally by performing stochastic meta-descent over a vector of meta-weights $\boldsymbol{\beta}$ that specify the step sizes $\boldsymbol{\alpha}$. On each time-step t the vector of step-sizes is given by $\boldsymbol{\alpha} = \exp(\boldsymbol{\beta})$. By exponentiating the meta-weight vector $\boldsymbol{\beta}$ to produce a step size $\boldsymbol{\alpha}$, a linear step in the meta-weight vector $\boldsymbol{\beta}$ produces a geometric step in $\boldsymbol{\alpha}$ and ensures all step sizes $\boldsymbol{\alpha}$ are positive.

Algorithm 2 Incremental Delta-Bar-Delta.

initialise:
Initialise vectors $\mathbf{h}$, $\boldsymbol{\beta}$, and $\mathbf{w}$ of size n number of features. Choose a meta step size $0 < \theta$.

repeat For each observation $\mathbf{o}$ and target $\hat{y}$
$y \leftarrow \mathbf{w}^\top \mathbf{o}$
$\delta \leftarrow \hat{y} - y$
repeat For $i = 1, 2, \cdots, n$
$\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_i + \theta \delta \mathbf{o}_i \mathbf{h}_i$
$\boldsymbol{\alpha}_i \leftarrow e^{\boldsymbol{\beta}_i}$
$\mathbf{w}_i \leftarrow \mathbf{w}_i + \boldsymbol{\alpha}_i \delta \mathbf{o}_i$
$\mathbf{h}_i \leftarrow \mathbf{h}_i \text{ relu}(1 - \boldsymbol{\alpha}_i \mathbf{o}_i^2) + \boldsymbol{\alpha}_i \delta \mathbf{o}_i$
until termination

While IDBD is presented alongside the underlying gradient descent learning mechanism in Algorithm 2, IDBD is truly the update and maintenance of $\boldsymbol{\beta}$, $\boldsymbol{\alpha}$, and $\mathbf{h}$. In this sense, IDBD is a meta-learning method that is distinct and separate from the underlying learning method that learns the resulting model to estimate $\hat{y}$; however, the order of IDBD's updates in relation to the underlying learning updates is important.

A possible criticism of IDBD is that it is only abstracting the problem of setting step size values to a higher level. Although IDBD learns the step size parameter, it also introduces the meta step size θ that governs the learning of the step sizes. IDBD is still an improvement over ordinary supervised learning: tuned IDBD outperforms methods which do not adapt their bias (Sutton, 1992). In addition, extensions of IDBD, including AutoStep (Mahmood et al., 2012), NOSID, and AUTOSID (Dabney, 2014) have had success in translating IDBD to methods that are relatively invariant to the setting of the meta step size θ.

Another possible criticism of IDBD is that of stability. One might hypothesise that several consecutive weight updates to $\mathbf{w}_i$ are in the same direction, then $\mathbf{h}_i$ will grow correspondingly large. As $\mathbf{h}_i$ grows, $\boldsymbol{\beta}_i$ grows, and $\boldsymbol{\alpha}_i$ increases geometrically in size. This could lead to instability, as a single large update could produce a large step-size $\boldsymbol{\alpha}_i$ and lead to divergence. To prevent this, Sutton suggests that updates to the meta-weight vector $\boldsymbol{\beta}$ are limited ± 2 and that each step size $\boldsymbol{\alpha}_i$ is limited to some maximum value (Sutton, 1992); however, it is also noted that clipping was not necessary to achieve empirical results as originally introduced. Moreover, subsequent work has demonstrated that IDBD-based methods can be stable across prediction and control problems (Dabney, 2014; Mahmood et al., 2012).

IDBD holds substantial promise as a step-size adaptation method for applications of Predictive Knowledge. IDBD's memory complexity and computational complexity are both linear in the number of features, making it ideal for scenarios where multiple independent learners are operating in parallel. For each additional prediction an agent makes, the computational cost of using IDBD scales linearly with the number of features. Moreover, IDBD is an incremental learning method, making it amenable to online continual learning. For these reasons, IDBD is an algorithm of interest for this thesis, and I seek to generalise IDBD to TD learning.

3.2.2 TIDBD: TD Incremental Delta-Bar-Delta

In this section, I generalise IDBD to TD learning, which I call TIDBD. TIDBD updates the meta-weights $\boldsymbol{\beta}$ that define the step-size $\boldsymbol{\alpha} = \exp(\boldsymbol{\beta})$ by minimizing the gradient of the squared one-step TD error $\frac{\partial \delta_t^2}{\partial \boldsymbol{\beta}_{i,t}}$. The meta-weights are updated by taking a gradient step with a step-size of θ to minimise the squared error on a particular example at time-step t for each individual meta weight indexed with i and j:

$$\begin{aligned}\boldsymbol{\beta}_{i,t+1} &= \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\frac{\partial\delta_t^2}{\partial\boldsymbol{\beta}_i} \\ &= \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\sum_j \frac{\partial\delta_t^2}{\partial\mathbf{w}_{j,t}}\frac{\partial\mathbf{w}_{j,t}}{\partial\boldsymbol{\beta}_i}\end{aligned} \tag{3.1}$$

The small scalar value θ is an additional step size value that determines the rate at which the meta-weights are updated by gradient descent. To approximate $\sum_j \frac{\partial\delta_t^2}{\partial\mathbf{w}_{j,t}}\frac{\delta\mathbf{w}_{j,t}}{\delta\boldsymbol{\beta}_i}$, assume that $\frac{\partial\mathbf{w}_{j,t}}{\partial\boldsymbol{\beta}_i} \approx 0$ where $i \neq j$. This assumes the effect of changing the step size for a particular weight will predominantly be on the weight itself; effects on other weights will be small. With this assumption, the updates to the meta-weights may be simplified as follows:

$$\boldsymbol{\beta}_{i,t+1} \approx \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\frac{\partial\delta_t^2}{\partial\mathbf{w}_{i,t}}\frac{\partial\mathbf{w}_{i,t}}{\partial\boldsymbol{\beta}_i} \tag{3.2}$$

To further simplify Equation 3.2, one must decide how to take the gradient with respect to the TD error δ. Recall, that the TD error $\delta = c_t + \gamma v(\phi(\mathbf{o}_{t+1})) - v(\phi(\mathbf{o}_t))$ depends on the signal of interest being predicted c, and the discounted predicted value of the future state $\gamma v(\phi(\mathbf{o}_{t+1}))$, resulting in a biased gradient. Bias is introduced by the dependence on the learned weights $\mathbf{w}_t$ in the calculation of $v(\phi(\mathbf{o}_{t+1}))$. For this reason, taking the gradient of a bootstrapped estimate is not true gradient descent (Barnard, 1993).

There are two choices: performing gradient descent using the full, biased gradient, or using a semi-gradient method. Semi-gradient methods do not use the estimate of the return at state $\phi(\mathbf{o}_{t+1})$ in the gradient calculation. In this chapter, I show the derivation for both choices. In the following section, the performance of each method is compared on a series of synthetic and robotic prediction tasks.

3.2.3 Derivation of semi-gradient method

In this section, I derive IDBD for TD(λ) using a semi-gradient, which I call semi-gradient TIDBD. In the linear case, the semi-gradient for $\delta\nabla_{\mathbf{w}}$ is $[c_{t+1} + \gamma v(\phi(\mathbf{o}_{t+1})) - v(\phi(\mathbf{o}_t))]\nabla_{\mathbf{w}} = -\phi(\mathbf{o}_t)$, as the influence of $v(\phi(\mathbf{o}_{t+1}))$ is not included in the gradient calculation. For the following generalization, we

use two subscripts: i, which describes the index in a vector; and t, which describes the time-step being referenced. Using the semi-gradient, $\frac{1}{2}\frac{\partial \delta_t^2}{\partial \mathbf{w}_{i,t}}$ can be simplified as follows:

$$\begin{aligned} -\frac{1}{2}\frac{\partial \delta_t^2}{\partial \mathbf{w}_{i,t}} &= -\delta_t \frac{\partial \delta_t}{\partial \mathbf{w}_{i,t}} \\ &= -\delta_t \frac{\partial}{\partial \mathbf{w}_{i,t}}[-v(\phi(\mathbf{o}_t))] \\ &= \delta_t \phi_i(\mathbf{o}_t) \end{aligned} \tag{3.3}$$

This simplification may then be substituted back into the $\boldsymbol{\beta}$ update rule in Equation 3.2 as so:

$$\boldsymbol{\beta}_{i,t+1} \approx \boldsymbol{\beta}_{i,t} + \theta \delta_t \phi_i(\mathbf{o}_t) \frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i} \tag{3.4}$$

The meta-weight $\boldsymbol{\beta}$'s update can then be completed by defining an additional memory vector $\mathbf{h}$, where $\mathbf{h}$ is an approximation, $\mathbf{h}_{i,t+1} \approx \frac{\partial \mathbf{w}_{i,t+1}}{\partial \boldsymbol{\beta}_i}$. These simplifications result in the following update rule:

$$\boldsymbol{\beta}_{i,t+1} \approx \boldsymbol{\beta}_{i,t} + \theta \delta_t \phi_i(\mathbf{o}_t) \mathbf{h}_{i,t} \tag{3.5}$$

To approximate $\frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i}$ incrementally in $\mathbf{h}_{i,t}$, I describe the update rule in terms of $\mathbf{w}$'s TD update: $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \mathbf{z}$, where $\mathbf{z}$ are the eligibility traces of a TD(λ) update.

$$\begin{aligned} \mathbf{h}_{i,t+1} &\approx \frac{\partial \mathbf{w}_{i,t+1}}{\partial \boldsymbol{\beta}_i} \\ &= \frac{\partial}{\partial \boldsymbol{\beta}_i}[\mathbf{w}_{i,t} + \boldsymbol{\alpha}_{i,t} \delta_t \mathbf{z}_{i,t}] \\ &= \mathbf{h}_{i,t} + \frac{\partial}{\partial \boldsymbol{\beta}_i}[\boldsymbol{\alpha}_{i,t} \delta_t \mathbf{z}_{i,t}] \\ &= \mathbf{h}_{i,t} + \frac{\partial \boldsymbol{\alpha}_{i,t}}{\partial \boldsymbol{\beta}_i} \delta_t \mathbf{z}_{i,t} + \frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i} \boldsymbol{\alpha}_{i,t} \mathbf{z}_{i,t} + \frac{\partial \mathbf{z}_{i,t}}{\partial \boldsymbol{\beta}_i} \boldsymbol{\alpha}_{i,t} \delta_t \end{aligned} \tag{3.6}$$

Using the product rule to simplify (3.6) leaves three remaining partial derivatives to simplify. First, let us examine $\frac{\partial \alpha_{i,t}}{\partial \boldsymbol{\beta}_i}$. The step-sizes are defined as $\alpha = e^{\boldsymbol{\beta}}$, and the partial derivative may be taken as follows.

$$\begin{aligned} \frac{\partial \boldsymbol{\alpha}_{i,t}}{\partial \boldsymbol{\beta}_i} &= \frac{\partial e^{\boldsymbol{\beta}_{i,t}}}{\partial \boldsymbol{\beta}_i} \\ &= e^{\boldsymbol{\beta}_{i,t}} \end{aligned} \tag{3.7}$$

Now, let us simplify $\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i}$.

$$\begin{aligned}
\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i} &= -\frac{\partial}{\partial \boldsymbol{\beta}_i}[v(\phi(\mathbf{o}_t))] \\
&= -\frac{\partial}{\partial \boldsymbol{\beta}_i} \sum_j \mathbf{w}_{j,t}\phi_j(\mathbf{o}_t) \\
&\approx -\frac{\partial}{\partial \boldsymbol{\beta}_i}[\mathbf{w}_{i,t}\phi_i(\mathbf{o}_t)] = -\mathbf{h}_{i,t}\phi_i(\mathbf{o}_t)
\end{aligned} \tag{3.8}$$

Let us simplify the final term $\frac{\partial \mathbf{z}_{i,t+1}}{\partial \boldsymbol{\beta}_i}$ as follows:

$$\frac{\partial \mathbf{z}_{i,t+1}}{\partial \boldsymbol{\beta}_i} = \frac{\partial}{\partial \boldsymbol{\beta}_i}[\gamma\lambda\mathbf{z}_{i,t} + \phi_i(\mathbf{o}_t)] = \gamma\lambda\frac{\partial \mathbf{z}_{i,t}}{\partial \boldsymbol{\beta}_i} = 0 \tag{3.9}$$

Equation (3.9) shows a decaying trace of the gradient of the eligibility traces. Since the gradient is 0, this value will always be 0. With Equations 3.7, 3.8, and 3.9, the update for $\mathbf{h}$ may be simplified as follows.

$$\begin{aligned}
\mathbf{h}_{i,t+1} &\approx \mathbf{h}_{i,t} + \frac{\partial \boldsymbol{\alpha}_t}{\partial \boldsymbol{\beta}}\delta_t\mathbf{z}_t + \frac{\partial \delta_t}{\partial \boldsymbol{\beta}}\boldsymbol{\alpha}_t\mathbf{z}_t + \frac{\partial \mathbf{z}_t}{\partial \boldsymbol{\beta}}\boldsymbol{\alpha}_t\delta_t \\
&\approx \mathbf{h}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}}\delta_t\mathbf{z}_{i,t} - e^{\boldsymbol{\beta}_{i,t+1}}\phi_i(\mathbf{o}_t)\mathbf{z}_{i,t}\mathbf{h}_{i,t} \\
&= \mathbf{h}_{i,t}[1 - \boldsymbol{\alpha}_{t+1}\phi_i(\mathbf{o}_t)\mathbf{z}_{i,t}] + \boldsymbol{\alpha}_{i,t+1}\delta_t\mathbf{z}_{i,t}
\end{aligned} \tag{3.10}$$

The memory vector $\mathbf{h}$ is a decaying trace of the cumulative sum of updates for each feature. The learning update on the meta-weights $\boldsymbol{\beta}$ is proportional to the current error, and a trace of recent weight updates $\delta\phi\mathbf{h}$, and is scaled by the magnitude of recent weight updates $\mathbf{h}$. Intuitively, if many updates are correlated, and $\mathbf{h}$ is large, then it would have been a more efficient use of experience to make a larger step. The decay term, $1-\boldsymbol{\alpha}_{t+1}\phi_i(\mathbf{o}_t)\mathbf{z}_{i,t}$ determines the rate at which historical weight updates contribute to the current estimate of $\mathbf{h}$. I make the assumption that the decay term should not change the direction of $\mathbf{h}$, so the update is positively bounded, denoted with $\texttt{relu}(1-\boldsymbol{\alpha}_{t+1}\phi_i(\mathbf{o}_t)\mathbf{z}_{i,t})$. Having completed the update for $\boldsymbol{\beta}$ and $\mathbf{h}$, the derivation of semi-gradient TIDBD is completed, as shown in Algorithm 3 below.

Following the original derivation of IDBD (Sutton, 1992), ϕ may produce a real-valued state vector. However, in this thesis all TIDBD experiments use a binary feature vector to produce the agent-state $\phi(\mathbf{o})$.

Algorithm 3 Semi-gradient TIDBD.

initialise:
vectors $\mathbf{h} \in 0^n$, $\mathbf{z} \in 0^n$, and both $\mathbf{w} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$ as desired; scalar $\theta > 0$; observe environment $\mathbf{o}$.

repeat For each observation $\mathbf{o}'$, cumulant c:

$\delta \leftarrow c + \gamma \mathbf{w}^\top \phi(\mathbf{o}') - \mathbf{w}^\top \phi(\mathbf{o})$

for element $i = 1, 2, \cdots, n$ **do**

$\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_i + \theta \delta \phi_i(\mathbf{o}) \mathbf{h}_i$

$\boldsymbol{\alpha}_i \leftarrow e^{\boldsymbol{\beta}_i}$

$\mathbf{z}_i \leftarrow \mathbf{z}_i \gamma \lambda + \phi_i(\mathbf{o})$

$\mathbf{w}_i \leftarrow \mathbf{w}_i + \boldsymbol{\alpha}_i \delta \mathbf{z}_i$

$\mathbf{h}_i \leftarrow \mathbf{h}_i \text{ relu}(1 - \boldsymbol{\alpha}_i \phi_i(\mathbf{o}) \mathbf{z}_i) + \boldsymbol{\alpha}_i \delta \mathbf{z}_i$

$\mathbf{o} \leftarrow \mathbf{o}'$

One may note that semi-gradient TIDBD is similar to the original IDBD formulation (Algorithm 2). The most notable change is that TIDBD's $\mathbf{h}$ trace is now modulated by not just the active features ϕ, but also the eligibility traces $\mathbf{z}$. This means that while the updates to step sizes will be limited to currently active features $\phi(\mathbf{o})$, the trace of recent weight updates include discounted past activations in $\mathbf{z}$.

3.2.4 Derivation of ordinary-gradient method

In the previous section, I derived IDBD for TD with a semi-gradient. In this section, I present an alternative, and derive TIDBD as stochastic meta-descent using the ordinary-gradient. The derivation of TIDBD is started by describing the update rule for $\boldsymbol{\beta}$—the meta-weight vector that defines the step-size. In the ordinary-gradient case, the gradient is calculated with respect to the estimated value, and the target. That is, $\frac{\partial \delta_t}{\partial \mathbf{w}_{i,t}} = \frac{\partial}{\partial \mathbf{w}_{i,t}} [c_{t+1} + \gamma \mathbf{w}_t^\top \phi(\mathbf{o}_{t+1}) - \mathbf{w}_t^\top \phi(\mathbf{o}_t)]$. In the ordinary-gradient setting, the update to $\boldsymbol{\beta}$ is simplified as follows:

$$
\begin{aligned}
\boldsymbol{\beta}_{i,t+1} &\approx \boldsymbol{\beta}_{i,t} - \theta \delta_t \frac{\partial \delta_t}{\partial \mathbf{w}_{i,t}} \frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i} \\
&= \boldsymbol{\beta}_{i,t} - \theta \delta_t \frac{\partial [c_{t+1} + \gamma \mathbf{w}_{i,t}^\top \phi_i(\mathbf{o}_{t+1}) - \mathbf{w}_{i,t}^\top \phi_i(\mathbf{o}_t)]}{\partial \mathbf{w}_{i,t}} \frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i} \\
&= \boldsymbol{\beta}_{i,t} - \theta \delta_t [\gamma \phi_i(\mathbf{o}_{t+1}) - \phi_i(\mathbf{o}_t)] \frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i} \\
&= \boldsymbol{\beta}_{i,t} - \theta \delta_t [\gamma \phi_i(\mathbf{o}_{t+1}) - \phi_i(\mathbf{o}_t)] \mathbf{h}_{i,t}
\end{aligned}
\tag{3.11}
$$

As was the case for semi-gradient TIDBD, $\boldsymbol{\beta}$'s update is completed by defining an additional memory vector $\mathbf{h}_i$ approximating $\frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i}$. Again, $\mathbf{h}$ is approximated recursively:

$$\begin{aligned}
\mathbf{h}_{i,t+1} &\approx \frac{\partial \mathbf{w}_{i,t+1}}{\partial \boldsymbol{\beta}_i} \\
&= \frac{\partial [\mathbf{w}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}} \delta_t \mathbf{z}_{i,t}]}{\partial \boldsymbol{\beta}_i} \\
&= \mathbf{h}_{i,t} + \frac{\partial e^{\boldsymbol{\beta}_{i,t+1}}}{\partial \boldsymbol{\beta}_i} \delta_t \mathbf{z}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}} \frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i} \mathbf{z}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}} \frac{\partial \mathbf{z}_{i,t}}{\partial \boldsymbol{\beta}_i} \delta_{i,t}
\end{aligned} \tag{3.12}$$

The simplification of Equation 3.12 follows the same pattern as the semi-gradient simplification. The only differentiating term separating the semi-gradient and ordinary-gradient method is $\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i}$, as the gradient of δ in the ordinary-gradient case includes both the target and the value estimate. The remaining $\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i}$ may be simplified as follows:

$$\begin{aligned}
\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i} &= \frac{\partial}{\partial \boldsymbol{\beta}_i} [c_{t+1} + \gamma \mathbf{w}_t^\top \phi(\mathbf{o}_{t+1}) - \mathbf{w}_t^\top \phi(\mathbf{o}_t)] \\
&= \frac{\partial}{\partial \boldsymbol{\beta}_i} [\sum_j c_{t+1} + \gamma \mathbf{w}_{j,t} \phi_j(t+1) - \mathbf{w}_{j,t} \phi_j(t)] \\
&\approx \frac{\partial}{\partial \boldsymbol{\beta}_i} [c_{t+1} + \gamma \mathbf{w}_{i,t} \phi_i(\mathbf{o}_{t+1}) - \mathbf{w}_{i,t} \phi_i(\mathbf{o}_t)] \\
&= \gamma \mathbf{h}_{i,t} \phi_i(\mathbf{o}_{t+1}) - \mathbf{h}_{i,t} \phi_i(\mathbf{o}_t)
\end{aligned} \tag{3.13}$$

The update rule for $\mathbf{h}$ (3.12) may be completed by substituting in $\frac{\partial \delta_t}{\partial \boldsymbol{\beta}_i}$ defined above in Equation 3.13, with the simplifications defined in the preceding semi-gradient generalisation for $\frac{\partial \mathbf{z}_{i,t}}{\partial \boldsymbol{\beta}_i}$ in Equation 3.9 and $\frac{\partial e^{\boldsymbol{\beta}_{i,t+1}}}{\partial \boldsymbol{\beta}_i}$ in Equation 3.7.

$$\begin{aligned}
\mathbf{h}_{i,t+1} &\approx \mathbf{h}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}} \delta_t \mathbf{z}_{i,t} + e^{\boldsymbol{\beta}_{i,t+1}} [\gamma \mathbf{h}_i \phi_i(\mathbf{o}_{t+1}) - \mathbf{h}_i \phi_i(\mathbf{o}_t)] \mathbf{z}_{i,t} \\
&= \mathbf{h}_{i,t} [1 + \boldsymbol{\alpha}_{i,t+1} \mathbf{z}_{i,t} [\gamma \phi_i(\mathbf{o}_{t+1}) - \phi_i(\mathbf{o}_t)]] + \boldsymbol{\alpha}_{i,t+1} \delta_t \mathbf{z}_{i,t}
\end{aligned} \tag{3.14}$$

Having defined the updates for $\mathbf{h}$ in Equation 3.14 and $\boldsymbol{\beta}$ in Equation 3.11, ordinary-gradient TIDBD may be implemented as shown in Algorithm 4.

Algorithm 4 Ordinary Gradient TIDBD

initialise:

Vectors $\mathbf{h} \in 0^n$, $\mathbf{z} \in 0^n$, and both $\mathbf{w} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$ as desired; A scalar $\theta > 0$; observe environment $\mathbf{o} \in \mathbb{R}^n$.

repeat For each observation $\mathbf{o}'$ and cumulant c:

$\delta \leftarrow c + \gamma \mathbf{w}^\top \phi(\mathbf{o}') - \mathbf{w}^\top \phi(\mathbf{o})$

for For element $i = 1, 2, \cdots, n$ **do**

$\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_i - \theta \delta [\gamma \phi(\mathbf{o}') - \phi(\mathbf{o})] \mathbf{h}_i$

$\boldsymbol{\alpha}_i \leftarrow e^{\boldsymbol{\beta}_i}$

$\mathbf{z}_i \leftarrow \mathbf{z}_i \gamma \lambda + \phi_i(\mathbf{o})$

$\mathbf{w}_i \leftarrow \mathbf{w}_i + \boldsymbol{\alpha}_i \delta \mathbf{z}_i$

$\mathbf{h}_i \leftarrow \mathbf{h}_i \text{ relu}(1 + \boldsymbol{\alpha}_i \mathbf{z}_i [\gamma \phi_i(\mathbf{o}') - \phi_i(\mathbf{o})]) + \boldsymbol{\alpha}_i \delta \mathbf{z}_i$

$\mathbf{o} \leftarrow \mathbf{o}'$

3.3 Does a Single, Shared Step Size by TIDBD Outperform Ordinary TD?

I now compare both semi-gradient and ordinary-gradient TIDBD algorithms to TD with a fixed step-size. As many aspects of both semi-gradient and ordinary-gradient TIDBD are the same, I refer to them collectively as TIDBD. Although TIDBD's strength is in its ability to tune step sizes on a per-feature basis, I first assess the ability of TIDBD to improve upon traditional TD prediction in a setting with a shared step-size. By examining the simpler case where TIDBD adapts a single step size, it is possible to examine what performance benefits we can attribute to step size adaptation via TIDBD separate from the advantages of learning feature relevance.

To use TIDBD, an initial step size must be chosen by initialising the meta-weights $\alpha_0 = e^{\boldsymbol{\beta}_0}$, and a meta step size θ must be chosen to manage learning of weights. It is desirable for TIDBD to perform as well as or better than TD with a fixed and tuned step size over a broad variety of initial values β_0, and θ: that TIDBD performs as well as TD learner that has been tuned, without requiring much tuning itself.

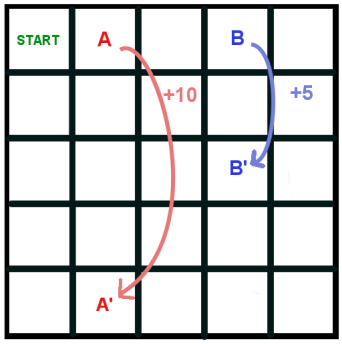
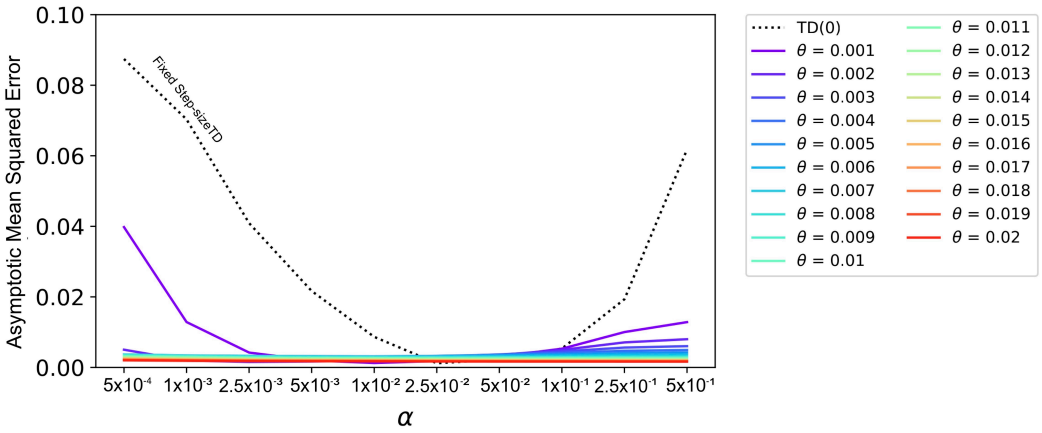


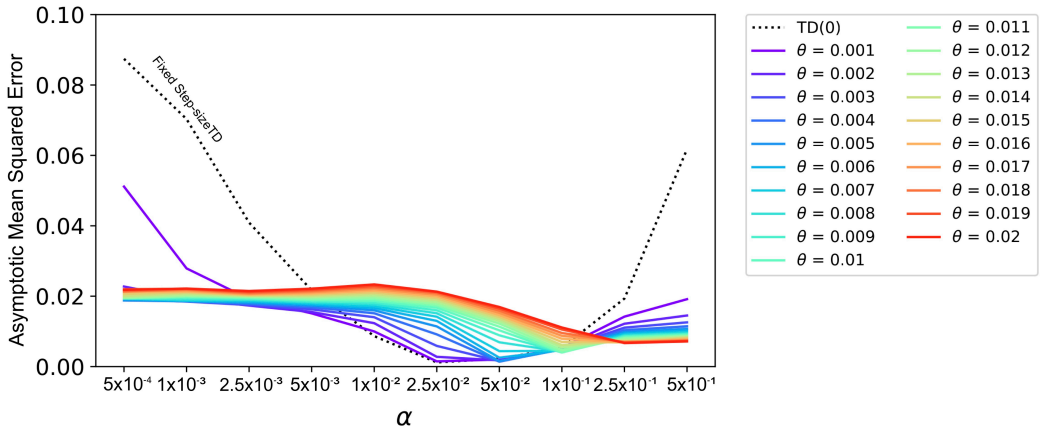
Figure 3.1: Gridworld Problem.

3.3.1 Grid world

I now compare ordinary-gradient and semi-gradient TIDBD to fixed step size TD on a synthetic prediction task. I created a prediction task by generating a Markov Reward Process from a grid-world problem originally described in (Sutton & Barto, 2018) (depicted in Figure 3.1). Each tile in the 5 $\times$ 5 grid-world represents a state. The state transitions are the four cardinal directions—north, south, east, and west—chosen by an equiprobable random policy. Transitions which would leave the grid resulted in staying in the same state and a reward of -1. Regardless of the action selected by the agent in state A or B, it transitions to states A' and B' respectively with a probability of 1. A transition from A to A' yields a reward of 10 and a transition from B to B' yields a reward of 5. All other transitions receive a reward of 0. The agent's starting state is the top left-hand corner. Each independent run consists of the equiprobable random policy acting for 30000 time-steps. All reported results are averaged over the last 30000 steps of 30 independent trials.



(a) Semi-gradient TIDBD.



(b) Ordinary-gradient TIDBD.

Figure 3.2: Parameter study of semi and ordinary-gradient TIDBD. Static step size TD in black. The x-axis denotes the initial step-size all learners use. In the case of TD(0), this step-size never updates. For both Semi and Ordinary-gradient TIDBD, the step-size is adapted online during the experiment with a meta step-size value denoted by the colour of the line, and described in the legend.

In this experiment, the performance of both semi-gradient and ordinary-gradient TIDBD are compared with a TD learner for a variety of initial step-size settings. Ordinary TD, semi-gradient TIDBD, and ordinary-gradient TIDBD were compared with initial step sizes α_0 distributed between 0.0005 and 0.5. For all prediction methods, I used $\lambda = 0$ and $\gamma = 0.99$. For TIDBD, the parameter sweep included 21 different meta step-size θ values equally distributed between 0 and 0.2—the range for which IDBD was originally compared over (Sutton, 1992). When $\theta = 0$, the initial step size $\boldsymbol{\alpha}_0$ is never updated and TIDBD and TD are equivalent.

Figure 3.2 depicts the mean squared error of both semi-gradient and ordinary-gradient TIDBD varying for settings of their meta step size. Semi-gradient TIDBD is less sensitive to meta step size θ values than ordinary TIDBD. For all but $\boldsymbol{\alpha}_0 = 2.5 \times 10^{-2}$, there are broad settings of θ such that semi-gradient TIDBD attains better final performance than ordinary TD. For every initial step size in this parameter sweep, except for $\boldsymbol{\alpha}_0 = 2.5 \times 10^{-2}$ and $\boldsymbol{\alpha}_0 = 1 \times 10^{-2}$, there are settings of θ such that ordinary-gradient TIDBD is an improvement over ordinary fixed step size TD.

For all but two initial step-size settings, there are meta step-size values θ for which ordinary-gradient and semi-gradient TIDBD perform as well as or better than TD with a single-shared step-size. That is, even without the advantages of adapting step-sizes on a per-feature basis, TIDBD can attain performance similar to TD on a synthetic prediction task. However, TIDBD's performance is sensitive to θ: an additional meta step-size parameter introduced by TIDBD. In the following section, we apply normalisation techniques from AutoStep (Mahmood et al., 2012) to reduce sensitivity θ, and create a step-size adaptation method that is effectively tuning-free.

3.4 Auto TIDBD: AutoStep for TD Learning

For most of the initial step-size settings in the previous section, both ordinary-gradient and semi-gradient TIDBD was able to perform as well as or better than TD for some meta-step-size θ value at each α_0 value; however, for ordinary-gradient TIDBD, the best θ values varied for different α_0 values.

One benefit of TD with a static shared step size is that Reinforcement Learning practitioners have an intuition of what range of step-size α values will yield acceptable performance in general; however, the optimal step size value will vary from between problems. With TIDBD and other similar meta-descent methods, the performance is dependent on both the initial step-size α_0, and a meta step size θ—an additional value for practitioners to develop an intuition for.

AutoStep (Algorithm 5)—an extension of IDBD—reduced sensitivity to the setting of the meta step size θ and prevented divergence in the supervised learning setting (Mahmood et al., 2012). To prevent divergence, AutoStep makes two additions to IDBD. First, the meta-weight update is normalized by a decaying average of recent weight updates. Second, the step sizes are normalized by the amount of error that was reduced on a given example by performing a learning update—termed the *effective step size*. By normalizing the current step sizes, a weight update will never overshoot as a result of a weight update on the current observed example: the error cannot be over-corrected on a given example to the point that error is introduced.

AutoStep maintains $\boldsymbol{\eta}$, a running trace of recent weight updates to normalize updates to the step sizes. At each time-step AutoStep takes the maximum between the current meta-weight update $|\delta \mathbf{oh}|$ and a decay of the previous maximum $\boldsymbol{\eta}_i + \frac{1}{\tau}\boldsymbol{\alpha}_i \mathbf{o}_i^2(|\delta \mathbf{o}_i \mathbf{h}_i| - \boldsymbol{\eta})$, where scalar τ is a large value that weights the decay of $\boldsymbol{\eta}$.

$$\boldsymbol{\eta} \leftarrow \max(|\delta \mathbf{oh}|, \boldsymbol{\eta} + \frac{1}{\tau}\boldsymbol{\alpha}\mathbf{o}^2(|\delta \mathbf{oh}| - \boldsymbol{\eta})) \tag{3.15}$$

One might consider why the maximum is decayed rather than simply stored—as is done with NOSID (Dabney, 2014). In real-world data sources,

noise and other outliers could distort the absolute recorded maximum, making the normaliser adjust input values into an unrepresentative range. By decaying the maximum, a learner can recover gradually from extreme data points.

After the meta-weight vector has updated, the resulting step size $\boldsymbol{\alpha}$ is normalised by the *effective step size* $\sum_{i=1}^{n}(\boldsymbol{\alpha}_i \mathbf{o}_i^2)$. The effective step size describes the amount by which error on the current example is reduced by making a weight update. An effective step size equal to one means that the error has been entirely reduced for the current example. By dividing the current step size $\boldsymbol{\alpha}$ by $\max(\sum_{i=1}^{n}(\boldsymbol{\alpha}_i \mathbf{o}_i^2), 1)$, over-shooting the update on a given example is prevented.

Algorithm 5 AutoStep

initialise:

Vectors $\mathbf{h} \in 0^n$, $\boldsymbol{\eta} \in 0^n$. Initialise $\mathbf{w} \in \mathbb{R}^n$ and $\boldsymbol{\alpha} \in \mathbb{R}^n$ as desired; a scalar $\mu > 0$ and a small constant $\tau > 0$. observe the environment $\mathbf{o} \in \mathbb{R}^n$.

repeat For each observation $\mathbf{o}'$, cumulant c:

$\delta \leftarrow c + \gamma \mathbf{w}^\top \phi(\mathbf{o}') - \mathbf{w}^\top \phi(\mathbf{o})$

for element $i = 1, 2, \cdots, n$: **do**

$\boldsymbol{\eta}_i \leftarrow \max(|\delta \mathbf{o}_i \mathbf{h}_i|, \boldsymbol{\eta}_i + \frac{1}{\tau}\boldsymbol{\alpha}_i \mathbf{o}_i^2(|\delta \mathbf{o}_i \mathbf{h}_i| - \boldsymbol{\eta}))$

if $\boldsymbol{\eta}_i \neq 0$: **then**

$\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i \exp(\mu \frac{\delta \mathbf{o}_i \mathbf{h}_i}{\boldsymbol{\eta}_i})$

$M \leftarrow \max(\sum_{i=1}^{n}(\boldsymbol{\alpha}_i \mathbf{o}_i^2), 1)$

for element $i = 1, 2, \cdots, n$ **do**

$\boldsymbol{\alpha}_i \leftarrow \frac{\boldsymbol{\alpha}_i}{M}$

$\mathbf{w}_i \leftarrow \mathbf{w}_i + \boldsymbol{\alpha}_i \delta \mathbf{o}_i$

$\mathbf{h} \leftarrow \mathbf{h}_i(1 - \boldsymbol{\alpha}_i \mathbf{o}_i^2) + \boldsymbol{\alpha}_i \delta \mathbf{o}_i$

$\mathbf{o} \leftarrow \mathbf{o}'$

3.4.1 AutoStep for TD

Having introduced AutoStep, I now add Auto-step's normalization to both semi-gradient and ordinary-gradient TIDBD to improve their stability.

First, I define the effective step-size in the on-policy TD setting. The effective step size is computed by taking the relative difference between the error before the weight update, δ_t, and the error after the weights have been updated δ_t^+, or $\frac{\delta_t - \delta_t^+}{\delta_t}$. The error $\delta_t^+ = c_{t+1} + \gamma v_{t+1}(\phi(\mathbf{o}_{t+1})) - v_{t+1}(\phi(\mathbf{o}_t))$ is defined using the weights after the update from time-step t.

For supervised learning, the notion of an effective step size is straightforward: there is a known target value and the error reduced on a given time-step is directly observable. However, TD learning uses bootstrapping. For TD learning, the effective step size is a biased estimation dependent on how accurate the value-function is in estimating the value of the following step $v(\phi(\mathbf{o}_{t+1}))$.

$$\begin{aligned}
\frac{\delta_t - \delta_t^+}{\delta_t} &= \frac{[c_{t+1} + \gamma v_t(\phi(\mathbf{o}_{t+1})) - v_t(\phi(\mathbf{o}_t))]}{\delta_t} \\
&- \frac{[c_{t+1} + \gamma v_{t+1}(\phi(\mathbf{o}_{t+1})) - v_{t+1}(\phi(\mathbf{o}_t))]}{\delta_t} \\
&= \frac{[\gamma v_t(\phi(\mathbf{o}_{t+1})) - v_t(\phi(\mathbf{o}_t))]}{\delta_t} \\
&- \frac{[\gamma(v_t(\phi(\mathbf{o}_{t+1})) + (\boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t)^\top\phi(\mathbf{o}_{t+1})) - (v_t(\phi(\mathbf{o}_t)) + (\boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t)^\top\phi(\mathbf{o}_t)]}{\delta_t}
\end{aligned} \tag{3.16}$$

Here, δ_t^+ is expanded as the TD error of the current time-step t using the value-functions from the following time-step, v_{t+1}. Value functions may be written recursively as the sum of the previous time-step's value-function $v_t(\phi(\mathbf{o}_t))$ and the current weight update $\boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t$. So, $v_{t+1}(\phi(\mathbf{o}_t)) = v_t(\phi(\mathbf{o}_t)) + [\boldsymbol{\alpha}_{t+1}\delta_t(t)\mathbf{z}_t]^\top\phi(\mathbf{o}_t)$.

$$\begin{aligned}
\frac{\delta_t - \delta_t^+}{\delta_t} &= \frac{[\gamma v_t(\phi(\mathbf{o}_{t+1})) - \gamma v_t(\phi(\mathbf{o}_{t+1}))]}{\delta_t} \\
&- \frac{[v_t(\phi(\mathbf{o}_t)) - v_t(\phi(\mathbf{o}_t))] - [\gamma\boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t^\top\phi(\mathbf{o}_{t+1}) - \boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t^\top\phi(\mathbf{o}_t)]}{\delta_t} \\
&= \frac{-[\gamma\boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t^\top\phi(\mathbf{o}_{t+1}) - \boldsymbol{\alpha}_{t+1}\delta_t\mathbf{z}_t^\top\phi(\mathbf{o}_t)]}{\delta_t} \\
&= -(\boldsymbol{\alpha}_{t+1}\mathbf{z}_t)^\top[\gamma\phi(\mathbf{o}_{t+1}) - \phi(\mathbf{o}_t)]
\end{aligned} \tag{3.17}$$

The resulting effective step size is $-(\boldsymbol{\alpha}\mathbf{z})^\top[\gamma\phi(\mathbf{o}_{t+1})-\phi(\mathbf{o}_t)]$. This is an intuitive result, as the amount by which an agent will reduce its error on a given example is the difference between the update made to the features active in the target $\phi(\mathbf{o}_{t+1})$ and the changes made to the features in the state $\phi(\mathbf{o}_t)$.

The effective step-size is the same for both ordinary-gradient and semi-gradient methods. Because the effective step-size is calculated with respect to the underlying TD error, and simplified using the underlying TD updates, it will not change if the meta-learning method uses a different objective, or if a different gradient is used.

Having defined the effective step-size for TD, I now generalise AutoStep's decaying trace of meta-weight updates to TIDBD. To do so, I return to bare-notation without time-step subscripts: for example, $\delta_t = \delta$. AutoStep normalises updates to the step-sizes by maintaining a running trace of the absolute value of the weight-updates.

$$\max(|\delta\mathbf{o}_i\mathbf{h}_i|, \boldsymbol{\eta}_i + \frac{1}{\tau}\boldsymbol{\alpha}_i\mathbf{o}_i^2(|\delta\mathbf{o}_i\mathbf{h}_i| - \boldsymbol{\eta}_i)) \tag{3.18}$$

For ordinary-gradient TIDBD, The absolute weight update is $|\delta[\gamma\phi(\mathbf{o}_{t+1}) - \phi(\mathbf{o}_t)]\mathbf{h}|$, and the current active step size is $\boldsymbol{\alpha}[\gamma\phi(\mathbf{o}_{t+1}) - \phi(\mathbf{o}_t)]$. Thus, the trace $\boldsymbol{\eta}$ of the maximum weight update is:

$$\max(|\delta[\gamma\phi_i(\mathbf{o}_{t+1}) - \phi_i(\mathbf{o}_t)]\mathbf{h}_i|, \boldsymbol{\eta}_i - \frac{1}{\tau}\boldsymbol{\alpha}_i[\gamma\phi_i(\mathbf{o}_{t+1}) - \phi_i(\mathbf{o}_t)]\mathbf{z}_i(|\delta\phi_i(\mathbf{o}_t)\mathbf{h}_i| - \boldsymbol{\eta}_i)) \tag{3.19}$$

For semi-gradient TIDBD, the absolute weight update is $|\delta\phi(\mathbf{o}_t)\mathbf{h}|$, and the current active step size is $\boldsymbol{\alpha}\phi(\mathbf{o}_t)$. Thus, the trace $\boldsymbol{\eta}$ of the maximum weight update would be:

$$\max(|\delta\phi_i(\mathbf{o}_t)\mathbf{h}_i|, \boldsymbol{\eta}_i + \frac{1}{\tau}\boldsymbol{\alpha}_i\phi_i(\mathbf{o}_t)\mathbf{z}_i(|\delta\phi_i(\mathbf{o}_t)\mathbf{h}_i| - \boldsymbol{\eta}_i)) \tag{3.20}$$

For simplicity, I only present ordinary-gradient AutoTIDBD in Algorithm 6. The only modification of Algorithm 6 required to produce semi-gradient AutoTIDBD, is to change the $\boldsymbol{\eta}$ update from the max provided in Equation 3.19 to the max in Equation 3.20.

Algorithm 6 Ordinary-gradient AutoTIDBD

initialise:

Vectors $\mathbf{h} \in 0^n$, $\boldsymbol{\eta} \in 0^n$, $\mathbf{z} \in 0^n$. Initialise $\mathbf{w} \in \mathbb{R}^n$ and $\boldsymbol{\beta} \in \mathbb{R}^n$ as desired; a scalar $\theta > 0$ and a small constant $\tau > 0$. observe the environment $\mathbf{o} \in \mathbb{R}^n$.

repeat For each observation $\mathbf{o}'$ and cumulant c:

$\delta \leftarrow c + \gamma \mathbf{w}^\top \phi(\mathbf{o}') - \mathbf{w}^\top \phi(\mathbf{o})$

$\boldsymbol{\eta}_i \leftarrow \max[$

$|\delta[\gamma\phi_i(\mathbf{o}') - \phi_i(\mathbf{o})]\mathbf{h}_i|,$

$\boldsymbol{\eta}_i - \frac{1}{\tau}\boldsymbol{\alpha}_i[\gamma\phi_i(\mathbf{o}') - \phi_i(\mathbf{o})]\mathbf{z}_i(|\delta\phi_i(\mathbf{o})\mathbf{h}_i| - \boldsymbol{\eta}_i)$

$]$

for element $i = 1, 2, \cdots, n$: **do**

if $\boldsymbol{\eta}_i \neq 0$ **then**

$\boldsymbol{\beta}_i \leftarrow \boldsymbol{\beta}_i - \theta\frac{1}{\boldsymbol{\eta}_i}\delta[\gamma\phi_i(\mathbf{o}')) - \phi_i(\mathbf{o})]\mathbf{h}_i$

$M \leftarrow \max(-e^{\boldsymbol{\beta}}[\gamma\phi(\mathbf{o}') - \phi(\mathbf{o})]^\top \mathbf{z},\, 1)$

$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \log(M)$

$\boldsymbol{\alpha} \leftarrow e^{\boldsymbol{\beta}}$

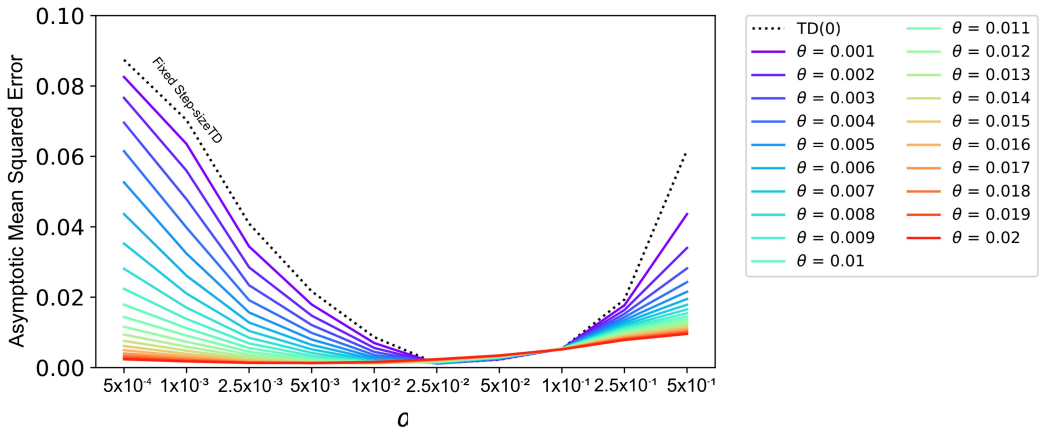
$\mathbf{z} \leftarrow \mathbf{z}\gamma\lambda + \phi(\mathbf{o})$

$\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\alpha}\delta\mathbf{z}$

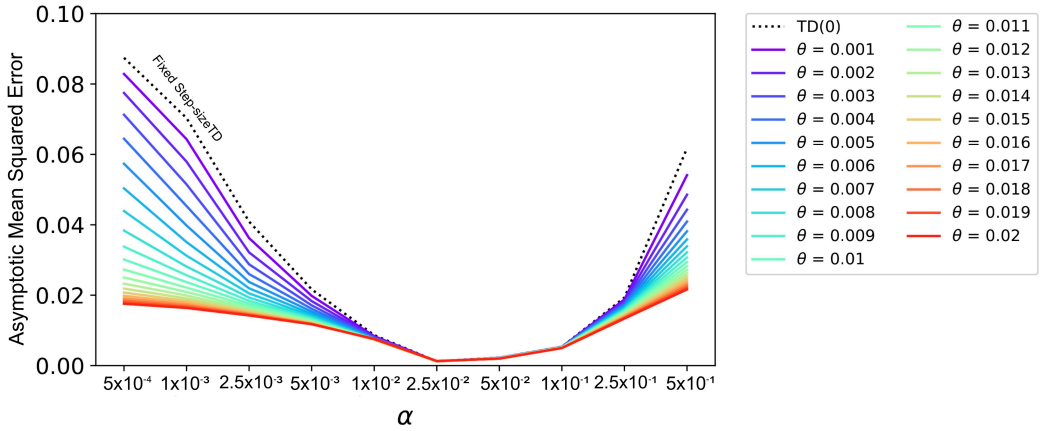
$\mathbf{h} \leftarrow \mathbf{h}\ \mathrm{relu}(1 + \boldsymbol{\alpha}[\gamma\phi(\mathbf{o}') - \phi(\mathbf{o})]\mathbf{z}) + \boldsymbol{\alpha}\delta\mathbf{z}$

$\mathbf{o} \leftarrow \mathbf{o}'$

I have generalised AutoStep to TD learning, which I call AutoTIDBD. Now I assess AutoTIDBD's performance to determine whether it can perform better or equal to than tuned ordinary TD(0) while being relatively insensitive to its meta step size θ, meeting one of my core criteria for an adaptive step size algorithm.



(a) Parameter study of semi-gradient AutoTIDBD for varying $\boldsymbol{\alpha}_0$ and θ values.



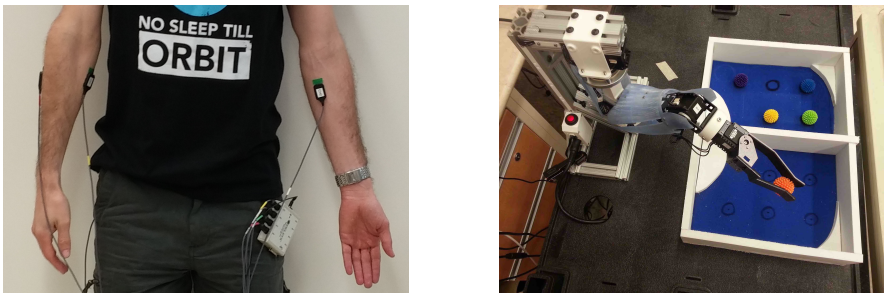
(b) Parameter study of ordinary-gradient AutoTIDBD for varying $\boldsymbol{\alpha}_0$ and θ values. Error reported is the asymptotic mean squared error over 30 independent trials.

Figure 3.3: Parameter study of semi and ordinary-gradient Auto TIDBD. Static step size TD in black. The x-axis denotes the initial step-size all learners use. In the case of TD(0), this step-size never updates. For both Semi and Ordinary-gradient TIDBD, the step-size is adapted online during the experiment with a meta step-size value denoted by the colour of the line, and described in the legend.

3.5 How well Does AutoTIDBD Adapt a Single Step Size?

In Figure 3.3, a parameter study of AutoTIDBD's sensitivity on the previously introduced grid world task (Section 3.3.1) is presented. Similar to the previous semi-gradient TIDBD results, there are broad ranges of θ values for which semi-gradient AutoTIDBD outperforms ordinary TD in Figure 3.3a. Moreover, there is no best meta step-size θ value across all initial step-size values α_0: the ordering of performance over the meta step-size varies for different initial step sizes. For the lowest points in the bowl, around 2.5×10^{-2} to 1×10^{-1}, the smallest meta-step size values are best performing. Outside this range of best performance, the largest meta step-size values are best performing. Semi-gradient AutoTIDBD still requires tuning in this setting to perform as well as fixed step-size TD consistently across initial step-sizes. In Figure 3.3b, ordinary-gradient AutoTIDBD performs as well as or better than TD for all meta step-size values θ swept over, and the ordering is consistent across initial step-size values α_0. While the absolute best performance may vary for different values of α_0, for ordinary-gradient AutoTIDBD, the change in performance as θ varies is predictable and consistent.

An important consideration of these experiments is the number of step sizes being adapted by the IDBD-based methods I generalised. For all experiments presented in this section, a single step-size is shared amongst all features. This limitation enables us to examine the performance of TIDBD and AutoTIDBD without the advantages of adapting step sizes on a per-feature basis. Even without the representation learning effects that per-feature step-size adaptation affords, ordinary-gradient AutoTIDBD performs as well as or better than ordinary TD learning on this synthetic prediction task.



(a) A subject with electrodes attached to their wrist flexors and extensors.

(b) The BentoArm performing a modified Box and Blocks task.

Figure 3.4: Experiment setup for the robotic prediction task.

3.6 How Robust is AutoTIDBD to Selection of Meta Step Size θ When Adapting Many Step-sizes?

In the previous sections, I showed that on a synthetic prediction task, ordinary-gradient AutoTIDBD was able to perform as well as or better than TD with a fixed step-size. I now evaluate how well AutoTIDBD performs when using a vector of many step sizes—when it is performing representation learning. From here forwards, I consider ordinary-gradient AutoTIDBD, which I refer to as OG AutoTIDBD. Although semi-gradient AutoTIDBD achieves lower error in than OG AutoTIDBD for many meta-parameter settings, I choose OG AutoTIDBD for the remainder of empirical analysis due to its stability: OG AutoTIDBD rarely has an error greater than ordinary tuned TD learning, and there is a consistent ordering of performance over meta step-size θ values. In this section, I evaluate OG AutoTIDBD on a known, challenging, real-world prediction problem that has been used to both assess new learning methods (Seijen & Sutton, 2014) and as a domain for applying predictive knowledge in practice (Edwards, Dawson, Hebert, Sherstan, et al., 2016; Pilarski, Dawson, Degris, Carey, Chan, et al., 2013).

3.6.1 Robotic prediction task

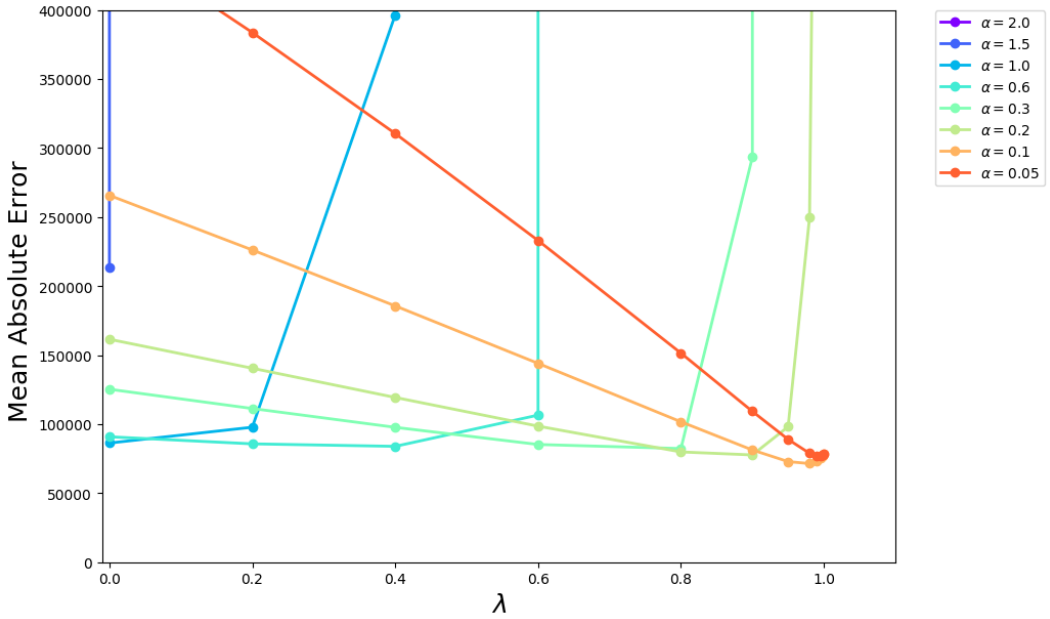
The prediction problem in this experiment consisted of predicting the temporally extended future values of signals of interest within the data stream of a robotic

arm as a user controlled it to perform a manual manipulation task. The data-set for this evaluation was drawn from a prior study (Edwards, Hebert, et al., 2016). Similar data has been used in the analysis of temporal difference learning methods (c.f. True Online TD Seijen and Sutton, 2014). In this dataset, tasks were carried out using the Bento Arm (Dawson et al., 2014), depicted in Figure 3.4b. Data was recorded while four participants used signals from their upper-arm muscles to control the robot to perform an object placement task (approved protocol #Pro00030709, University of Alberta institutional ethics review board). Signals in the data stream included the moment-by-moment position, velocity, load, temperature of all the robot's motors, along with the control signals being sent by the human user. Users were tasked with switching between the multiple controllable degrees of the robot arm to move balls from one side of a divided box to another.

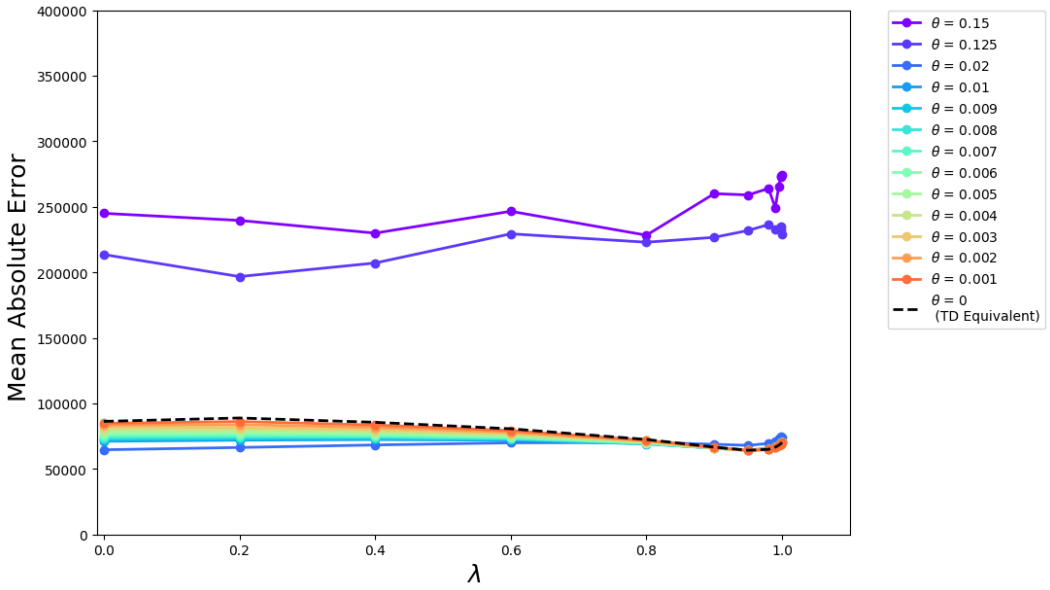
In this dataset four participants each performed the manipulation task a total of six times: three times with a non-adaptive control system, and three times with an adaptive control system that changes over time in response to the participant's behaviour (c.f. Edwards, Hebert, et al., 2016), creating a total of 24 independent trials. Robotic platforms provide data that is inherently non-stationary: sensor readings drift over time, and mechanical systems change through wear-and-tear. The inclusion of data from the adaptive control case brings additional non-stationarity to the prediction problem: non-stationarity is introduced as the user becomes more proficient at the manipulation task, and as the control system begins to adapt to the user's preferred movements.

In this setting no single scalar step-size is best at all times, as the prediction problem changes over time. As has been previously noted (Pilarski, Dawson, Degris, Carey, Chan, et al., 2013), finding appropriate features and setting appropriate parameters for learning systems is a known challenge in this robotic control domain. End-user time is precious, and designers cannot possibly test their prediction algorithms on datasets which are representative of all the situations the robot might encounter in deployment. Aspects such as non-stationarity and the irregularities introduced through human-in-the-loop control therefore make this dataset an appropriate one for studying the robustness of

OG AutoTIDBD in predicting a real-world data stream.



(a) Average cumulative error of TD(λ) for various $\boldsymbol{\alpha}$ settings.



(b) Average cumulative return error of OG AutoTIDBD for various θ settings.

Figure 3.5: Cumulative error comparison of TD and OG AutoTIDBD for prediction of hand position. The best setting of α varies across different λ settings for ordinary TD learning. For OG AutoTIDBD, there exists a broad range of θ values for which performance is acceptable.

3.6.2 Sensitivity to meta step size θ in a prosthetic prediction problem and performance relative to existing methods.

In this section, two kinds of comparisons are made. First, I analyse the performance of OG AutoTIDBD in comparison to fixed step-size TD to determine whether OG AutoTIDBD requires less tuning than TD learning when adapting many step-sizes. Second, I compare OG AutoTIDBD to a variety of existing step-size adaptation methods for RL to assess how well existing methods translate to the non-stationary prediction setting, and how well OG AutoTIDBD performs in comparison to them.

Each learner in this comparison predicted the angular position of the robot's hand motor (the gripper's aperture), as in Seijen and Sutton (2014). TileCoding (Sutton & Barto, 2018) was used to construct a binary feature vector of size 2^{10} with 8 tilings and used the velocity of the hand, the position of the hand, and the participant's control signals to construct the feature vector. An additional bias feature was concatenated to the feature-vector, resulting in 9 active features.

OG AutoTIDBD was compared to NoSID, SID, AutoSID (Dabney & Barto, 2012), RMSProp (Tieleman & Hinton, 2012), and AlphaBound (Dabney, 2014), implemented as described in their respective source material. AlphaBound is initialised with a step size of 1, as originally specified. Each learning method used a discount of $\gamma = 0.95$.

All IDBD-based methods initialize their step-sizes as $\frac{1}{9}$. This step-size was chosen because it follows a practitioner rule-of thumb for TD to set the step-size as 1/#active features. We can see some empirical sense in this folk-knowledge by examining the performance of TD on this prediction task (Figure 3.5a): a step-size of $\frac{1}{9}$ yields performance that does not diverge on the task.

First, let us compare OG AutoTIDBD when adapting many step sizes to TD's performance. For OG AutoTIDBD to be an improvement over ordinary, fixed step size TD, it should be less sensitive to settings of θ than TD is to settings of α. In Figure 3.5a, the sensitivity of static step size TD(λ) is shown for a variety of α values across λ settings. There is no single step size α which

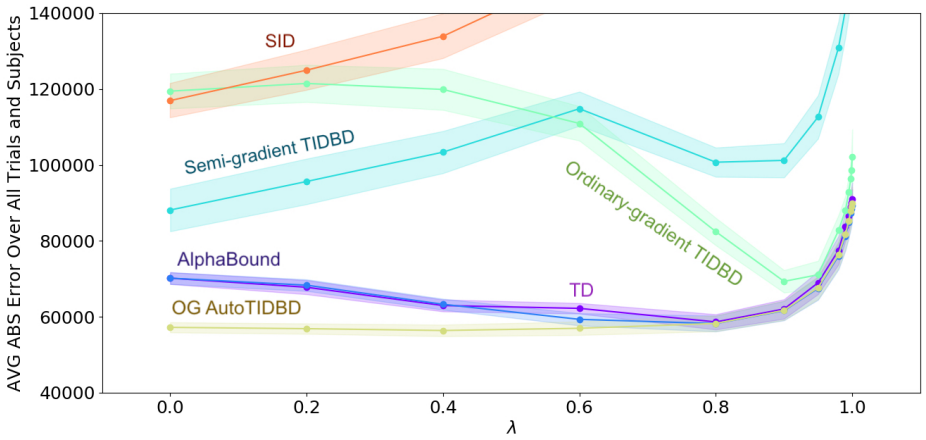


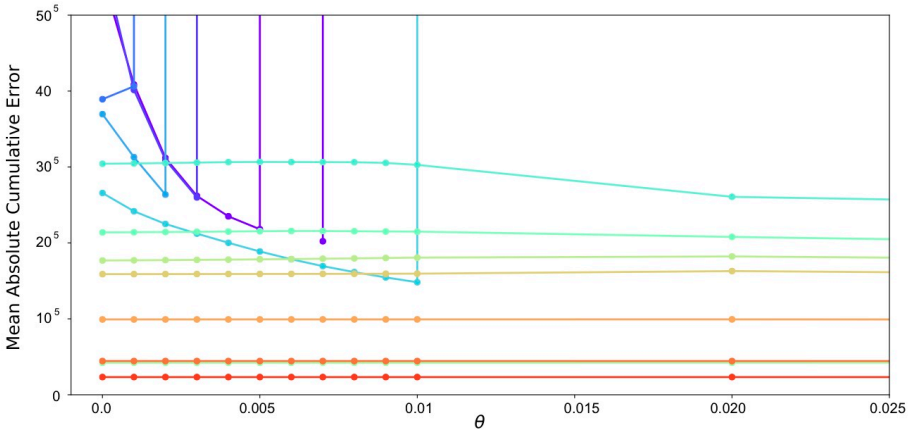
Figure 3.6: Absolute cumulative return error averaged over 24 independent trials for a variety of learning methods. Error bars are SEM. Each algorithm is presented for the best setting of $\boldsymbol{\alpha}$ or θ and is compared by varying λ between 0 and 0.9. TD with RMSProp, AutoSID, and NOSID could not be plotted on the same chart, as they had errors which were too large to be compared.

performs well for all λ values. In Figure 3.5b OG AutoTIDBD's performance on the same prediction problem is shown. OG AutoTIDBD is less sensitive to θ than TD(λ) is to the setting of α, and OG AutoTIDBD performs as well as or better than TD(λ), except for large meta step-size values. Importantly, one of the best performing meta step-size values across the experiment is $\theta = 10^{-2}$, the meta step-size originally identified by Mahmood et al. as the best performing meta step-size for AutoStep over a variety of supervised learning problems (Mahmood et al., 2012).

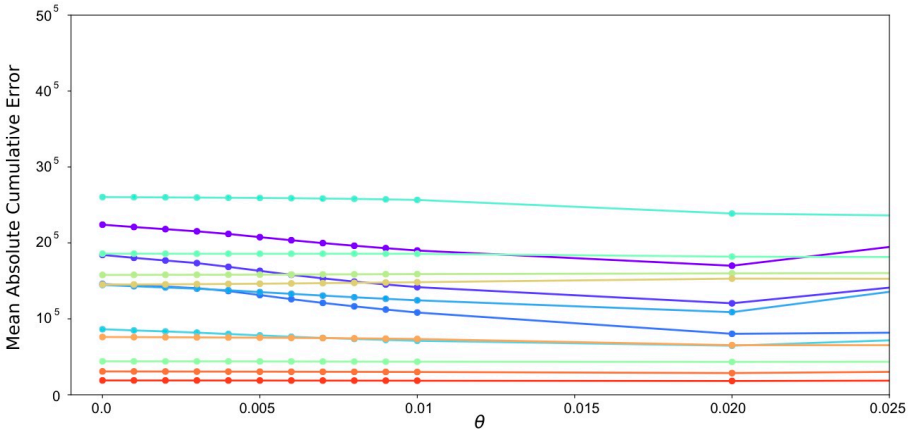
Now let us examine how well existing step-size adaptation methods translate to this non-stationary prediction setting, and evaluate how well OG AutoTIDBD performs in comparison to them. In Figure 3.6, the average cumulative error for the best tuned parameter settings for multiple prediction learners is shown. TIDBD is visually compared against both SID and AlphaBound. TD with RMSProp, AutoSID, and NOSID could not be plotted on the same chart, as they had errors which were too large to be compared.

Both Semi-gradient TIDBD and Ordinary-gradient TIDBD outperform SID—A scalar version of IDBD for TD learning; however, neither consistently attain errors less than nor equal to ordinary TD.

On this non-stationary prediction task, OG AutoTIDBD attained performance equal to, or better than TD for all settings. Except for AlphaBound at large values of λ, OG AutoTIDBD performed as well as, or better than all existing step-size adaptation methods compared against.



(a) Average absolute cumulative return error of TIDBD(0) for different values of θ. Each line represents the error of a prediction with its own unique signal of interest. Each on-policy prediction has the same γ, but is predicting a different signal of interest.



(b) Average absolute cumulative return error of OG AutoTIDBD(0) for different values of θ. Each line represents the error of a prediction with its own unique signal of interest. Each on-policy prediction has the same γ, but is predicting a different signal of interest.

Figure 3.7: Cumulative error for different meta-step size values across a variety of different prediction problems on the bionic limb. Predictions of velocity, position, and load for all five servos of the robot arm are depicted.

3.6.3 Sensitivity to meta step size θ across prediction problems

I previously assessed the performance of OG AutoTIDBD across meta step size settings for predictions of the gripper. On the gripper prediction task, there was a broad range of meta step-size θ settings for which OG AutoTIDBD attained acceptable performance. Moreover, OG AutoTIDBD achieved lower cumulative error than all methods compared against, with the exclusion of AlphaBound at large λ values. In this section, I explore whether the best settings of θ are relatively invariant over different robot prediction problems.

Using the data and experimental setup from Section 3.6.1, I now compare the sensitivity of OG AutoTIDBD and TIDBD on a variety of prediction problems. Each of the signals produced by the arm are distinct; therefore predicting each signal is a different problem. If the best meta step-size is invariant across prediction problems, then OG AutoTIDBD can be seen as a tuning-free step-size adaptation method.

In Figure 3.7a, the sensitivity of ordinary-gradient TIDBD to its meta-parameter θ settings is shown. Each line represents the performance of ordinary-gradient TD in predicting a different signal of interest from the arm. For Ordinary-gradient TD without AutoStep's normalization, the best setting of θ for each prediction problem is different. For several prediction problems, the best value of θ results in divergence on for other prediction problems. ordinary-gradient TIDBD may be less sensitive to initialization of its parameters than ordinary TD, but must still be tuned for each target domain.

In Figure 3.7b the same meta-sensitivity for OG AutoTIDBD is depicted across meta step-size θ settings. The valley of best performance is relatively invariant across θ values: the best meta step-size for any prediction problem is one of the best settings for all the other problems.

OG AutoTIDBD meets the requirements I introduced at the beginning of this chapter: It performs as well as or better than ordinary TD(λ)—even when only adapting a single step size, it is less sensitive to settings of θ than TD(λ) is to α, it has broad ranges of meta step-size values θ for which performance

is acceptable, and the best θ settings are relatively invariant across problems. OG AutoTIDBD is a bias-adaptation step size method which does not require tuning across applications.

3.6.4 Can AutoTIDBD perform representation learning?

I have demonstrated that OG AutoTIDBD can outperform scalar step-size adaptation methods and ordinary TD on real-world prediction problems by tuning its step sizes, and that it is less sensitive to its parameters than ordinary TD. I now evaluate whether OG AutoTIDBD can effectively perform representation learning by assigning step-size values based on feature relevance.

Can OG AutoTIDBD differentiate between features that are relevant to the prediction problem, and those which are not? To answer this question, I return to the prediction task introduced in Section 3.6.1 and analyse the change in step size values at $\lambda = 0.95$. I introduced noisy features to the agent's feature vector by randomly masking 25% of the binary features. Masked features were activated with a probability of 0.5. After completion of the experiment, the noisy features selected were analysed to ensure that some masked features were normally highly active.

If OG AutoTIDBD can learn the relevance of features, we should expect that the noisy features will be given smaller step-sizes over the duration of an experiment, and that there should be a separation between the ordinary features and the noisy features. Figure 3.8 depicts the magnitude of all step sizes averaged over all trials. step sizes corresponding to noisy features (coloured in greyscale) consistently decrease over time. Noisy features shrink as experience determines them to be uncorrelated with the error. This creates a separation between features: none of the noisy features have values within the range of ordinary features—all of the noisy features were correctly given smaller step sizes. There are certain time-steps for which all the step sizes suddenly decrease. On these time-steps, the effective step size was greater than 1, leading to the normalization of the meta-weight vector to prevent over-shooting. OG AutoTIDBD can perform representation learning by assigning appropriate step sizes, meeting my final criteria for a step size adaptation method.

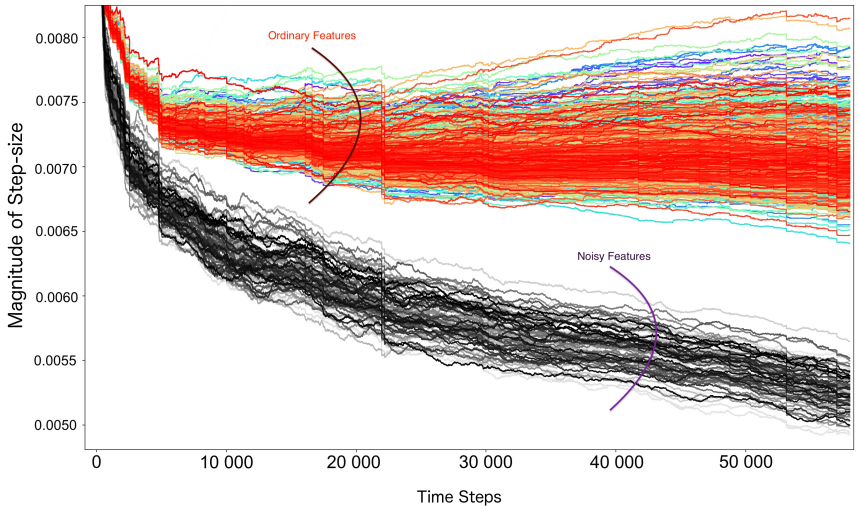


Figure 3.8: Average magnitude of step sizes over all trials. Noisy features are in greyscale, ordinary features are in colour.

3.7 Examining AutoTIDBD for Real-world Robotics

Artificial agents deployed in the real-world face continued challenges: the real-world is non-stationary and complex. How the environment might change over time, and the situations encountered by the agent cannot be entirely foreseen by system designers. These challenges are compounded when they are deployed for long periods of time. For these reasons, developing learning methods that support lifelong continual learning is of importance for agents that inhabit the real-world. As articulated in prior sections of this chapter, it is natural that a long-lived agent should modify its learning parameters independent of human intervention, and over time. In mammals, the process of destabilising memories and making them susceptible to change is similar to a temporary increase in learning rates in an autonomous agent (Sinclair & Barense, 2018).

In this section, I continue to assess OG AutoTIDBD on a robotics platform. First, I demonstrate again that OG AutoTIDBD is less sensitive to step size settings than ordinary TD learning, and that OG AutoTIDBD performs consistently for a broad set of initial parameter values. Second, I demonstrate that by examining step sizes learned through OG AutoTIDBD, common sensor failures can be detected. In particular, I consider situations where settings

where sensor readings exhibit two common failure modes: 1) where sensor values become "stuck" constant values, and 2) where sensors themselves become noisy. As a primary conclusion of this chapter, I demonstrate that by examining the step size values learned via OG AutoTIDBD, insight into the operation and internal learning processes of long-lived continual learning agents can be gained.

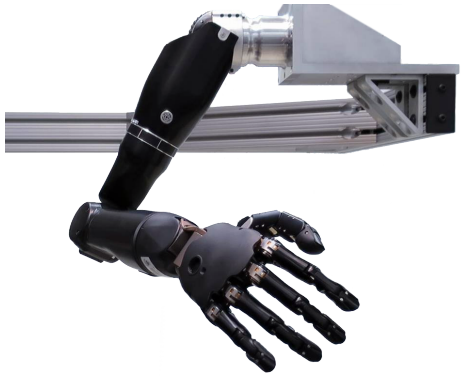


Figure 3.9: The Modular Prosthetic Limb (MPL), a robot arm with many degrees of freedom and sensors used for the experiments in this chapter.

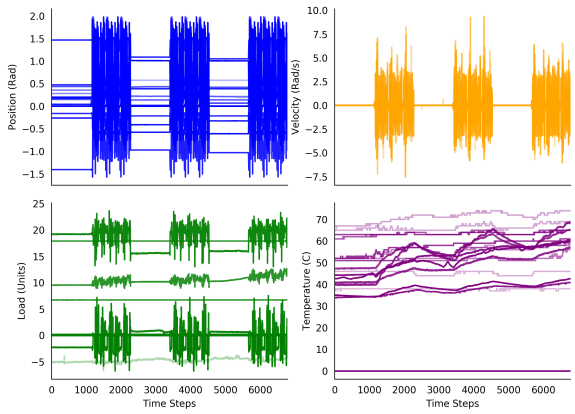


Figure 3.10: Decoded percept data from the robot over the 30 minutes of the experiment. The phases of the arm resting and the phases of the arm moving are clearly distinguishable for the position, velocity, and load sensors. The values of the temperature sensors increase over the experiment, with additional increases during the phases of movement.

3.7.1 Experimental setup

The experiments in this section use data collected from the Modular Prosthetic Limb (MPL v3) (Bridges et al., 2011)—a state-of-the-art bionic limb capable of human-like movements. The MPL has 17 degrees of freedom, across 26 articulated joints in the shoulder, elbow, wrist, and hand. Each individual motor has a load, position, temperature, and current sensor. Each fingertip on the hand has a 3-axis accelerometer with 14 pressure pad sensor arrays. Altogether, the MPL provides 108 real-valued sensor readings (Figure 3.10).

Robot Actuation:

To create a non-stationary stream of experience during experimental trials, the robot arm performed a series of complex natural motions, followed by a phase of rest. The experiment started with the robot holding its position for five minutes and was followed by five minutes in which the arm repeated a complex pattern. The arm engaged in a complex natural movement: e.g., grasping motions and flexing individual fingers. The movement was 100 seconds long and was repeated three times during the five-minute movement phase. The arm alternated between rest and movement three times, resulting in six distinct phases of behaviour over 30 minutes.

During the rest phase, the sensor readings on the robotic limbs report a relatively constant signal up to machine precision. In contrast, during the movement phase sensor readings are challenging to predict. By switching between different modes of actuation, non-stationarity is explicitly created within the prediction problem. I report results on a per-phase basis to capture the effects of non-stationarity on learning.

Predictions: For the following experiments, estimates of the return of each of the 108 individual sensor readings are learned. The cumulant c_i is the ith sensor reading. Each GVF is learned on-policy, and each GVF has a constant discount of $\gamma = 0.9$ for all predictions.

A discount of 0.9 is approximately a horizon of 10 time-steps. Time-steps were approximately 0.265 seconds apart, resulting in an estimated accumulation over approximately 2.65 seconds. This time-horizon captures slow movements

with appropriate resolution: e.g., extension and flexion of the elbow joint. However, for faster movements, such a prediction horizon provides a more coarse accumulation.

Each GVF's state was constructed using the whole observation vector: all 108 sensor readings. Sensor readings were normalised and passed through a function approximator to produce a feature vector. For the following experiments, a Selective Kanerva Coder (Travnik & Pilarski, 2017) is used as a function approximator. Selective Kanerva Coders are parameterised by two values: the number of prototypes used n, and $\boldsymbol{\eta}$ the number of prototypes active on each time-step. The number of prototypes n determines the size of the feature vector, and the prototypes $\boldsymbol{\eta}$ determine how many features are active for a given input vector.

GVF estimates were computed offline after data collection. As a result, the clock time required to update a prediction and produce an estimate did not impact the duration of a time-step.

The performance of TD and OG AutoTIDBD are compared based on the root mean squared prediction error (RMSE) (Equation 3.21), averaged over all prediction estimates.

$$RMSE_t = \sqrt{\frac{\sum_{i=1}^{108} \left(\frac{G_t^{(i)} - \mathbf{x}(s_t)^\top \mathbf{w}_t^{(i)}}{\left| G_t^{(i)} \right|} \right)^2}{108}} \tag{3.21}$$

The superscript (i) denotes the ith prediction of all 108 GVFs. Each difference is normalised by the absolute value of the return to ensure that prediction errors between sensors of different ranges were comparable.

Parameter	Count	Candidates
n	4	10000, 20000, 30000, 40000
$\boldsymbol{\eta}$	6	0.001, 0.002, 0.004, 0.008, 0.016, 0.032
α	11	$\frac{0.001}{n\cdot\boldsymbol{\eta}}$, $\frac{0.002}{n\cdot\boldsymbol{\eta}}$, $\frac{0.004}{n\cdot\boldsymbol{\eta}}$, $\frac{0.008}{n\cdot\boldsymbol{\eta}}$, $\frac{0.016}{n\cdot\boldsymbol{\eta}}$, $\frac{0.032}{n\cdot\boldsymbol{\eta}}$, $\frac{0.064}{n\cdot\boldsymbol{\eta}}$, $\frac{0.128}{n\cdot\boldsymbol{\eta}}$, $\frac{0.256}{n\cdot\boldsymbol{\eta}}$, $\frac{0.512}{n\cdot\boldsymbol{\eta}}$, $\frac{1.024}{n\cdot\boldsymbol{\eta}}$

Table 3.1: Parameter candidates tested in full factorial design.

All OG AutoTIDBD learners use an initial step size $\boldsymbol{\alpha}_0$ of $\frac{1}{n\boldsymbol{\eta}}$, where $n\boldsymbol{\eta}$ corresponds to the number of active features. For ordinary TD, results are presented for the step-size that yielded the best RMSE for each individual sensor. Step size values ranging from $\frac{0.001}{n\boldsymbol{\eta}}$ to $\frac{1.024}{n\boldsymbol{\eta}}$ were swept over for step-size selection. Both TD and OG AutoTIDBD are compared on different numbers of prototypes n and activation ratios $\boldsymbol{\eta}$. Values of n and $\boldsymbol{\eta}$ were chosen based on the recommendations provided by (Travnik & Pilarski, 2017). OG AutoTIDBD introduces a meta step-size parameter, θ. For the parameter sensitivity in section 3.7.3, the meta step-size is presented over a sweep of $\theta = \{0.005, 0.01, 0.02, 0.04, 0.08, 0.16\}$. For all other results, the meta step-size is not tuned, but set to 0.02, a value that was in the range of best performance across many prediction problems presented earlier in this chapter (Figure 3.7b).

Failure simulation:

Mechanical failure and noisy sensor readings are common in real-world robotics. A physical system will eventually fail, and no sensors provide perfect readings. Learning methods should be sensitive to this reality: learning methods should be able to cope with the special challenges of real-world learning. To this end, this section investigates how OG AutoTIDBD reacts when confronted with two commonly occurring sensor failures: 1) stuck sensors and 2) broken sensors. Stuck sensors output a constant signal with a small amount of sensor noise (Li & Yang, 2012); broken sensors often output Gaussian noise with a high variance (Ni et al., 2009).

In both experiments, the signals from all four sensors in the elbow were replaced: in the first, with Gaussian noise of $\mathcal{N}(1, 0.5)$ for the stuck sensors, and with Gaussian noise of $\mathcal{N}(0, 10)$ for the broken sensors.

3.7.2 Experiment: comparison of fixed step size TD and OG AutoTIDBD

I now examine whether OG AutoTIDBD can obtain acceptable performance in comparison to TD learning. In particular, I focus on how OG AutoTIDBD performs in response to non-stationarity: how quickly OG AutoTIDBD adapts to changes in the prediction task. Over a series of movements, which I refer to as

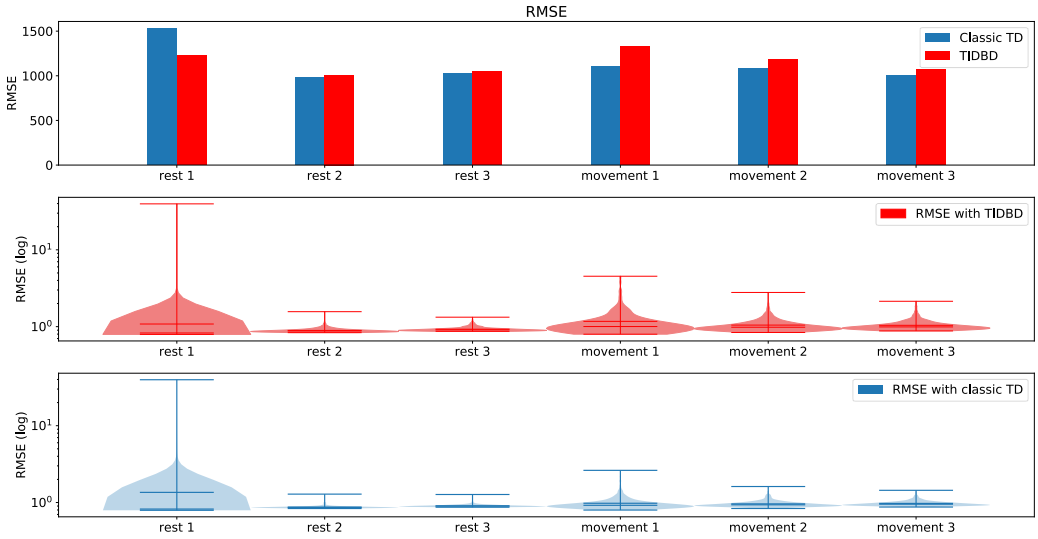


Figure 3.11: RMSE for Experiment 1: functioning sensors. The top pane shows the RMSE for both classic TD and OG AutoTIDBD for each of the different phases. The middle and bottom panes show violin plots for the RMSE, for OG AutoTIDBD and classic TD, respectively. All results are the average over 30 independent runs.

Movement Phase	Fixed step size	OG AutoTIDBD
Rest 1	1532	1223
Rest 2	984	1009
Rest 3	1023	1047
Movement 1	1105	1324
Movement 2	1086	1184
Movement 3	1003	1072

Table 3.2: Average RMSE over 30 independent runs

phases, OG AutoTIDBD's error is compared with TD learning. I compare how prediction error is accumulated over each phase, the variance in performance across parameter settings, sensitivity to parameters, and how the distribution of learned step-sizes change in response to non-stationarity.

I first examine how much error OG AutoTIDBD accumulates relative to ordinary TD learning across the successive phases of learning. In Figure 3.11 the root mean squared error is presented for both classic TD and OG AutoTIDBD in a setting where all sensors are functional. The top pane of Figure 3.11 reports the RMSE for each phase of rest and movement. Error starts high at the beginning of the experiment, and decreases over each successive

phase. The error for the second rest phase is considerably lower. Perhaps unintuitively, the error for the third rest phase increased again. This can be explained by the sensor data from Figure 3.10. One of the load sensors started to drift in the third rest phase. As this pattern had not been seen in any of the rest phases before, the RMSE peaked again—the pattern of a drifting sensor had not been learned yet. For the phases of movement, a steady decrease in RMSE was observed. Overall, OG AutoTIDBD had a slightly higher RMSE for the 30-minute experiment. The exact errors for each phase of rest and movement are given in Table 3.2. It is important to recognise that our parameter sweep over step sizes provides an advantage to classic TD; because the step sizes were chosen to minimise the RMSE for the full experimental data, the choice of step-size for the TD learner inherently provided an advantage, which OG AutoTIDBD did not receive. In a real-world application, providing this advantage via tuned parameters would not be possible, as the learner would be constantly faced with new, unknown data after the parameters have been set. Despite this advantage, OG AutoTIDBD and classic TD performed comparably with respect to RMSE.

The errors reported are with respect to the best parameter settings for both OG AutoTIDBD and TD learning. How sensitive is OG AutoTIDBD to the selection of learning parameters in comparison to TD? Unsurprisingly, TD's performance was strongly impacted by the chosen learning rate, as evidenced by the standard deviation of error across parameter combinations. TD with a shared single step size had a standard deviation of the RMSE over all 264 n, $\boldsymbol{\eta}$, and α combinations of $\sigma_{TD\_264} = 4.3 \times 10^4$. In comparison, once the best step sizes for classic TD were selected, the standard deviation for the remaining 24 combinations of n and $\boldsymbol{\eta}$ was $\sigma_{TD\_24} = 3.0 \times 10^2$. By tuning the step-size parameter, TD's performance is substantially improved, independent of the function approximator's parameters. In contrast, OG AutoTIDBD has less variance over parameter combinations: it is less sensitive to its selected parameters. OG AutoTIDBD attained a standard deviation of $\sigma_{OG\ AutoTIDBD} = 1.5 \times\ 10^3$ over the 24 Kanerva coder parameters. This value is approximately 30 times smaller than σ_{TD_264}, but approximately 5 larger

than σ_{TD_24}. OG AutoTIDBD has higher standard deviation than TD learning with a tuned step-size σ_{TD_24}; however, tuned TD's performance depends on knowledge that is determined after the experiment is complete: which step-size values yield the best performance for TD learning. OG AutoTIDBD attained acceptable performance without the need to manually tune the step size. This result indicates that OG AutoTIDBD can act as an alternative to tuned classic TD learning, without the time and labour-intensive setup that TD learning requires for tuning.

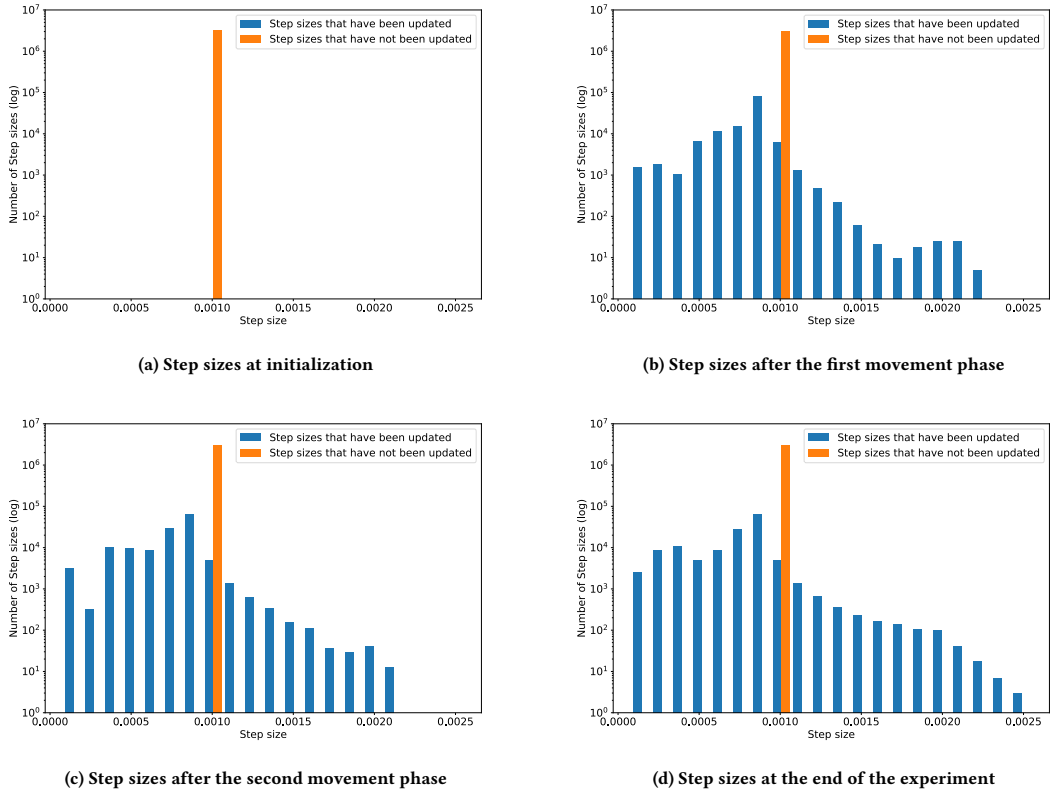
The suitability of OG AutoTIDBD for real-world robotics depends not only on its sensitivity. The computational complexity and additional time required to maintain OG AutoTIDBD's meta-weight vector must also be taken into consideration. For each weight in the 108 GVFs, an additional step size was required. Given a feature representation with $30,000$ features per GVF, $3,240,000$ step sizes were required in this particular setting. Per GVF, three additional vectors of the same size as the number of features are required. In the implementation of this experiment, each of the three additional weight vectors required 0.24 megabytes, totalling an additional 0.72 megabytes.

The additional computation to update this larger number of step sizes increased the time to update all GVFs from 0.025 seconds to 0.28 seconds. However, since this corresponds to nearly four updates per second, it was still within the requirement for prosthetic limb control.

The computations were performed using a Linux Mint 18.3 OS system with an $\mathrm{i}7-7700\mathrm{HQ}$ CPU with a clock rate of (3.80) GHz, (6) MB of shared L3 cache and 32GB DDR4 RAM. With the ongoing evolution of hardware, in the future it will be possible to maintain and update even greater numbers of GVFs or to reduce the time needed for computation.

OG AutoTIDBD assigns step sizes on a per-feature basis. Because each feature has a unique step size, the amount that each weight is affected by an update will vary over time. Using OG AutoTIDBD, step sizes related to features uncorrelated with the current prediction task will be reduced. As introduced previously, weighting updates on a per-feature basis can be interpreted as a feature selection mechanism—OG AutoTIDBD actively adapts

its representation based on interactions with the environment.



(a) Step sizes at initialization

(b) Step sizes after the first movement phase

(c) Step sizes after the second movement phase

(d) Step sizes at the end of the experiment

Figure 3.12: Step size development over the course of the experiment. As OG AutoTIDBD adapts the step sizes, this distribution will change. Subplot (A) shows the step sizes at initialisation. Subplot (B) shows the step size distribution after the first movement phase. Subplot (C) shows the step size distribution after the second movement phase. Subplot (D) shows the step size distribution at the end of the experiment.

Figure 3.12 shows four snapshots of the distribution of the step sizes over the experiment, where the sensors are functional. From these plots, we can see signs of representation learning by identifying feature relevance. While all step-sizes are initialised with the same value, some features are given small step sizes–limiting their impact on learning updates–while other features are given greater weight and more impact on learning. In each subplot, the orange bar shows step sizes that had not yet been updated due to the corresponding features not being activated; the blue bars represent the step sizes that had been updated by OG AutoTIDBD. Subplot (a) shows the step sizes at initialisation. All step sizes were initialised to 0.00104. As expected, subplots (b), (c) and

(d) show that the longer the experiment had run, the more the step sizes had spread out from their initial value. Subplot (d) shows that the step sizes were set by the end of the experiment within the range from 8.0×10^{-5} to 2.5×10^{-3}.

3.7.3 Experiment: parameter sensitivity for TD and OG AutoTIDBD

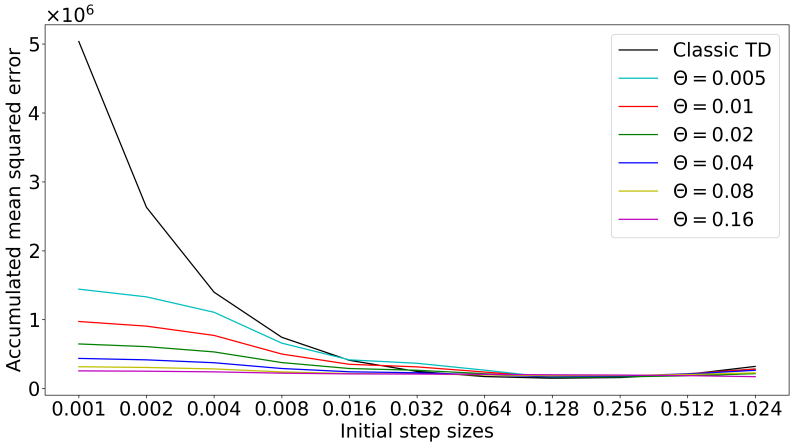
Using OG AutoTIDBD to automatically adapt parameters requires the use of meta-parameters. To investigate the sensitivity of OG AutoTIDBD with respect to initial step sizes α and the newly introduced meta step size θ, sweeps were performed on different initial values of step sizes and different meta step sizes (Figure 3.13). As expected, the RMSE for TD forms a bowl over the initial step size values: for both large and small step sizes, TD performs poorly. To find step size values that provide acceptable performance, experiment designers typically perform a sweep over many settings, as has been done here.

As expected, OG AutoTIDBD is less sensitive to initial parameters. While the performance is more steady over different initial step size values, RMSE does vary over different meta-step sizes.

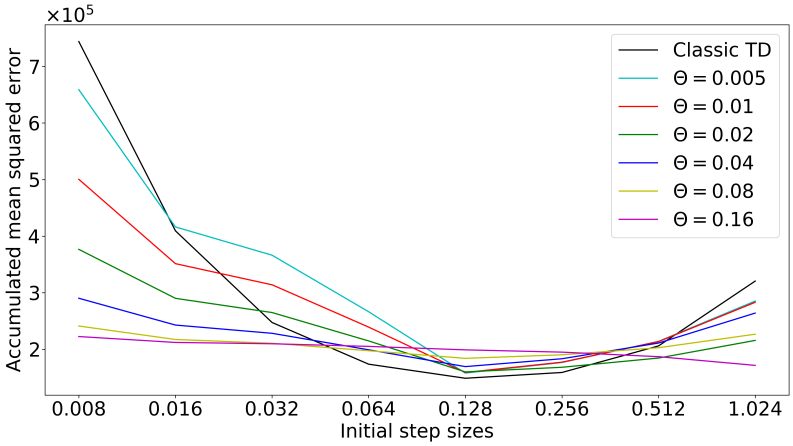
It can be seen that the highest error for both classic TD and OG AutoTIDBD occurs during the first phase (Rest 1). This can easily be explained by all GVFs being initialised without any knowledge about the sensor readings; the RMSE for the first time steps will therefore be high. Together, the results in this chapter not only support the usability of OG AutoTIDBD to autonomously learn and update step sizes for predictions without the need of human assistance, but also autonomously adapt the representation that is used for a Predictive Knowledge approach. As OG AutoTIDBD updates the step sizes based solely on interactions with the environment and is grounded in the observations that are received from said environment, it can truly function on its own—even when implemented in a long-lived application.

3.7.4 Experiment: stuck sensors

Within the realm of robotics, the state representation is often negatively impacted by damage to the sensors. In this section, I explore how OG AutoTIDBD

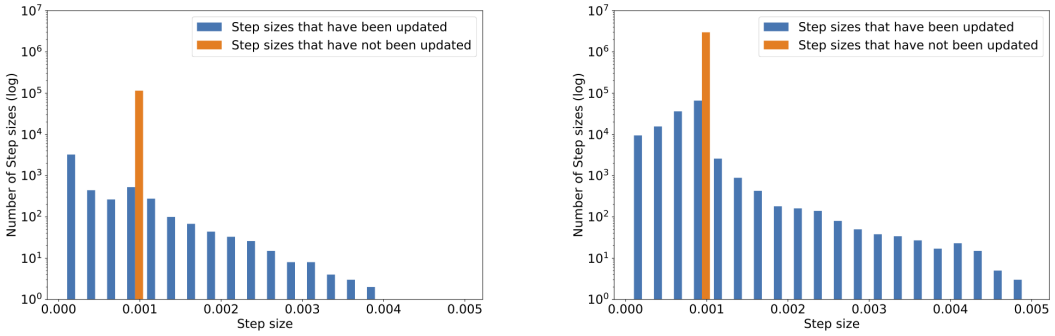


(a) Accumulated RMSE over all tested initial step sizes



(b) Zoom-in for selected initial step sizes

Figure 3.13: Accumulated RMSE over the experiment, depending on the initial step size. The first plot shows the overall accumulated error over the whole range of tested step sizes for TD and OG AutoTIDBD with different meta step sizes θ. While the performance of TD dramatically worsens for small step sizes, OG AutoTIDBD exhibits more consistent and better behaviour for different meta step sizes. Subplot (b) zooms in on larger step sizes to highlight the typical bowl-shaped performance line for TD. Although the error for TD is slightly smaller with carefully tuned step sizes, OG AutoTIDBD shows more robust performance with respect to the initial step sizes and the meta step sizes.



(a) Step sizes for the 4 elbow sensors when stuck

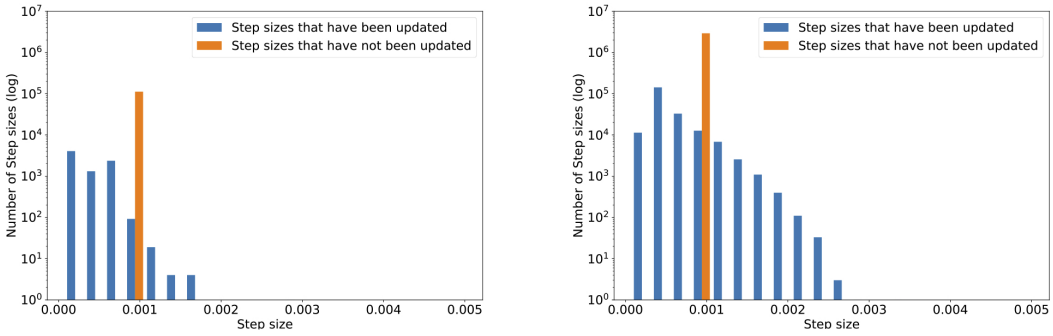
(b) Step sizes for the remaining 104 sensors

Figure 3.14: Step sizes distribution for the four elbow sensors (a) and the remaining 104 sensors (b), when the four elbow sensors are stuck. Step sizes increase noticeably in comparison to the original experiment. The largest step sizes are twice as big.

reacts to physical damage by examining performance on data with simulated broken and stuck sensors. In this section, I continue to use the behaviour demonstrated in the first experiment to generate data. In this experiment, the values of the elbow sensors are replaced with low-variance Gaussian noise, $\mathcal{N}(1, 0.5)$, to simulate them being stuck. The distribution of the adapted step sizes at the end of this experiment can be found in Figure 3.14.

In comparing Figure 3.14 with Figure 3.12, of particular note is the fact that, with simulated stuck sensors, some step sizes were adapted to be much larger than any adapted during normal operation; the maximum step size when all sensors were functioning was 0.0025, while Figure 3.14 shows step sizes of up to 0.005, approximately twice as large.

The step sizes for both the predictions with stuck sensor signals as their cumulants *and* for the remaining "unaffected" predictions increased in magnitude. At first, this result may seem counter-intuitive. For a constant signal with a small amount of noise, one could expect the step sizes to decrease, as such a signal does not contain a significant amount of information. In the setting at hand, this reaction is countered by the choice of representation. As the Kanerva coder prototypes were randomly distributed in space, the small amount of noise could be expected to constantly lead to different prototypes being activated. At the same time, the cumulants were nearly constant because



(a) Step sizes for the 4 elbow sensors when broken

(b) Step sizes for the remaining 104 sensors

Figure 3.15: Step sizes distribution for the four elbow sensors and the remaining 104 sensors.

Step sizes distribution for the four elbow sensors (a) and the remaining 104 sensors (b), when the four elbow sensors are broken. The step sizes for the four broken sensors are noticeably reduced compared to the experiment without broken sensors.

the variance was small.

This discrepancy between almost stationary cumulants and a changing representation appears to have led to increasing step sizes, since OG AutoTIDBD tried to achieve the necessary updates in fewer steps. Each feature was assigned a higher value, likely due to these updates being distributed over a wider range of features, resulting in higher step sizes. Although these increasing step sizes did not necessarily improve the representation, they are clearly distinguishable from step sizes that occurred during normal operation of the robotic arm, thus providing important information about sensor failure.

3.7.5 Experiment: broken sensors

The problems of broken sensors are common in robotics and of great interest in long-term autonomous systems. For the final experiment, I examine whether OG AutoTIDBD produces different step-size distributions when sensors emit noise consistent with a common physical failure. In this case, the four elbow sensors are replaced with Gaussian noise, $\mathcal{N}(0, 10)$, which corresponds to broken sensors that output noise. Such broken sensors do not contain meaningful information, as their output will be purely random.

Figure 3.15 shows the step size distribution averaged over all predictions

for the experiment with broken sensors that produce high-variance noise drawn from $\mathcal{N}(0, 10)$. Subplot (a) depicts the step size distribution for the four sensors that output noise. The maximum step size is only 0.0017. The step sizes observed during this experiment were considerably smaller than they were in the experiments where all sensors functioned normally. The average step size for broken sensors is 0.00037, while the average step size for these four sensors in the experiment with functioning sensors is 0.00065. Subplot (b) shows the distribution for the remaining 104 sensors. Although the maximum in this experiment, with a value of 0.0028, was almost identical to the maximum of 0.0025 in the experiment where all sensors function normally, there is a larger relative difference in the average step sizes. For the experiment with broken sensors, the average step size was 0.0006, while it was 0.00077 in the experiment where all the sensors function as expected.

As broken sensors are part of the feature representation $\mathbf{x}(s)$ that is used to estimate each GVF, the RMSE of functioning sensors will increase significantly. The RMSE for the 104 functioning sensors, given broken elbow sensors, was calculated for both OG AutoTIDBD learner and classic TD learner. The information provided by the elbow sensors was used in the feature representation $\mathbf{x}(s)$, but since these sensors are broken, they only provided irrelevant, distracting information to the predictors. For the classic TD learners, the RMSE for the 104 functioning sensors increased to 1.3×10^6. Step-size adaptation using OG AutoTIDBD resulted in a considerably lower RMSE of 5×10^5 in this experiment.

As expected, the step sizes corresponding to the four sensors that were replaced by noise decreased considerably compared to the step sizes during normal operation. Based on the interaction with these sensors, OG AutoTIDBD decreases the step sizes for the noisy sensors, reducing their influence on learned estimates. The step sizes for the remaining 104 sensors remained almost the same as in normal arm operation. However, the distribution of step sizes in the intact sensors changed slightly as more step sizes decreased in value, potentially to exclude features that correspond to the noisy inputs from impacting the predictions about the functioning sensor values. The RMSE for the remaining

104 sensors supports this intuition, as it is ∼2.5 times lower (1.3×10^6 for classic TD vs. 0.5×10^6 for OG AutoTIDBD) for OG AutoTIDBD than for classic TD.

3.7.6 Discussion on real-world experiments

In this section, I investigated OG AutoTIDBD through four different experiments to better understand how OG AutoTIDBD performs in a real-world setting. Experiments were performed with the Modular Prosthetic Limb (MPL)(Johannes et al., 2011; Johannes et al., 2020). All four experiments utilise data from alternating patterns of rest and movement. These four experiments resulted in three contributions. First, I demonstrated OG AutoTIDBD to be a practical alternative to an extensive step size parameter search. Second, I showed how OG AutoTIDBD can be used to detect and characterise common sensor failures. As a third contribution, I explored OG AutoTIDBD's sensitivity to its meta step size and to its initial step sizes in comparison to classic TD.

Experiment 3.7.2 I compared the predictive performance of classic TD with an extensive parameter search to the predictive performance of OG AutoTIDBD. The additional computation required by OG AutoTIDBD was still within reasonable limits for real-time computation, and the memory used for OG AutoTIDBD is negligible on modern systems.

The results show that OG AutoTIDBD and classic TD performed comparably in terms of root mean squared error (RMSE). Although there is a set of fixed step sizes for which classic TD exhibits slightly less error on the data set.

Experiment 3.7.3: I examined the accumulated RMSE for TD and OG AutoTIDBD when initialised with step sizes of different magnitudes. While the performance for TD is affected by the initial step size value, OG AutoTIDBD is less sensitive to its initial step sizes and its meta step size. This indicates that the usage of OG AutoTIDBD is more robust with respect to its initialisation, making it a viable alternative to a large search for parameters.

Experiment 3.7.4: I then explored the changes in the learning rates with several stuck sensors. The changes in the OG AutoTIDBD step sizes were clearly distinguishable from changes seen during normal functioning of the arm

(as explored in the first experiment), therefore providing an indicator to detect this type of sensor failure.

Experiment 3.7.5: I replaced several sensors with high-variance noise, simulating broken sensors. OG AutoTIDBD decreased the step sizes corresponding to the broken sensors, which resulted in these inputs being gradually excluded from the updates—it automatically learned the unimportance of these inputs.

These four results—the permanent updates of step sizes to accommodate non-stationarity, the distinct reaction to stuck sensors, the automatic feature selection for uninformative sensors, and the robustness with respect to its initialisation—are promising key features for long-term autonomous agents. They empower an agent not only to adapt its learning based on interactions with its environment, but also to evaluate and improve its own perception of the said environment. Furthermore, since step sizes contain information about the past for each feature, they can provide an important source of information for the agent itself to learn from. As argued before this work (Günther et al., 2018; Schultz et al., 1997; Sherstan et al., 2016), these introspective signals provide a helpful source of information to enable an agent to better understand its environment and its own functioning within its environment. The insights presented in this chapter provide a deeper understanding and intuition about the effects of OG AutoTIDBD, with the aim of helping other designers create agents that are capable of autonomous learning and adaptation through interaction with their environment.

3.8 Related Literature, Limitations, and Future Work

As discussed in Section 2.5.3, this thesis is not the first work to generalize IDBD to the Reinforcement Learning problem setting. There exist five other generalisations of IDBD to RL. In Dabney's thesis Scalar Incremental Delta-Bar-Delta (SID), Normalized Scalar Incremental Delta-Bar-Delta (NOSID), and a variant of AutoStep are introduced (Dabney, 2014). Each of these generalizations use the λ-return as the objective, use an ordinary-gradient, and adapt a single step-size shared across features. Concurrent with this thesis, Thill generalized IDBD to policy evaluation, using the λ-return as the objective, by taking a semi-gradient, and adapting a vector of many step-sizes (Bagheri et al., 2016). Similarly, IDBD was recently generalized to a policy-gradient variant called Meta-Trace (Young et al., 2019).

This thesis adds to the literature by performing a comparison of ordinary-gradient and semi-gradient derivations in situations where either a single shared step-size is adapted, or a vector of many step-sizes is adapted. Moreover, I explore the effects of learning feature-relevance using TIDBD. I demonstrated that TIDBD gives features with random noise smaller step-sizes, reducing their contributions to weight updates. In Section 3.6, I compare my vector-based IDBD methods against scalar step-sized methods including SID, NOSID, and AlphaBound (Dabney & Barto, 2012) to evaluate whether TIDBD's representation learning provides some benefit over methods without representation learning. Feature relevance was further explored in two common sensor failure settings: sensors with stuck values, and high-variance noise. In both cases, indications of failure were present when analysing the step-size values. Finally, this thesis explores IDBD-based step-size adaptation in non-stationary settings, including real-world robotic prediction tasks, giving us greater insight into the performance of IDBD-based methods outside of simulation. This chapter contributes to the literature by making new empirical comparisons, exploring the effects of feature relevance, and applying IDBD-based methods to real-world tasks.

The derivation of OG AutoTIDBD presented in this thesis is limited to methods that use linear function approximation. Moreover, all experiments presented in this chapter use binary features. Generalisations of IDBD for non-linear supervised learning methods exist (Schraudolph, 1999), and could be generalised to TD learning in the future. In addition, this generalisation is limited to on-policy predictions with replacing and accumulating traces. Further extension is required before OG AutoTIDBD can be used with off-policy prediction methods (see Chapter 5), and methods with different eligibility traces such as True Online TD (Seijen & Sutton, 2014).

Future research could explore alternative uses of learned step sizes. Step sizes learned with IDBD methods describe the relevance of a given feature to the task at hand. Similar to how examination of step-sizes indicated broken sensors, examination of step-sizes may aid in evaluating the potential of a prediction based on its given feature representation before it has been completely learned. Preliminary evaluation of predictions based on step sizes could be beneficial to prediction architectures such as Horde (Sutton et al., 2011), where large collections of predictions are proposed, learned, and maintained in real time as a learner interacts with their environment. In such situations, limited computational resources must be used effectively; being able to better identify promising predictions in early learning could support agents in selecting what predictions to learn.

Learned step sizes may also be an effective way to drive computational curiosity and intrinsic motivation (Linke et al., 2020). Many intrinsic motivation systems rely on metrics that drive exploration based on error on a given task (Oudeyer & Kaplan, 2009). One challenge of error-based motivation is differentiating between situations where the error is high because not enough learning has occurred and situations where the error is high because some signal or portion of the environment is not learnable. Learned step sizes, if used in combination with traditional error-based forms of intrinsic motivation, may be better able to differentiate between aspects of the environment are novel and aspects that are unlearnable.

3.9 Conclusion

In this chapter, I presented an approach to generalising Incremental Delta-Bar-Delta to temporal-difference learning. I extended TIDBD to AutoTIDBD, using normalization methods from AutoStep to improve the robustness of AutoTIDBD. Adapting step sizes with OG AutoTIDBD yields performance equal to or better than TD methods with a tuned static step size, even on stationary problems. On non-stationary tasks, I showed that OG AutoTIDBD can find appropriate step sizes and differentiate between relevant and irrelevant features. In a number of real-world robotic prediction tasks, I demonstrated that OG AutoTIDBD is less sensitive to choices of meta step sizes θ and initial step sizes $\boldsymbol{\alpha}_0$ than ordinary TD is to settings of $\boldsymbol{\alpha}$. OG AutoTIDBD performs as well as or better than TD with a tuned step size for broad a broad range of meta step size settings that are relatively invariant over prediction problems. AutoTIDBD based step size learning systems show promise of learning feature relevance and performing meta learning in an incrementally and online, lessening dependence on feature construction and parameter tuning.

Chapter 4

What's a Good Prediction? New Directions for Evaluating Agent Knowledge

Contributions of this chapter:

1. An analysis of accuracy as a means of determining usefulness of learned GVF estimates as features.
2. A proposal for evaluating the usefulness of GVFs as features which considers both the accuracy of estimates and the relevance of the features.
3. A generalisation of IDBD and AutoStep to off-policy learning to provide a measure of feature relevance.

In the previous chapter, I examined how meta-learning methods might enable an agent to modify how learning occurs independent of designer intervention. Now, let us turn our attention to *what* an agent learns. Our discussion of what agents should learn to predict begins with an examination of common evaluation methods used in Predictive Knowledge agents. I argue that contrary to popular belief, the accuracy of a prediction estimate is insufficient to determine whether a prediction estimate is *useful*. In this chapter, I demonstrate the limitations of existing approaches to evaluation: that strict measures of accuracy are insufficient for evaluating GVFs for use as inputs in further decision-making processes. In particular, I demonstrate the consequences of poor evaluation when GVF estimates are used as features for further learning

processes. Addressing these shortcomings, I introduce a new method of evaluating GVFs that takes into account the relevance of a GVF's features. To this end, I generalise AutoTIDBD to the off-policy policy-evaluation setting.

4.1 Introduction

Constructing general knowledge by learning task-independent models of the world can help agents solve challenging problems. However, the construction and evaluation of such models remains an open challenge. The most common approach to evaluating models is to assess their accuracy with respect to observable values. However, the prevailing reliance on estimator accuracy as a proxy for the usefulness of knowledge has the potential to lead us astray. This belief is rooted in how truth is understood within Predictive Knowledge: that an agent's beliefs about the world—its predictions—are true if they can be verified by comparing estimates to observed values (Ring, 2021; Sutton, 2011).

Such a verification stance is far from unusual. Throughout machine intelligence, researchers often use accuracy as a means of measuring performance and as a way of determining quality across many possible models (Barbieri, 2007; Russell & Norvig, 2010; Sutton, 1992). It is common to justify a particular learning method or architecture as superior to others by demonstrating how well it beats the state of the art on benchmarks (Bellemare et al., 2013; Deng, 2012; Krizhevsky & Hinton, 2009; Panayotov et al., 2015). Time has shown that when state-of-the-art systems are deployed and are finally evaluated via their use, there are unforeseen practical effects that are brought to bear. Computer vision systems can suffer from catastrophic misclassification when adversarial patches are applied to a stream (Brown et al., 2017). Permutations to an audio stream can similarly result in misclassification, while remaining imperceptible to human participants (Qin et al., 2019).

In this chapter, I demonstrate the conflict between accuracy and usefulness through a series of illustrative examples, including both a thought experiment and an empirical example in MineCraft, using the General Value Function framework (GVF). In particular, this chapter demonstrates that accuracy is not

a guarantor of usefulness. Having identified challenges in assessing an agent's knowledge, I propose an alternate evaluation approach that arises naturally in the online continual learning setting: I recommend evaluation by examining internal learning processes, specifically the relevance of a GVF's features to the prediction task at hand. This chapter contributes a first look into evaluation of predictions through their use, an integral component of Predictive Knowledge that is unexplored.

A cornerstone of intelligence is knowledge. It is no surprise that much Artificial Intelligence research has focused on designing algorithms that enable agents to construct knowledge of their world. In this thesis, I consider knowledge to be an agent's ability to conceptualise aspects of its environment by forming predictive models of its world. The term model is sometimes restricted to estimating the probability of state transitions; however, there are multiple approaches to building world models that enable agents to better perform on decision-making tasks (Barreto et al., 2017; Ha & Schmidhuber, 2018; Jaderberg et al., 2017). In this chapter, I take a broad view of what counts as a model, including predictions that forecast future input values an agent might experience (Koop, 2008; Ring, 2021). In this sense, agents construct knowledge of their world by learning to model and forecast aspects of the environment they inhabit.

The benefits of constructing knowledge by forecasting inputs are evident in computational Reinforcement Learning (Sutton & Barto, 2018), where an agent must learn to act optimally to maximise some expected cumulative future reward. Instead of directly finding the optimal policy, agents often learn the expected reward, or *value*, of states in their environment. When learning the value of a state, it becomes easier to determine what the optimal actions are.

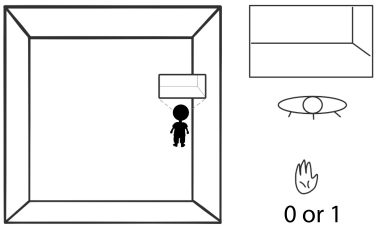
Value functions are deeply related to the problem of control, and the distinction between the main task (finding the optimal policy) and the model (estimating the value of a state) is subtle. However, modelling the environment does not need to end with estimating the value of states: modelling other aspects of the environment can also support decision-making (Comanici et al., 2018; Edwards, Hebert, et al., 2016; Günther et al., 2016; Jaderberg et al.,

2017; Koop, 2008; Sherstan et al., 2015; White, 2015). For example, it may be useful for an agent to estimate how different inputs change in response to its behaviour (Jaderberg et al., 2017): How an agent can control what it observes through its actions. These models of the world that are independent of a particular task or goal an agent is trying to achieve can be used flexibly on different problems, including new and unseen tasks (Barreto et al., 2017; Sherstan et al., 2018).

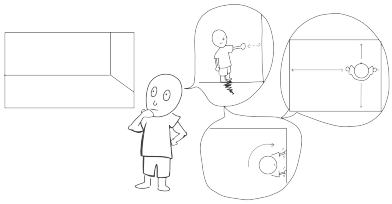
Learning models independent of the main task both supports agents in solving complex problems and forms the basis of general knowledge of the world that can be applied to new and unseen problems. How well an agent has acquired knowledge is often measured using quantitative metrics: for example, by directly measuring the accuracy of a model's estimate (Modayil et al., 2014; Pilarski & Sherstan, 2016; White, 2015), or by examining the reward received by an agent on the main task (Jaderberg et al., 2017). Systems with better quantitative results are believed to better encode knowledge on a particular task.

As the main contribution of this chapter, I demonstrate that evaluating knowledge is not the same as evaluating task performance: there are new challenges that need to be addressed. In particular, a model with higher estimated accuracy does not imply that the model better supports learning to solve the main problem, or task.

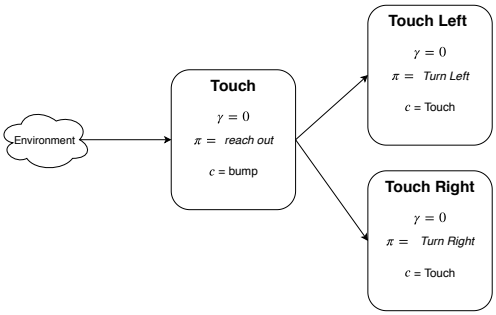
This distinction is introduced by constructing a series of examples and related experiments. In the first experiment, traditional evaluation techniques led to poor model choices. In the second experiment, these poor model choices result in poor performance when used to inform decision-making. Finally, the last experiment demonstrates that by examining internal learning processes, system designers can avoid some consequences of relying solely on measures of accuracy for model selection.



(a) An agent that cannot observe the true state of the environment; This agent in a room can only observe what it can see in front of itself and whether the agent bumped into something.



(b) Using limited sight and touch sensation, spatial awareness can be phrased as predictions about moving around the room: e.g., "can I touch something in front of me?", or "how many steps until I bump into a wall on my left"?



(c) A prediction about bumping is used to construct a `touch` prediction, the output of which is used as the target for the `touch-left` and `touch-right` predictions. Adapted from Ring (Ring, 2021).

Figure 4.1: Using the limited senses available to the agent, it must construct an abstraction such that it can understand a world it can never completely see. One way of constructing an agent's knowledge of the world is by predicting what would happen if the agent behaved in a certain way.

4.2 Understanding the World Through General Value Functions

The arguments apply broadly to evaluating machine learning models via accuracy and error alone. To focus this discussion, I ground the arguments in a single learning problem of interest: learning predictions as an agent interacts with its world. Predictions play an important role in the construction of knowledge for both artificial agents and biological intelligence. Humans and animals continually make many predictions about their sensations (Clark, 2013; Gilbert, 2009; Pezzulo, 2011; Pezzulo et al., 2013; Rao & Ballard, 1999; Wolpert et al., 1995). With this in mind, I use predictions to discuss the challenge of analysing knowledge in artificial agents.

As introduced earlier in this thesis, General Value Functions (GVFs) are a way for artificial agents to learn and make predictions incrementally and online, as an agent interacts with the environment (White, 2015). GVFs are entirely self-supervised and can be learned independent of the task an agent is undertaking through off-policy learning (Maei, 2011). In this chapter, I use GVFs as a computational tool to make the arguments presented clear, although the arguments presented are independent of GVFs themselves and broadly applicable to situations where models are evaluated independent of their use.

4.3 How GVFs are Specified and Learned

General Value Functions estimate the future value of a signal in a sequential decision-making process. On each time-step t an agent observes inputs $\mathbf{o}_t$ from the environment and takes an action a_t that results in a change in the environment, and thus a new observation $\mathbf{o}_{t+1}$. GVFs estimate the future accumulation of a *cumulant* c, where c is some signal of interest available to the agent through its subjective stream of experience. In the simplest case, this could be the accumulation of some element of an agent's observation $c \in \mathbf{o}$. The accumulation is discounted by a scalar value $0 \leq \gamma < 1$ and is conditioned on a particular policy π: the probability of taking action a_t given $\mathbf{o}_t$. The

discounted sum of future c is called the *return*, and is defined over discrete time-steps t as $G_t = \mathbb{E}_\pi[\sum_{k=0}^{\infty}(\prod_{j=1}^{k}(\gamma_{t+j}))c_{t+k+1}]$—the expectation of how a signal will accumulate over time.

When humans interact with the environment, they construct models of the world by constantly forecasting and anticipating what will happen next (Gilbert, 2009; Rao & Ballard, 1999; Wolpert et al., 1995). Similarly, an agent can build up self-supervised models that describe the environment through predictive questions such as "If I do this, I expect that" with General Value Functions (Comanici et al., 2018; Ring, 2021; White, 2015). An agent can express more abstract aspects of its environment with predictions by beginning with simple, primitive predictions about future features and interrelating them—making forecasts of forecasts. Such primitive predictions can inform more complex predictions in two ways: one prediction may be used as an input in another, or one prediction may be used as a cumulant c of another prediction. I refer to these predictions of another GVF's output as *higher-order* predictions. By interrelating predictions, agents can conceptualise aspects of the environment that extend beyond the immediate observation stream (Koop, 2008; Ring, 2021; Schlegel et al., 2021).

Predictions as knowledge are constructed by starting with low-level immediate predictions about sensation. For example, an agent may begin to build a model of spatial awareness by predicting whether there is something in front of it: if the agent reaches out, would it be able to touch something? This simple primitive prediction could be used to inform more abstract models: e.g., if the agent were to turn left or right, would there be something next to it? How long can the agent drive before it hits an obstacle? By interrelating predictive models, an agent can express more abstract, conceptual aspects of the environment (Comanici et al., 2018; Ring, 1997; Sutton et al., 1999; Veeriah et al., 2019) in a self-supervised way.

GVFs can be estimated using Temporal-difference (TD) learning methods (Sutton, 1988). TD learning estimates a value-function v such that $v(\phi(o_t)) \approx \mathbb{E}_\pi[G_t|\mathbf{o}_t]$: a function that estimates the return at a given time-step given the agent's observations. On each time-step the agent receives a vector of

observations $\mathbf{o} \in \mathbb{R}^m$. A function approximator $\phi : \mathbf{o} \to \mathbb{R}^n$—such as a neural net, Kanerva coder, or tile coder—encodes the observations into a feature vector. The estimate for each time-step $v(\phi(\mathbf{o}_t), \mathbf{w})$ function of learned weights $\mathbf{w} \in \mathbb{R}^n$, and the current feature vector—$v(\mathbf{o}_t, \mathbf{w}) = \mathbf{w}^\top \phi(\mathbf{o}_t)$.

The parameters that modify the learning process are *learning parameters*. Learning parameters change how the value function is approximated, but do not change the value function definition. Learning parameters include the step size α which scales updates to the weights, the eligibility trace decay λ and the function-approximator ϕ used to construct state.

4.3.1 The Challenge of Constructing Knowledge

It is impossible to know everything about the world. Certainly, an agent cannot predict everything about its world. One challenge for constructing models of the world is deciding of all the predictions an agent could learn to make, which subset can inform decision-making best. Not all predictions are created equally: two approximate GVFs may have the same question parameters—γ, π, and c–and yet produce very different estimates. Disparity in accuracy can be caused my many factors, including the learning parameters chosen, the distribution of experience trained on, feature construction, the step size parameter. Each factor contributes to how well an estimator can be learned.

4.4 Experiment 1: How Poor Evaluation Impacts Predictive Features

In this section, I construct a synthetic prediction problem and explore how common online error metrics can be misleading.

4.4.1 Evaluation by empirical return error

To choose between models, there needs to be a method to compare them. In long-lived domains, it is not feasible to compare GVFs to the true expected return of their cumulant c: an agent does not often have access to the true return from its stream of experience. Instead, GVFs are often assessed based

on an estimate of the true return, the *empirical return error*: the difference between the current estimate $v(\phi(\mathbf{o}_t))$ with an approximation of the true return (Edwards, Hebert, et al., 2016; Günther et al., 2016; Pilarski et al., 2012). The return is estimated by maintaining a buffer of length b of previous cumulants c, such that $\tilde{G}_t = \sum_{k=0}^{b}(\prod_{j=1}^{k}(\gamma_{t+j}))c_{t+k+1})$. The error for time-step t can then be constructed given the agent's experience by $\tilde{G}_t - v_t(\phi(\mathbf{o}_t))$. The empirical return error is not objective: it is calculated with respect to what the agent happens to experience and store in its buffer—it can only express the error for observations represented in the buffer, not the error for all possible observations or states of the world.

In simple Markov Reward Processes, this may not be an issue: maintaining a large enough buffer b will yield an error relatively unbiased over states. However, in many domains of interest, this is not possible: i.e., in robotics the state-space is often so immense that maintaining a buffer of observations would be a time-intensive and impractical demand. Instead, applications often settle for an empirical return error that covers only a portion of the state-space (Edwards, Hebert, et al., 2016; Günther et al., 2020; Günther et al., 2016; Pilarski & Sherstan, 2016). In doing so, some states are inherently prioritised over others, as they are split into two categories: the portions of state-space that are evaluated, and the portions that are not. When evaluating methods in this way, it is implicit that some states are privileged over others: that error matters more in one set of states over another.

4.4.2 A synthetic example

I present two hypothetical estimators of the same value-function in Figure 4.2 as an example of how empirical error can be gerrymandered by state. A binary square-pulse is the cumulant c for which two hypothetical value functions estimate the discounted return. The dotted line is the scaled return G_t of the cumulant c with a discount factor of $\gamma = 0.3$ that is being estimated. A perfect prediction will match the return G of the signal: rising before the signal of interest c rises, and falling before the pulse returns to 0. Such a value estimate is predictive—it forecasts the signal of interest.

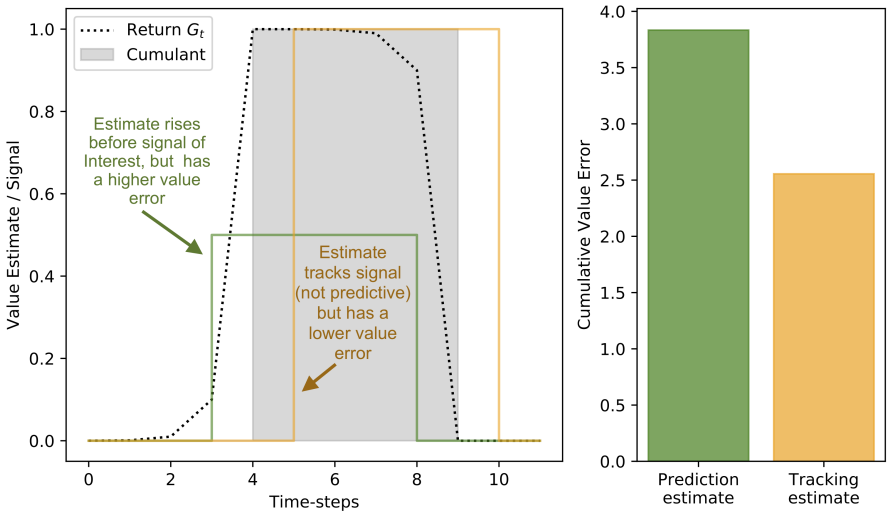


Figure 4.2: Two estimates of the same signal: one in green and one in orange. The cumulant c is indicated by the grey square pulse. The scaled return G_t of the cumulant is presented as a dotted line. Two hypothetical value function estimates of the return are presented in green and orange.

Two hypothetical value-functions are presented: 1) In orange, an estimator that tracks the cumulant by returning the last observed cumulant value; 2) in green, an imperfect but predictive estimate. The tracking estimator is not predictive: it rises and falls after the signal of interest. The predictive estimate does not exactly match the return being estimated, but rises and falls prior to changes in the underlying cumulant being estimated. While the tracking estimate fails to anticipate the square pulse, it has a lower empirical return error for the movement phases is presented. If a designer were evaluating the two predictions and choosing between these two estimators using prediction error alone, they would observe that the tracking estimator is superior to the predictive estimator: it has a lower cumulative error. This becomes an issue when these estimates are intended to inform decision-making. For instance, if an agent is predicting a collision, identifying the collision has occurred after the fact is not useful in supporting decision-making.

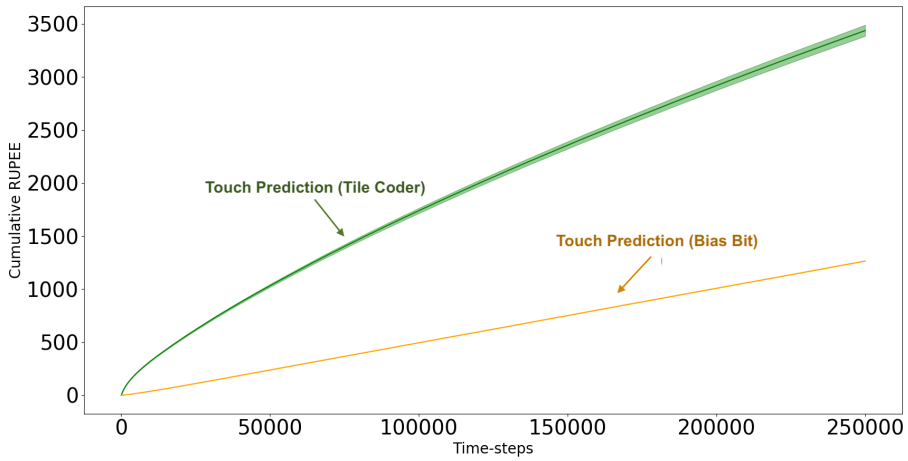
4.4.3 Experimental summary

While this synthetic example is contrived, there are many situations in which an agent would benefit from making such a prediction; being able to detect the onset of events is often useful in decision-making (Modayil & Sutton, 2014; Schlegel et al., 2021). For example, in the previous section, I worked out an example where an agent built a sense of spatial awareness (Figure 4.1) by predicting whether it could touch something in front of itself; In the spatial awareness example, touch is a binary signal that rises and falls, similar to this simple synthetic example. Such predictions are not made in a vacuum: the motivation for learning models is to use them to inform decision-making.

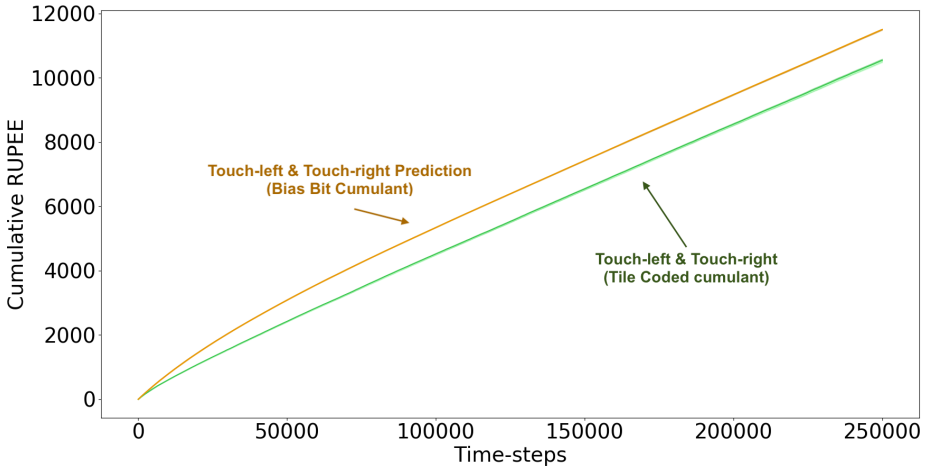
4.5 Experiment 2: How Performance is Impacted by Poor Predictive Features

With a simple example, the previous section demonstrated how accuracy can be misleading in differentiating between forecasts. Such forecasts are motivated by their use: using the learned estimates as either 1) predictive input features to another learning process, or 2) a signal of interest for further abstract predictions. I now discuss how dependence on accuracy negatively impacts downstream learning processes that use these learned estimates, and can critically undermine representation learning. To this end, I construct a network of interrelated predictions: a collection of predictions where a learned estimate is used to inform other learning processes.

The core motivation of learning models of the environment is to use such models to improve decision-making. The appeal of learning GVFs is the ability to build modular and hierarchical forecasts about the world—forecasts that can be used as predictive features for other learning processes. This is achieved by 1) using an estimate as an input feature when making a higher-order GVF, or 2) using a learned estimate as a cumulant for another GVF. In this section, I demonstrate that poor evaluation in lower-order GVFs has consequences for the performance of higher-order GVFs. To demonstrate these challenges in evaluation, let us turn our attention to the off-policy prediction setting.



(a) Cumulative RUPEE for tile-coded `touch` estimate (green) and bias-bit `touch` estimate (orange).The tracking estimate accumulates error at a slower rate than the anticipatory prediction. Evaluating based on RUPEE alone, it appears that the tracking model is best, despite leading to catastrophic prediction error when used to inform `touch-left` and `touch-right` (c.f. Figure 4.5). The anticipatory touch estimate has a greater accumulation of error throughout the experiment, despite being a better estimator for informing `touch-left` and `touch-right` predictions.



(b) Cumulative RUPEE for `touch-left` and `touch-right` estimates that use as a cumulant the tile-coded (green) and bias bit (orange) `touch` estimate. Estimates dependent on the tracking GVF for learning have a greater cumulative error than the GVFs dependent on the Tile Coder GVF. Error as accumulated at roughly the same rate as the anticipatory GVFs, making it challenging to distinguish which of the prediction is better, despite wildly different outcomes when comparing prediction to ground-truth (c.f. Figure 4.5). The error of the lower-order models does not always determine their effectiveness in informing further learning.

Figure 4.3: Cumulative Recent Unsigned Projected Error Estimate (RUPEE) over 250,000 time-steps for the 'touch-left' and 'touch-right' predictions averaged over 30 independent trials.

4.5.1 Estimating error for off-policy learning

Off-policy predictions are conditioned on a particular policy that may not be the agent's present behaviour policy. Though conditioned on a specific policy, off-policy GVFs can be learned while engaging in behaviours that do not strictly match the target policies of the prediction. Because the behaviour an agent is engaging in may not precisely match the policy an off-policy prediction is using for target behaviour, it is often not possible to compute the empirical return error. The buffer b collected from the agent's experience may represent experience induced by a policy different from the policy with which a prediction is specified; therefore, the return calculated from the buffer will not be representative of the off-policy return.

An off-policy error metric that can be calculated incrementally online is RUPEE: the Recent Unsigned Projected Error Estimate (White, 2015). RUPEE estimates the mean squared projected Bellman error of a single GVF. See White (2015) for an explanation of RUPEE on pages 119-122. Intuitively, RUPEE is an estimate of learning progress with respect to the input features used by the agent in learning. While RUPEE does not imply prediction accuracy, RUPEE provides a computationally efficient way to determine when a forecast learned off-policy is approaching its best estimate (White, 2015).

RUPEE requires an additional parameter $\beta_0 > 0$ which specifies a decay rate for the exponential moving average of both τ and $\overline{\delta e}$—an exponential moving average of the TD error and eligibility traces. See Sutton and Barto (2018) for a discussion of eligibility traces and TD(λ). A higher β_0 value results in a longer horizon for the moving average. Where e is the forecast's eligibility traces, δ is the TD error, and $\hat{h}$ is the same as the update in GTD(λ); RUPEE is estimated as follows:

$$\tau \leftarrow (1 - d_0)\tau + d_0$$

$$d \leftarrow \frac{d_0}{\tau}$$

$$\overline{\delta \mathbf{z}} \leftarrow (1-d)\overline{\delta \mathbf{z}} + \beta \delta \mathbf{z}$$

$$\text{RUPEE} \leftarrow \sqrt{|\hat{h}^\top \overline{\delta \mathbf{z}^d}|}$$

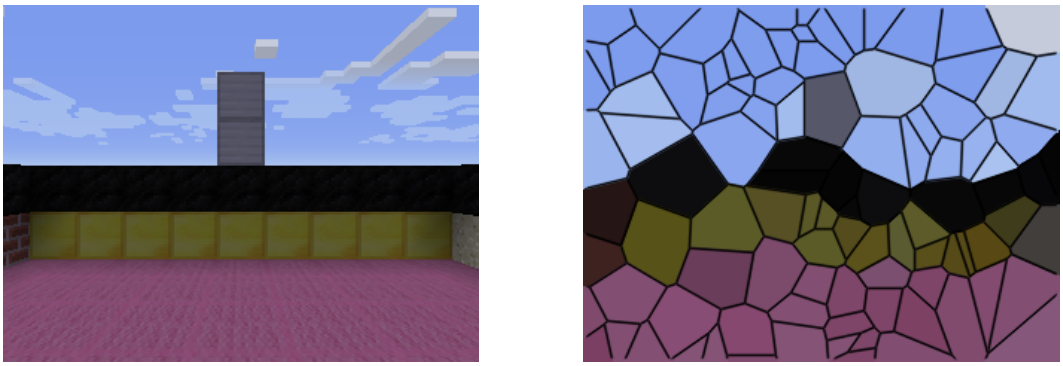
As was the case when evaluating on-policy predictions via empirical return error, estimating off-policy learning progress using RUPEE, is insufficient to differentiate between useful and useless estimators.

4.5.2 Predictions estimated

In the previous experiment, I demonstrated how using prediction error as a direct proxy for model quality can mislead. I now demonstrate how misevaluating the quality of GVFs can lead to poor performance in general. To do so, I introduce a network of predictions adapted from Ring's thought experiment on spatial knowledge (Ring, 2021), depicted in Figure 4.1. In this setting, the most basic GVF is `touch`: in plain terms, predict whether the agent would feel a surface if it extended its hand. Two natural higher-order predictions can be based on this: `touch-left` and `touch-right` (predict whether the `touch` GVF would activate if the agent turned left or right, respectively). Further higher-order predictions can build up to basic navigation and spatial awareness (Ring, 2021). However, to successfully build these concepts an agent must first get the simple, primary prediction right.

4.5.3 Experimental environment

These predictions are made in a MineCraft (Johnson et al., 2016) grid-world that reflects the spatial awareness task I previously introduced (Figure 4.1). This example is a simplification of the thought experiment introduced in Ring (Ring, 2021). The world is a square pen which is 30×30 and two blocks high. The mid-section of each wall has a silver column, and the base of each wall is a unique colour. On every time-step, the agent receives observations o_t which contain: 1) the pixel input from the environment (Figure 4.4a), and 2) whether the agent is touching something.



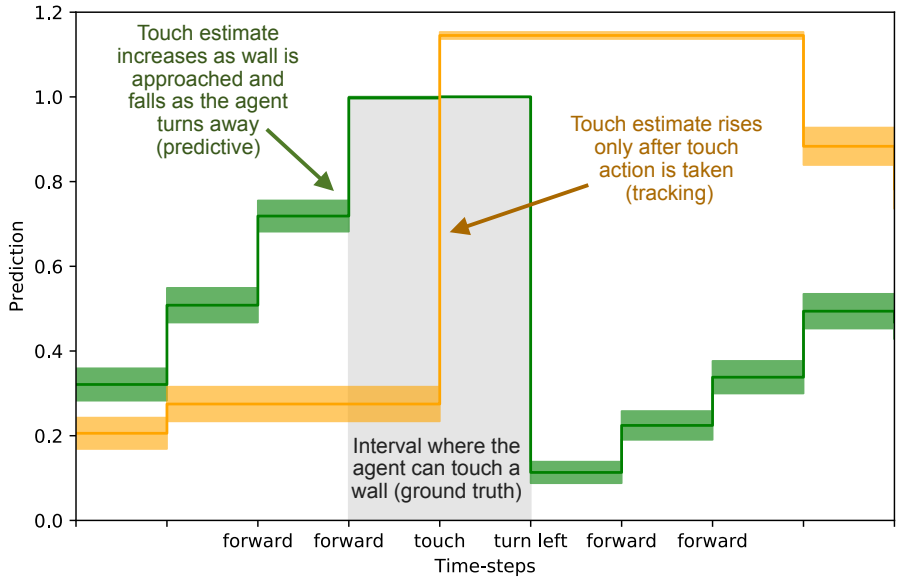
(a) The visual input of the agent. (b) Subsampling of 100 random pixels.

Figure 4.4: A visual representation of the agent's visual input by subsampling 100 random pixels.

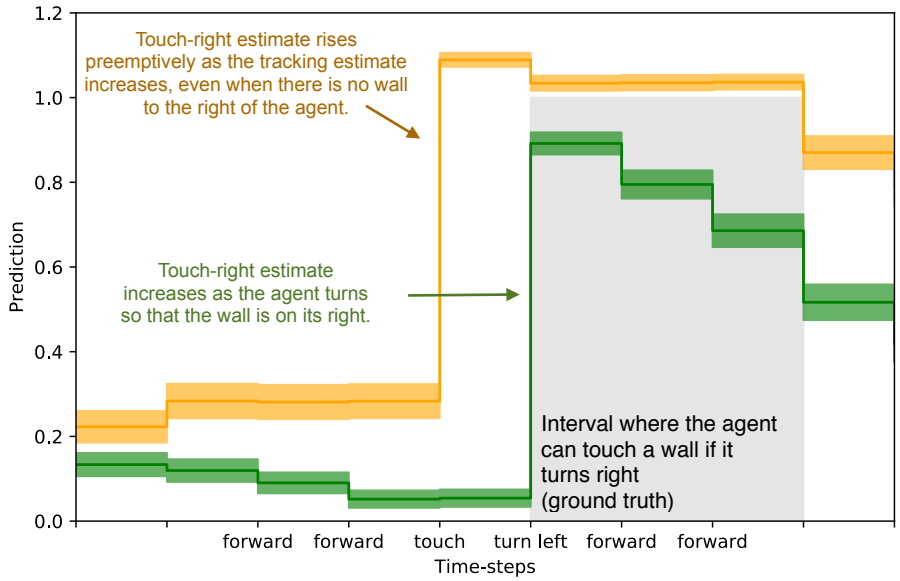
4.5.4 Results

Similar to the previous synthetic example, there are two sets of value functions: one that predicts, and one that tracks. The two GVF networks presented are specified with the same question parameters, but differ in answer parameters used. Both sets of GVFs are approximating the same values; however, the way they learn their approximation differs. One `touch` prediction uses a Tile Coder (Edgar An et al., 1991; Sutton & Barto, 2018) as a function approximator, and the tracking GVF uses only a single bias unit as a representation. This representation is chosen, as it is clear that a bias unit feature is insufficient to inform any of the chosen predictions: an agent cannot predict whether it can touch a wall using a constant feature to represent the MineCraft world.

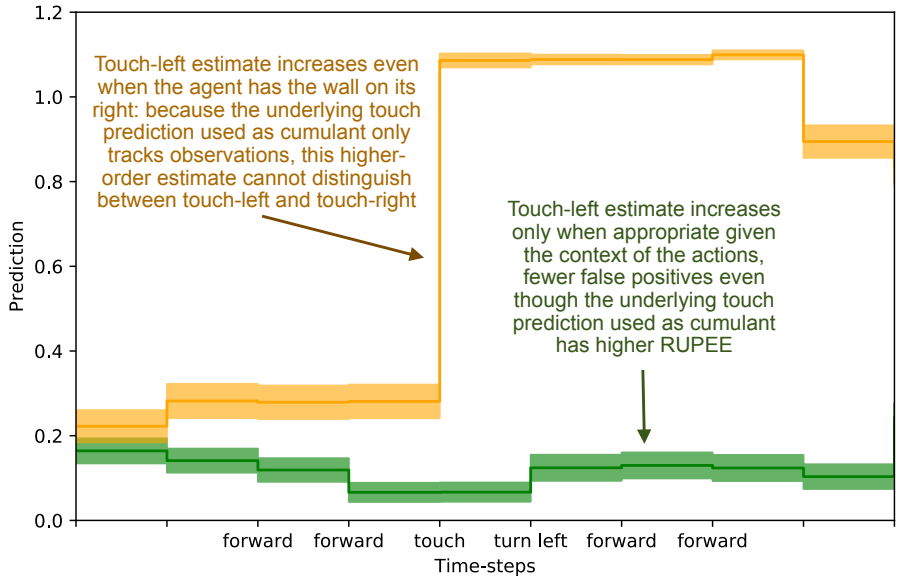
This experimental setup directly parallels the on-policy synthetic example in a more complex environment. As was the case in the previous thought experiment, by comparing the two `touch` predictions based on their error (Figure 4.3a), it appears that the bias unit GVF is superior to the tile-coded GVF—that the estimate that provides no predictive information is superior. When examining the actual predictions made by each GVF, it is clear that the prediction estimate with a greater RUPEE more closely anticipates the signal of interest (Figure 4.5a). The reason the prediction using a bias unit is not useful is that it tracks. An architect designing a system understands this prediction is poor because it is redundant: the immediate sensation of touch



(a) Tile-coded `touch` estimate (green) and bias-bit `touch` estimate (orange). Estimates are presented after having calculated the weight update for each time-step.



(b) `touch-right` estimates. Estimates are presented after having calculated the weight update for each time-step.



(c) `touch-left` estimates. Estimates are presented after having calculated the weight update for each time-step.

Figure 4.5: Each sub-figure depicts estimates of each of the GVFs in the agent's network for 150 examples of the agent approaching a wall and then turning left. 5 examples of the trajectory are drawn from 30 independent trials: results presented are averaged over 150 examples of the same trajectory.

tells us whether an agent *is* touching something. The intent of the prediction is to compactly express whether an agent *can* touch a wall without needing to engage in the behaviour. When the agent does touch a wall, the prediction is updated and stored in the weights of the GVF. Only when the agent is touching a wall will the bias unit GVF predict that it can touch a wall. By looking at RUPEE alone, this critical shortcoming is missed.

These predictions are not learned in a vacuum: the purpose of making the `touch` prediction, is to enable the higher-order predictions to be learned. In systems that use GVFs to construct an agent's knowledge of the world, these predictions are intended to inform further learning processes: either other value functions that describe more abstract aspects of the world, or the behaviours

an agent uses to accomplish its goals. Low RUPEE or low return error in an estimator does not necessarily equate to more useful predictions for these further decision-making purposes. The challenges of differentiating between a good and bad `touch` prediction have an impact that extends beyond the single prediction and influences the `touch-left` and `touch-right predictions`.

In this minecraft example, an agent benefits from not only an accurate `touch` prediction, but one which is capable of informing `Touch Left` and `Touch Right` predictions. In Figure 4.5, the RUPEE of `Touch Left` and `Touch Right` is depicted. There are two sets of these predictions: the first, using the bias bit GVF's prediction as its cumulant; the second, using the tile-coded GVF as its cumulant. In the second layer, the GVFs all share the same function approximator: they both use sufficient representations to learn a reasonable estimate. For the higher-order predictions, a random sub-sampling of the pixel input, binary touch signal, and `touch` prediction are all tiled together to construct the state for each GVF. The only differentiating factor is which cumulant is used: the prediction from either the tracking `touch` GVF, or the anticipatory `touch` GVF.

When examining the first layer's `Touch` predictions, the tracking GVF seemed superior based on RUPEE. When examining the RUPEE of the second set of predictions (Figure 4.3b), it is possible to glimpse the downstream effects of this misunderstanding. Although only slight, the GVFs dependent on the tracking `Touch` prediction have a higher RUPEE than those using the predictive `Touch` GVF. This point is brought into focus when examining the predictions made by each `touch-left` and `touch-right` prediction (Figures 4.5b and 4.5c). Examining average trajectories where the agent approaches a wall and turns left, the `touch-right` prediction using the tracking `touch` GVF as a cumulant (Figure 4.5b, in orange) rises and falls with its underlying GVF. The `touch-right` prediction with a tracking cumulant predicts the wall even before turning such that the wall is to its right, while the `touch-right` prediction with a predictive cumulant can better match the ground-truth. This disparity is further exacerbated in Figure 4.5c, where it is clear that the `touch-left` prediction dependent on the tracking `touch` GVF as a cumulant incorrectly

anticipates a wall is on its left, even as it turns away from it. Through examining the error—the metric used to inform Predictive Knowledge architectures—this is missed. The use of a prediction tells us more about the quality of that prediction than error alone. By using a poor underlying `touch` prediction, the higher-order GVFs become unlearnable.

4.5.5 Experimental summary

In the previous section, it was shown that poor behaviour of estimates can be hidden by commonly used error metrics. This kind of inquiry into the structure of predictions undertaken in the prior section is not easily automated: it relies on inspection by system designers. Moreover, these precise comparisons are limited to simple domains. The room that the agent inhabits is so simple that it is possible to acquire the ground-truth to examine the predictions as is done in Figure 4.5. In many domains of interest, this ease of comparison to the ground-truth is simply impossible. Each of these factors further frustrates the problem of determining what to learn, and whether particular GVFs are useful for informing decision-making.

Predictive Knowledge aims to be more than a niche of machine learning research. At its heart, Predictive Knowledge is a proposal about machine knowledge (Koop, 2008; Ring, 2021; Sutton et al., 2011; White, 2015)—as much epistemology as it is computing. If an agent's beliefs are predictions of sensorimotor inputs, then it is necessary to have a notion of whether such beliefs are true and can be considered knowledge. The prevailing suggestion is that a prediction is true—that an agent's beliefs can be considered knowledge—insofar as they are accurate (Ring, 2021; Sutton et al., 2011). More than an anaemic academic discussion, deciding what beliefs count as true is of great importance for machine knowledge. One must take care to ensure that as an agent is constructing knowledge, it is building from sound beliefs. In the preceding section, I highlighted the limits of accuracy as the arbiter of truth in machine knowledge, and the consequences that follow. In the following section, I propose a metric to fill this gap: evaluation of prediction usefulness by examining a value function's internal learning process.

4.6 Proposal: Evaluation of Feature relevance

I demonstrated that error in isolation of any additional information is misleading: empirical return error and RUPEE are insufficient to determine whether a model is useful for informing downstream decision-making by an agent. This inability to assess the usefulness of predictions is a major hurdle, the purpose of constructing knowledge is its use in supporting decision-making. If measures of accuracy verified using data available to the agent are not enough to assess the usefulness of a model, what should a designer do?

One need not only look at signals external to the agent for clues about performance: it is possible to look inwards and examine the learning process to assess an agent's knowledge—how the agent is modifying its parameters. Examining an agent's parameters is not unusual. For example, Unexpected Demon Error (UDE), can be used to gauge how 'surprising' a given observation is to an agent (White, 2015). By examining the surprise, it is possible to gauge how current experience relates to past experiences—e.g., detecting faults in a system (Günther et al., 2018).

There are many such parameters that an agent can modify during learning, and that modification can be monitored. Of particular interest are meta-learning methods: higher-order learning processes that modify the learning parameters of an agent (e.g., IDBD Sutton, 1992). One notable example is step size (learning rate) adaptation.

As discussed in the preceding chapter, IDBD-based step-size adaptation can be viewed as a form of *representation learning*. Representation learning describes how an agent encodes data or experience to support decision-making (Bengio et al., 2013). By assigning each individual input a specific step size, an input is weighted proportional to its relevance to some downstream learning task. For instance, AutoTIDBD assigns a step size α_i to each weight $\mathbf{w}_i$, adjusting the step size based on the correlation of recent weight updates. If many weight updates are made in the same direction, it would have been more efficient to make one large update with a larger $\boldsymbol{\alpha}_i$. If an update has over-shot, then the weight updates will be uncorrelated, and thus the step size should be

smaller.

All else being equal, a good model is one whose features are well aligned with the prediction problem at hand. Even in early learning where an agent is adjusting its model, or in situations where non-stationarity in the environment may introduce unexpected error, if the features are relevant to the prediction task can be expected to produce a reasonable estimate in expectation. One way to determine the relevance of features is by learning step sizes.

4.6.1 Derivation of off-policy Semi-gradient AutoTIDBD

To demonstrate how step sizes as feature relevance can be informative, I generalize Semi-gradient AutoTIDBD (Kearney et al., 2019) to GTD(λ), creating a step-size adaptation method suited for the off-policy `touch`, `touch-left`, and `touch-right` predictions I previously introduced. Off-policy AutoStep for GTD adds a few additional memory parameters to perform step-size adaptation. From here forwards, I refer to off-policy semi-gradient AutoTIDBD as SG AutoTIDB.

Here, I derive the relevant updates as follows. SG AutoTIDBD minimises the gradient of the squared TD error with respect to the meta-weight vector $\boldsymbol{\beta}$ that specifies the agent's step size on each time-step.

$$\begin{aligned}\boldsymbol{\beta}_{i,t+1} &= \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\frac{\partial\delta_t^2}{\partial\boldsymbol{\beta}_i} \\ &= \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\sum_j \frac{\partial\delta_t^2}{\partial\mathbf{w}_{j,t}}\frac{\partial\mathbf{w}_{j,t}}{\partial\boldsymbol{\beta}_i}\end{aligned} \tag{4.1}$$

$\frac{\partial\delta_t^2}{\partial\boldsymbol{\beta}_i}$ is expanded using the chain-rule. As in (Sutton, 1992), and prior chapters, the assumption is made that the effect of changing the step size $\boldsymbol{\alpha}_i = \exp(\boldsymbol{\beta}_i)$ for some feature $\phi_{i,t}$ will predominantly be on the weight $\mathbf{w}_i$.

$$\boldsymbol{\beta}_{i,t+1} \approx \boldsymbol{\beta}_{i,t} - \frac{1}{2}\theta\frac{\partial\delta_t^2}{\partial\mathbf{w}_{i,t}}\frac{\partial\mathbf{w}_{i,t}}{\partial\boldsymbol{\beta}_i} \tag{4.2}$$

TIDBD minimizes the squared TD error $\delta = c_{t+1} + \gamma v(\phi_{t+1}) - v(\phi_t)$, where c is the cumulant, γ is the discount factor, and v is the value function, and ϕ is

the state as constructed by a function approximator.

$$
\begin{aligned}
-\frac{1}{2}\frac{\partial \delta_t^2}{\partial \mathbf{w}_{i,t}} &= -\delta \frac{\partial[-v(\phi_{i,t})]}{\partial \mathbf{w}_{i,t}} \\
&= \delta_t \phi_{i,t}
\end{aligned}
\tag{4.3}
$$

$$
\boldsymbol{\beta}_{i,t+1} \approx \boldsymbol{\beta}_{i,t} + \delta_t \phi_{i,t} \frac{\partial \mathbf{w}_{i,t}}{\partial \boldsymbol{\beta}_i} \tag{4.4}
$$

I denote $\frac{\partial w_{i,t}}{\partial \boldsymbol{\beta}_i}$ as $\boldsymbol{\omega}$. GTD(λ) updates the weights as $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\alpha}[\delta \mathbf{z} - \gamma(1-\lambda)(\mathbf{z}^\top \mathbf{h})\phi_{t+1}]$. The update to $\boldsymbol{\omega}$ can be written recursively as follows:

$$
\begin{aligned}
\boldsymbol{\omega}_{t+1} &= \frac{\partial}{\partial \boldsymbol{\beta}} \Big[\mathbf{w}_t + \boldsymbol{\alpha}_{t+1}(\delta_t \mathbf{z}_t - \gamma(1-\lambda)\phi_{t+1}\mathbf{z}_t^\top \mathbf{h}_t) \Big] \\
&= \boldsymbol{\omega}_t + \delta_t \mathbf{z}_t \frac{\partial}{\partial \boldsymbol{\beta}}[\boldsymbol{\alpha}_{t+1}] + \boldsymbol{\alpha}_{t+1}\mathbf{z}_t \frac{\partial}{\partial \boldsymbol{\beta}}[\delta_t] + \boldsymbol{\alpha}_{t+1}\delta_t \frac{\partial}{\partial \boldsymbol{\beta}}[\mathbf{z}_t] \\
&- \frac{\partial}{\partial \boldsymbol{\beta}}[\boldsymbol{\alpha}_{t+1}]\gamma(1-\lambda)\phi_{t+1}\mathbf{z}_t^\top \mathbf{h}_t - \boldsymbol{\alpha}_{t+1}\gamma(1-\lambda)\phi_{t+1}\frac{\partial}{\partial \boldsymbol{\beta}}[\mathbf{z}_t^\top \mathbf{h}_t] \\
&\approx \boldsymbol{\omega}_t + \boldsymbol{\alpha}_{t+1}\delta_t \mathbf{z}_t - \boldsymbol{\alpha}_{t+1}\boldsymbol{\omega}_t \phi_t \mathbf{z}_t - \boldsymbol{\alpha}_{t+1}\gamma(1-\lambda)\phi_{t+1}\mathbf{z}_t^\top \mathbf{h}_t \\
&- \boldsymbol{\alpha}_{t+1}\gamma(1-\lambda)\phi_{t+1}\mathbf{z}_t^\top \frac{\partial}{\partial \boldsymbol{\beta}}[\mathbf{h}_t] \\
&= \boldsymbol{\omega}_t + \boldsymbol{\alpha}_{t+1}\Big(\delta \mathbf{z}_t - \boldsymbol{\omega}_t \phi_t \mathbf{z}_t - \gamma(1-\lambda)\phi_{t+1}(\mathbf{z}_t^\top \mathbf{h}_t + \mathbf{z}_t^\top \boldsymbol{\eta}_t) \Big)
\end{aligned}
\tag{4.5}
$$

In GTD(λ) the bias-correction update is $\mathbf{h} \leftarrow \mathbf{h} + \boldsymbol{\alpha}(\delta \mathbf{z} - (\mathbf{h}^\top \phi_t)\phi_t)$. Similar to $\boldsymbol{\omega}$, $\frac{\partial h_t}{\partial \boldsymbol{\beta}}$ is denoted as $\boldsymbol{\eta}$. The $\boldsymbol{\eta}$ update is as follows:

$$
\begin{aligned}
\boldsymbol{\eta}_{t+1} &= \frac{\partial}{\partial \boldsymbol{\beta}} \Big[\mathbf{h}_t + \boldsymbol{\alpha}_{t+1}(\delta_t \mathbf{z}_t - (\mathbf{h}_t^\top \phi_t)\phi_t) \Big] \\
&= \boldsymbol{\eta}_t + \frac{\partial}{\partial \boldsymbol{\beta}}[\boldsymbol{\alpha}_{t+1}]\delta_t \mathbf{z}_t + \boldsymbol{\alpha}_{t+1}\frac{\partial}{\partial \boldsymbol{\beta}}[\delta_t]\mathbf{z}_t + \boldsymbol{\alpha}_{t+1}\delta_t \frac{\partial}{\partial \boldsymbol{\beta}}[\mathbf{z}_t] - \boldsymbol{\alpha}_{t+1}(\mathbf{h}^\top \phi_t)\phi_t \\
&- \boldsymbol{\alpha}_{t+1}\frac{\partial}{\partial \boldsymbol{\beta}}(\mathbf{h}_t^\top \phi_t)\phi_t \\
&\approx \boldsymbol{\eta}_t + \boldsymbol{\alpha}_{t+1}\delta_t \mathbf{z}_t - \boldsymbol{\alpha}_{t+1}\boldsymbol{\omega}_t \phi_t \mathbf{z}_t - \boldsymbol{\alpha}_{t+1}(\mathbf{h}_t^\top \phi_t)\phi_t - \boldsymbol{\alpha}_{t+1}(\boldsymbol{\eta}_t^\top \phi_t)\phi_t
\end{aligned}
\tag{4.6}
$$

SG AutoTIDBD's three additional updates are now defined for GTD(λ) IDBD. This results in GTD IDBD.

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} + \theta\delta\phi_t\boldsymbol{\omega}_t \tag{4.7}$$

$$\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \boldsymbol{\alpha}\bigg(\Big(\mathbf{z}\big(\delta - \boldsymbol{\omega}_t\big) - \big(\mathbf{h} + \boldsymbol{\eta}\big)^\top\phi_t\Big)^\top\phi_t\bigg) \tag{4.8}$$

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\alpha}\bigg(\mathbf{z}\big(\delta - \boldsymbol{\omega}_t\phi_t\big) - \gamma\phi_{t+1}(1-\lambda)\mathbf{z}^\top(\mathbf{h} + \boldsymbol{\eta})\bigg) \tag{4.9}$$

To generalize AutoStep (Mahmood et al., 2012) to GTD(λ) requires two additions to GTD(λ): 1) a running average of meta-weight updates to prevent instability in the meta-weight vector caused by dramatic changes in the target of the underlying learning method, and 2) a normalization by the *effective step size* to prevent over-shooting on an individual example.

The effective step size describes the amount by which the error has been reduced on a particular example after a weight update. If the effective step size is greater than one, then the agent has over-shot on a particular example. To prevent over-shooting, the step size on each time-step is divided by $\max(1, \frac{\delta_t - \delta_t^+}{\delta_t})$, where is the TD error using the weights after taking a learning step $\delta_t^+ = c_{t+1} + \gamma v_{t+1}(\phi_{t+1}) - v_{t+1}(\phi_t)$. To find the effective step size, the following is simplified:

$$\begin{aligned}
\frac{\delta_t - \delta_t^+}{\delta_t} &= \frac{1}{\delta_t} - \Big[\Big(c_{t+1} + \gamma v_t(\phi_{t+1}) - v_t(\phi_t)\Big) \\
&\qquad \Big(c_{t+1} + \gamma v_{t+1}(\phi_{t+1}) - v_{t+1}(\phi_t)\Big)\Big] \\
&= \frac{1}{\delta_t}\Big[\Big(\gamma v_t(\phi_{t+1}) - v_t(\phi_t)\Big) - \\
&\qquad \Big(\gamma v_{t+1}(\phi_{t+1}) - v_{t+1}(\phi_t)\Big)\Big]
\end{aligned} \tag{4.10}$$

Which can be simplified to the resulting effective step-size:

$$\Big[\boldsymbol{\alpha}_{t+1}\mathbf{z}_t - \frac{\gamma(1-\lambda)\phi_{t+1}\mathbf{z}_t^\top\mathbf{h}_t}{\delta}\Big]^\top\Big[\phi_t - \gamma\phi_{t+1}\Big] \tag{4.11}$$

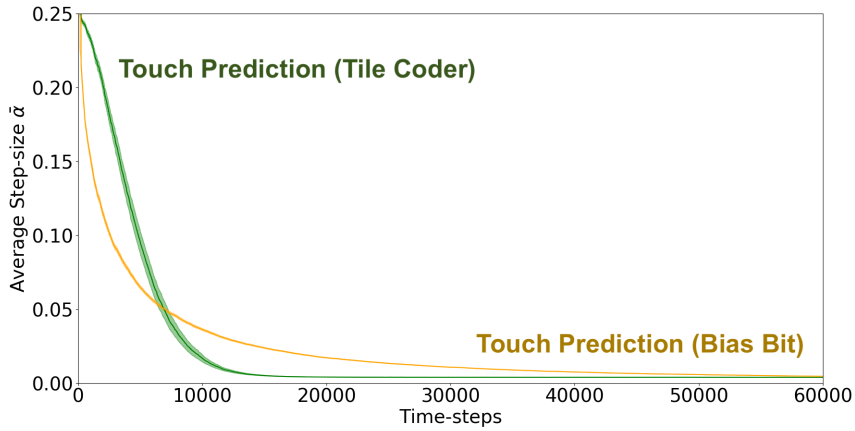
On each time-step IDBD updates the step sizes by $\delta\phi\boldsymbol{\omega}$. AutoStep takes a decaying trace of the IDBD's weight update, $\xi \leftarrow \max(|\delta\phi\boldsymbol{\omega}|, \xi + \frac{1}{\tau}\boldsymbol{\alpha}\phi\mathbf{z}(|\delta\phi\boldsymbol{\omega}| -$

$\xi)$), where τ is a parameter that specifies how quickly ξ decays. This has the effect of maintaining a decaying trace of the maximum update, such that a large change in the underlying learning target does not lead to instability in the step size parameter update.

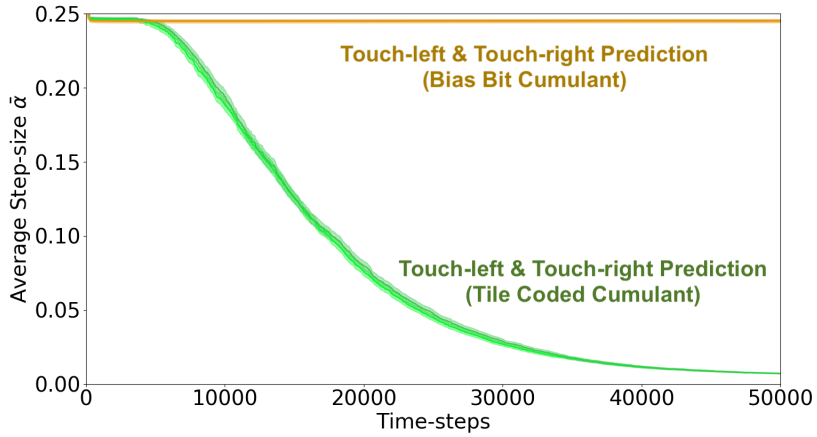
These updates can then be combined with the underlying GTD(λ) updates to produce SG AutoTIDBD(λ) (Algorithm 7).

Algorithm 7 SG AutoTIDBD: GTD(λ) with AutoStep step size tuning.

initialise:
  initialise vectors $\boldsymbol{\omega}$, $\boldsymbol{\eta}$, $\boldsymbol{\alpha}$, $\mathbf{z}$, ξ, and $\mathbf{w}$ of size n (number of features).
  Set τ as a decay value e.g., 10^4 and θ as a meta step size (e.g., 10^{-2}).
begin:
    Observe initial ϕ
    Take initial action a
repeat interaction with environment:
    Observe next state ϕ' and cumulant c.
    $\delta \leftarrow c + \gamma \mathbf{w}^\top \phi' - \mathbf{w}^\top \phi_t$
    $\xi \leftarrow \mathbf{max}: \big($
        $|\delta\phi\boldsymbol{\omega}|,$
        $\xi + \frac{1}{\tau}\boldsymbol{\alpha}\phi\mathbf{z}(|\delta\phi\boldsymbol{\omega}| - \xi)$
    $\big)$
    for $i = 1, 2, \ldots, n$: **do**
        if $\xi_i \neq 0$: **then**
            $\boldsymbol{\alpha}_i \leftarrow \boldsymbol{\alpha}_i \exp(\theta \frac{\delta\phi\boldsymbol{\omega}}{\xi_i})$
    $M \leftarrow \mathbf{max}: \big($
        $1,$
        $\Big[\boldsymbol{\alpha}\mathbf{z} - \frac{\gamma(1-\lambda)\phi_{t+1}\mathbf{z}^\top\mathbf{h}}{\delta}\Big]^\top \Big[\phi_t - \gamma\phi_{t+1}\Big]$
    $\big)$
    $\boldsymbol{\alpha} \leftarrow \frac{\boldsymbol{\alpha}}{M}$
    $\rho \leftarrow \frac{\pi(\phi,a)}{\mu(\phi,a)}$
    $\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\alpha}(\delta\mathbf{z} - \gamma(1-\lambda)\mathbf{z}^\top\mathbf{h}\phi')$
    $\mathbf{h} \leftarrow \mathbf{h} + \boldsymbol{\alpha}(\delta\mathbf{z} - (\mathbf{h}^\top\phi)\phi)$
    $\mathbf{z} \leftarrow \rho(\mathbf{z}\gamma\lambda + \phi)$
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \boldsymbol{\alpha}\Big(\mathbf{z}\big(\delta - \boldsymbol{\omega}\phi\big) - \gamma\phi'(1-\lambda)\mathbf{z}^\top(\mathbf{h} + \boldsymbol{\eta})\Big)$
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \boldsymbol{\alpha}\Big(\Big(\mathbf{z}\big(\delta - \boldsymbol{\omega}_t\big) - \big(\mathbf{h} + \boldsymbol{\eta}\big)^\top\phi\Big)\phi\Big)$
    $\phi \leftarrow \phi'$

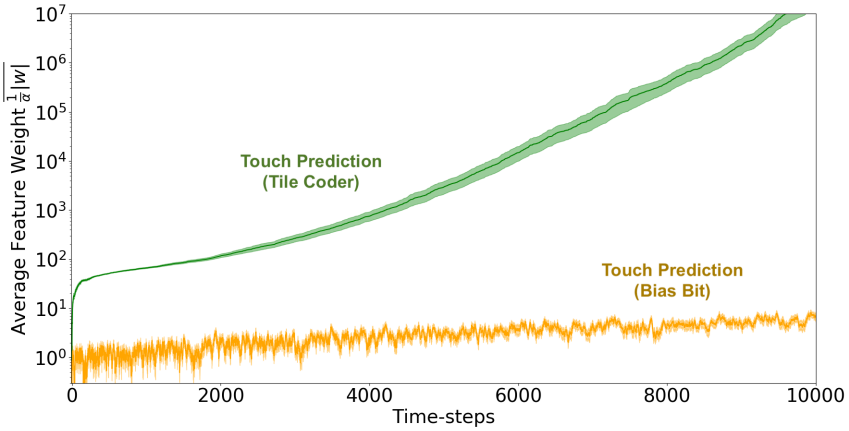


(a) The average active step size for both `touch` predictions. Anticipatory prediction in green; tracking-based prediction in orange.

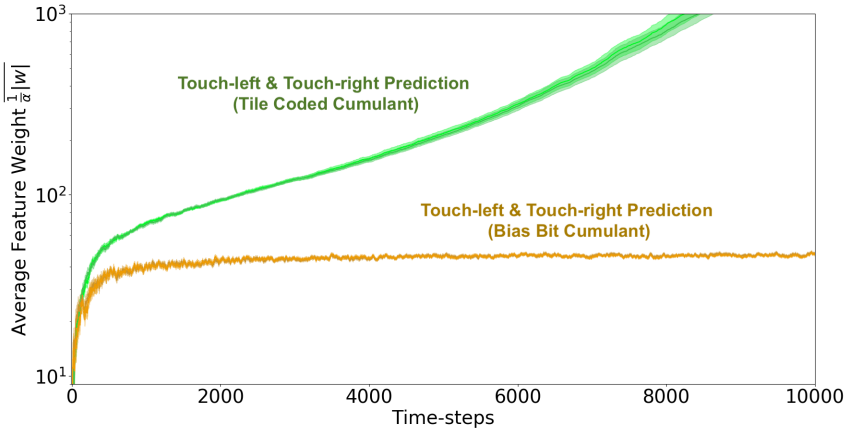


(b) Average active step size for the `touch-left` and `touch-right` predictions. Anticipatory predictions in green; tracking-based predictions in orange.

Figure 4.6: The average active step sizes for each layer of both the prediction and tracking networks averaged over 30 independent trials. Error bars are the standard error of the mean.



(c) Average weighted feature relevance $\overline{\frac{1}{\boldsymbol{\alpha}}|\mathbf{w}|}$ for `touch` predictions. Anticipatory tile-coded prediction in green; tracking bias-bit prediction in orange.



(d) Average weighted feature relevance $\overline{\frac{1}{\boldsymbol{\alpha}}|\mathbf{w}|}$ for the `touch-left` and `touch-right` predictions. Anticipatory predictions in green; tracking-based predictions in orange.

Figure 4.6: The average weighted feature relevance $\overline{\frac{1}{\boldsymbol{\alpha}}|\mathbf{w}|}$ for each layer of both the prediction and tracking networks. Each is run over 30 independent trials. Error bars are the standard error of the mean.

4.7 Experiment 3: Analysing Feature Relevance

Having generalised TIDBD to GTD(λ), I now return to the previously introduced MineCraft room domain and perform the same experiments, now using step-size adaptation.

4.7.1 Experimental setup

In Figure 4.6 the average active[1] step size value for the duration of the experiment is depicted. As was the case in the prior experiments, two agents are each learning three predictions: `touch`, `touch-right`, and `touch-left`. One agent has a representation sufficient to learn the underlying `touch` prediction with reasonable accuracy (green), while the other does not (orange).

4.7.2 Results: examining feature relevance

By examining the step size values, it is possible to visually differentiate between the tracking and predictive `touch-left` and `touch-right` predictions (Figure 4.6b); however, the corresponding `touch` predictions are not appreciably different upon examination of their step sizes late in learning progress (as shown in Figure 4.6a). Independent of learned weights, step sizes do not tell the full story; step sizes $\boldsymbol{\alpha}$ are a weighting of features ϕ when learning some weights $\mathbf{w}$. The learned step sizes $\boldsymbol{\alpha}$ in combination with the learned weights $\mathbf{w}$ give greater insight into the performance of a GVF. In Figure 4.6 a combination of the absolute value of the learned weights and step sizes are plotted: $\overline{\frac{1}{\boldsymbol{\alpha}}|\mathbf{w}|}$. The magnitude of the step size describes progress in learning, so $\frac{1}{\boldsymbol{\alpha}_i}$ is a measurement of learning progress for a particular feature. Intuitively, a feature which is stable, and thus has a small $\boldsymbol{\alpha}_i$, and has a relatively large weight $\mathbf{w}_i$ is preferable.

By examining the learned step sizes and weights $\overline{\frac{1}{\boldsymbol{\alpha}}|\mathbf{w}|}$, it is finally possible to separate the tracking and anticipatory `touch` predictions using an easily

[1]This agent's function approximator is a tile-coder. The tile-coder outputs a binary feature vector—only a portion of all features are active on a given time-step. The average absolute step size is multiplied by the number of active features so that two function approximators with differing active feature sizes will have equivalent scale and can be compared.

calculated metric (Figure 4.6d). As the step sizes decrease, the value of both the tracking and anticipatory predictions rises; however, since the magnitude of the weight $\mathbf{w}$ is low for the bias-bit, its weighted feature value remains low. This clarity in comparison carries over to the `touch-left` and `touch-right` predictions (Figure 4.6d). From Figure 4.6b, it is clear the tracking-based `touch-left` and `touch-right` predictions' step sizes never decay—the tracking predictions' step sizes maintain an average value of approximately 0.25 for the duration of the trials, while the anticipatory predictions' step sizes decay as the predictions are learnt. This results in a pronounced bifurcation between the two predictions. By looking at weighted features, it is possible to see and interpret what has been lost in the error estimate.

4.7.3 Conclusion

Using step sizes to inform other aspects of learning is a well-established practice. For instance, learned step sizes have been used for feature discovery (Mahmood & Sutton, 2013), and to drive exploration (Linke et al., 2020). Recent work has suggested that step sizes can be used to monitor the status of robots and indicate when physical damage has occurred in a system (Günther et al., 2020; Günther et al., 2019). Prior work in biological systems has shown there is more to representation learning than error minimization: for example, attention plays an important role in shaping how humans cognitively map their environment (Radulescu et al., 2019). This provides a suggestive interpretation of the benefits of adaptive step sizes. Moreover, using internal learning measurements to evaluate Predictive Knowledge systems has been suggested in other works (Sherstan et al., 2016); in this chapter, I provide a first exploration of how learned step-sizes may be utilised for introspection. Using the learning method generalised in this chapter, SG AutoTIDBD for GTD(λ), an agent can learn step sizes online and incrementally while interacting with the environment. In situations where traditional prediction error metrics fail, the magnitude of learned weights and step sizes enables differentiation between GVFs that are useful in informing further predictions, and GVFs that are not. In brief, GVFs can be evaluated in a meaningful, scalable way using feature relevance.

4.8 Relevance & Related Work

In this chapter, the arguments presented focus on a particular set of predictions in two experiments; however, the conclusions drawn apply to real-world applications of GVFs as well. From industrial laser welding (Günther et al., 2020) to autonomous vehicle navigation (Graves et al., 2021), error estimation is the means by which model quality is estimated prior to and during deployment. In situations like these where models are evaluated based on strict measures of accuracy, further decisions based on computed results are susceptible to the evaluation and performance issues raised in this chapter. While I focus on machine intelligence, similar observations about the primacy of prediction error have been made in cognitive neuroscience. For example, accuracy is not all that informs internal representations of location; additional factors such as attention also shape human spatial models (Radulescu et al., 2019). Moreover, attention has been used successfully to augment explainability when ML models are used for decision-making (Xu et al., 2015). I proposed that in general, solely considering estimates of model error are insufficient. While this chapter's discussion has focused in particular on applications of General Value Functions, I believe the conclusions drawn are not dependent on the learning methods themselves. The issues raised with respect to the use of models in decision-making transcend the learning methods discussed, and are relevant across all discussions of modelling in machine learning.

4.9 Contributions of This Chapter

In this chapter, I challenged a belief in Predictive Knowledge: that a prediction's accuracy is related to its usefulness. As a first contribution of this chapter, I demonstrated how strict measures of accuracy can be misleading. I further showed how critical areas of performance can be hidden by biased measures of error, leading to a poor choice of model. Building on this observation, I demonstrated how poor evaluation of learned models can lead to more serious errors in downstream learning tasks (e.g, prediction) that depend on these model estimates. As a final contribution, I proposed an alternative evaluation

approach that instead examines an agent's learned parameters as a basis for certifying learned knowledge, specifically focusing on learned weights and step size values. Using these additional sources of information, I showed that it was possible to differentiate between useful and useless models in a setting that was indistinguishable when using standard error or accuracy-based assessments. This chapter therefore contributes a novel look into how predictive models evaluation and use are related. Decoupling the evaluation of predictions from strict measures of accuracy is a key step towards building general, modular representations of knowledge.

Chapter 5

What Should An Agent Know? Online Discovery of Useful Predictions

Contributions of this chapter

1. An online incremental learning process by which agents can shape what predictions are learned for use as input features.
2. A demonstration that an agent with no prior knowledge of the environment can find predictions that provide useful inputs for decision-making in two partially observable environments.

In the previous chapter, I discussed the challenges an agent faces in choosing *what* to learn. In particular, I highlighted how an agent should evaluate predictions based on their usefulness for decision-making rather than their accuracy. In this chapter, I build on the insight of usefulness as a means of determining *what* to learn about by constructing a method of learning the question parameters of both off-policy and on-policy GVFs by meta-gradient descent while an agent is interacting with its environment.

5.1 Introduction

In computational Reinforcement Learning, a growing body of work seeks to construct an agent's perception of the world through predictions of future sensations (Comanici et al., 2018; Koop, 2008; Rafiee, 2018; Ring, 1994; Sutton, 2009; Sutton & Tanner, 2004; White, 2015); predictions about environment

observations are used as additional input features to improve goal-directed decision-making. An open challenge in this line of work is determining from the infinitely many predictions that the agent could possibly make which predictions support decision-making. This challenge is apparent in continual learning problems where a single stream of experience is available to a singular agent. As a primary contribution, I introduce a meta-gradient descent process by which an agent learns what predictions to make, the estimates for its chosen predictions, and how to use those estimates to generate policies that maximize future reward—all during a single ongoing process of continual learning. In this chapter, I consider predictions expressed as General Value Functions: temporally extended estimates of the accumulation of a future signal. I demonstrate that through interaction with the environment an agent can autonomously select predictions that reduce the impact of partial-observability, resulting in performance similar to expertly specified GVFs. By learning, rather than manually specifying these predictions, the agent can identify useful predictions in a self-supervised manner, taking a step towards fully autonomous systems.

It has long been suggested that predictions of future experience can provide useful and intuitive features to support decision-making—particularly in partially observable or non-Markovian environments (Jaeger, 2000; Littman et al., 2002; Wolfe et al., 2005). It is true for biological agents: humans and animals build predictive sensorimotor models of their world. These predictions of experience form the basis for biological perception (Gilbert, 2009; Rao & Ballard, 1999; Wolpert et al., 1995). A principled and well-understood way of making temporally extended predictions in computational Reinforcement Learning is by estimating many value functions. Value functions predict the long-term expected accumulation of a signal in a given state, and can predict not only reward, but any signal available to an agent via its senses (White, 2015).

In this chapter, I studied how an agent can autonomously choose GVFs to augment its observations to construct its own *agent-state*: an approximation of the environment state from the agent's subjective perspective.

An open challenge when using GVF estimates as input features is determining what aspects of an agent's experience to predict. Of all the possible predictions an agent could make, which subset of GVFs are useful to inform and support decision-making? The system designer often makes this choice (Dalrymple et al., 2020; Edwards, Hebert, et al., 2016; Günther et al., 2016; Modayil & Sutton, 2014). However, recent work has explored how an agent may autonomously specify its own GVFs. Previous work has explored *generate-and-test* approaches to specifying GVFs: an agent uses a heuristic to propose what predictions should be estimated and after a period of learning GVFs with a low perceived utility are replaced with new candidates (Schlegel et al., 2018).

Determining which GVFs to replace is a core challenge for generate-and-test approaches: A GVF may be accurate and have low prediction error, but just because a prediction is well estimated does not mean that it is useful as a predictive feature for control, as explored in the previous chapter. Examining a learned prediction estimate without considering its use—as is done in generate-and-test for GVF specification—inherently limits the ability of an agent to choose *useful* predictive features.

An alternative to random selection of GVFs is to parameterise the specification of a GVF and perform meta-gradient descent. By taking the gradient of a control learner's error with respect to a GVF's meta-parameters, what each prediction is about can be incrementally updated based on feedback from the control learner. Although not used for learning predictive inputs, recent work has shown early success in using meta-gradient descent as a means of learning meta-parameters that specify GVFs (Veeriah et al., 2019) for use as auxiliary tasks (Jaderberg et al., 2017).

When used as auxiliary tasks, GVF estimates themselves are not directly used in decision-making, but rather as regularisers for the control agent's artificial neural network. The auxiliary constrain the network to not only provide good action-value estimates, but also value estimates. The value estimates, or predictions, learned by the agent are not used in further decision-making, but are rather a by-product of the constraints placed on the network. In this auxiliary task setting, the parameters that determine what is being

predicted and the parameters that are used to select actions are explicitly kept and learned independent of one another. I propose a meta update where the core RL update of a control learner directly influences *what* an agent is predicting.

In this chapter, I integrate the discovery and use of GVFs for Reinforcement Learning control problems. I present a fully self-supervised approach, using meta-gradient descent to autonomously discover GVFs that are useful as predictive features for control. I do so by parameterising the functions that determine what aspect of the environment a GVF prediction is about, and constructing a loss that shapes the predictions based on the control agent's learning process. The resulting meta-learning method can be successfully implemented incrementally and online. By this process, agents can autonomously specify GVFs to be used directly as features by a control learner to solve two partially observable problems. The meta-learning process introduced in this thesis provides a new solution to a long-standing problem in using GVFs as predictive input features.

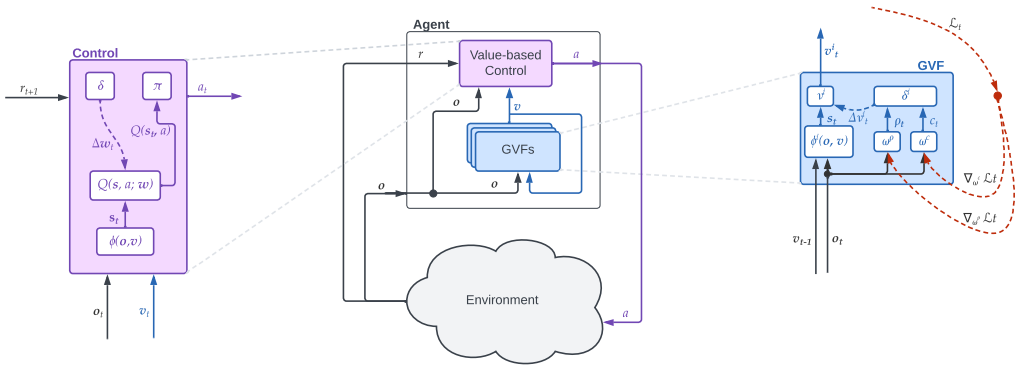


Figure 5.1: Depiction of meta-gradient agent structure. **Left:** the control agent updates its action-value function $Q(\boldsymbol{s}, a; \boldsymbol{w})$ according to the TD-error δ and chooses actions according to policy π based on value estimates. **Center:** The typical agent-environment diagram of Reinforcement Learning, where the control agent learns values as a function of both observations and GVF outputs. **Right:** a GVF outputs predictions according to its current weights $V(\boldsymbol{s}; \boldsymbol{\nu})$, while its updates are defined by the cumulant c, discount γ, and policy-correction ρ.

5.2 Learning What to Predict by Meta-gradient descent

The meta-learning process (Figure 5.1) operates on an agent structured in three parts :

1. a value-based control unit that learns weights $\boldsymbol{w}$ for an action-value function (Figure 5.1, left);

2. a collection of GVFs that each learn weights $\boldsymbol{\nu}$ to output prediction vector $\boldsymbol{v}$ (Figure 5.1, right); and

3. a set of meta-weight vectors $\boldsymbol{\omega}$ that parameterize each GVF's learning rule (the right half of Figure 5.1, right).

The control unit is a typical Q-learning agent, although it learns a value function over the *agent-state* $\boldsymbol{s}$, which is constructed from the observations $\boldsymbol{o}$ and a vector of GVF predictions $\boldsymbol{v}$, rather than observation alone. The vector of GVF predictions, along with the current observation, is transformed into the control agent-state using a differentiable function ϕ.

$$\boldsymbol{s}_t = \phi(\boldsymbol{o}_t, \boldsymbol{v}_t) \tag{5.1}$$

The approximate action-value function $Q(\boldsymbol{s}_t, a; \boldsymbol{w}_t)$ may be learned by any relevant RL algorithm.

The prediction vector $\boldsymbol{v}_t$ is output by a collection of GVFs. The i^{th} GVF has an agent-state $\boldsymbol{s}_t^i \leftarrow \phi^i(\boldsymbol{o}_t, \boldsymbol{v}_{t-1})$ independent of the control agent's. Here ϕ^i is the state transformation function for the i^{th} GVF. GVFs may use any Reinforcement Learning method for learning, but their structure is defined by three functions of the current state: the cumulant or target c, the discount or termination signal γ, and the policy π (c.f. Chapter 4 in White, 2015). The cumulant function $c_{t+1}^i \leftarrow z(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}_t^{c,i})$ determines the current target for GVF i: in classic RL, the cumulant simply selects the current reward r_t. The discount factor γ determines how far into the future the signal-of-interest should be attended to. While most commonly the discount is a fixed value, the discount

can be any function of the current state $\bar{\gamma}^i_{t+1} \leftarrow g(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}^{g,i}_t)$. The policy allows each GVF to condition its prediction on specific behaviours, and is used to compute the importance-sampling correction against the behaviour μ: $\rho^i_t = \frac{\pi^i(\boldsymbol{s}^i_t, a_t; \boldsymbol{\omega}^{\rho,i}_t)}{\mu(\boldsymbol{s}^i_t, a_t)}$. The output of a GVF is determined not only by the current weights $\boldsymbol{\nu}$ but also the functions that define its update procedure. Here, $\boldsymbol{\omega}$ refers to the collective parameters that define the GVF structure, disambiguated with superscripts when necessary.

During execution of the meta-learning process, each time-step contains an action and learning phase. First, the agent receives an observation from the environment $\boldsymbol{o}_t$, which is used to compute the GVF states.

$$\boldsymbol{s}^i_t \leftarrow \phi^i(\boldsymbol{o}_t, \boldsymbol{v}_{t-1}) \tag{5.2}$$

For each GVF i, the prediction value v^i_t is calculated as a function of the GVF state $\boldsymbol{s}^i_t$ and prediction weights $\boldsymbol{\nu}^i$. In this case, the value is a linear combination of the weights and the current state.

$$v^i_t = V(\boldsymbol{s}^i_t; \boldsymbol{\nu}^i_t) = \boldsymbol{\nu}^{i\top}_t \boldsymbol{s}^i_t \qquad\qquad v^i_{t+1} = \boldsymbol{\nu}^{i\top}_t \boldsymbol{s}^i_{t+1} \tag{5.3}$$

The vector of GVF predictions $\boldsymbol{v}_t = v^{1\cdots n}_t$, along with the current observation, is transformed into the control agent-state.

$$\boldsymbol{s}_t \leftarrow \phi(\boldsymbol{o}_t, \boldsymbol{v}_t) \tag{5.4}$$

The policy unit π uses $Q(\boldsymbol{s}_t, a; \boldsymbol{w}_t)$ to determine the next action[1]. For instance, the Q values may be defined as a linear combination of weights and state:

$$Q(\boldsymbol{s}_t; \boldsymbol{w}_t) = \boldsymbol{w}^\top_t \boldsymbol{s}_t \tag{5.5}$$

Once the action is executed and $(\boldsymbol{o}_{t+1}, r_{t+1})$ received, the learning phase begins.

The key to this meta-learning method is that the Q-learning error, as noted earlier (illustrated with the red lines in Figure 5.1, right), is a function of not only the value function weights $\boldsymbol{w}$, but also the agent-state vector $\boldsymbol{s}$.

[1] in the following results, the agent uses ϵ-greedy action selection.

The control learner's agent-state is constructed from the observations, and the GVF predictions. During the learning step, each prediction's weights are updated according to their question parameters: the cumulant $c^i_{t+1} \leftarrow z(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}^{c,i}_t)$, discount $\bar{\gamma}^i_{t+1} \leftarrow g(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}^{g,i}_t)$, and policy $\pi(\boldsymbol{s}^i_t, a_t; \boldsymbol{\omega}^\rho_t)$. The GVFs are updated according to $\delta^i_t \leftarrow c^i_{t+1} + \bar{\gamma}^i_{t+1} v^i_{t+1} - v^i_t$. Having updated the predictions, the control learner's agent-state at $t+1$ is $\boldsymbol{s}_{t+1} = \phi(\boldsymbol{o}_{t+1}, \boldsymbol{v}_{t+1})$, where $\boldsymbol{v}_{t+1} = v^{1\cdots n}_{t+1}$.

The control agent-state a function of the GVF estimates, which in turn are adjusted according to the meta-weight vectors $\boldsymbol{\omega}$ through the GVF's question parameters. Using weights from $t+1$ and the observations from t an auxiliary estimate $v'^{,i}_t \leftarrow V(\boldsymbol{s}^i_t; \boldsymbol{\nu}^i_{t+1})$ is used to form a new prediction vector $\boldsymbol{v}'_t = v'^{,1\cdots n}_t$ with which a loss is constructed to update the meta-weight vector based on the control agent's error: $\mathcal{L}_t = \delta^{C^2}_t$ where $\delta^{\mathrm{C}}_t = r_{t+1} + \gamma^{\mathrm{C}} \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{w}_t) - Q(\boldsymbol{s}'_t, a_t; \boldsymbol{w}_t)$. For the ith GVF, meta-weight vector $j \in \{c, g, \rho\}$ is adjusted.

$$\boldsymbol{\omega}^{c,i}_{t+1} \leftarrow \boldsymbol{\omega}^{c,i}_t - \alpha^{\mathrm{c}} \nabla_{\boldsymbol{\omega}^{c,i}} \delta^{\mathrm{C}^2}_t \tag{5.6}$$

Expanding the squared TD error without the non-differentiable max function yields the following

$$\boldsymbol{\omega}^{c,i}_{t+1} \leftarrow \boldsymbol{\omega}^{c,i}_t + \alpha^c \delta^{\mathrm{C}}_t \nabla_{\boldsymbol{\omega}^{c,i}} Q(\boldsymbol{s}'_t, a_t; \boldsymbol{w}_t) \tag{5.7}$$

$$\boldsymbol{\omega}^{\rho,i}_{t+1} \leftarrow \boldsymbol{\omega}^{\rho,i}_t + \alpha^\rho \delta^{\mathrm{C}}_t \nabla_{\boldsymbol{\omega}^{\rho,i}} Q(\boldsymbol{s}'_t, a_t; \boldsymbol{w}_t) \tag{5.8}$$

$$\boldsymbol{\omega}^{\gamma,i}_{t+1} \leftarrow \boldsymbol{\omega}^{\gamma,i}_t + \alpha^\gamma \delta^{\mathrm{C}}_t \nabla_{\boldsymbol{\omega}^{\gamma,i}} Q(\boldsymbol{s}'_t, a_t; \boldsymbol{w}_t) \tag{5.9}$$

Using the meta-weight vectors, each GVF computes the current ρ_t, c_{t+1}, and $\bar{\gamma}_{t+1}$, and updates its predictions weights $\boldsymbol{\nu}$. Full pseudocode is provided in Algorithm 8.

Algorithm 8 MGD self-supervised predictive Reinforcement Learning.

Choose hyperparameters: control agent's state function approximator ϕ, step-size α^C, discount γ^C, and exploration rate ϵ.
for all $i \in$ GVFs **do**
    Choose state approximator ϕ^i, eligibility trace factor λ and step-size α^ν for GVF updates; choose α^j for $j \in c, g, \rho$ for meta-learning updates.
Initialise learners:
Initialise Q-learning weights $\boldsymbol{w}_1$ # Q values defined in Eq'n 5.3
for all $i \in$ GVFs **do**
    Initialise weights $\boldsymbol{\nu}_1^i$, traces $\mathbf{z}_0^i$ and $\boldsymbol{\omega}_1^{j,i}$ for $j \in c, g, \rho$, and prediction v_0^i.
BEGIN: Receive initial observation $\boldsymbol{o}_0$; choose action a_0
$\boldsymbol{v}_0 \leftarrow v_0^{1\cdots n}$ # Collect initial value estimates in a vector
Take action a_0, observe r_1 and $\boldsymbol{o}_1$; choose a_1; skip updates such that $\boldsymbol{v}_1 \leftarrow \boldsymbol{v}_0$
$\boldsymbol{s}_1^{i\cdots n} \leftarrow \phi^{i\cdots n}(\boldsymbol{o}_1, \boldsymbol{v}_0)$ # Construct initial GVF states
$\boldsymbol{s}_1 \leftarrow \phi(\boldsymbol{o}_1, \boldsymbol{v}_1)$ # Construct initial control agent state
repeat For timestep $t = 1$ onwards:
    Take action a_t, observe r_{t+1} and $\boldsymbol{o}_{t+1}$
    # **GVF Value Update**
    for $i \in$ GVFs **do**
        $\boldsymbol{s}_{t+1}^i \leftarrow \phi^i(\boldsymbol{o}_{t+1}, \boldsymbol{v}_t)$ # Compute state for each GVF
        $c_{t+1}^i \leftarrow z(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}_t^{c,i})$
        $\bar{\gamma}_{t+1}^i \leftarrow g(\boldsymbol{o}_{t+1}; \boldsymbol{\omega}_t^{g,i})$
        $\rho_t \leftarrow \frac{\pi(\boldsymbol{s}_t^i, a_t; \boldsymbol{\omega}_t^{\rho,i})}{\mu(\boldsymbol{s}_t^i, a_t)}$
        $v_t^i \leftarrow \boldsymbol{\nu}_t^{i\top} \boldsymbol{s}_t^i$
        $v_{t+1}^i \leftarrow \boldsymbol{\nu}_t^{i\top} \boldsymbol{s}_{t+1}^i$
        $\delta_t^i \leftarrow c_{t+1}^i + \bar{\gamma}_{t+1}^i v_{t+1}^i - v_t^i$
        $\mathbf{z}_t^i \leftarrow \rho_t(\bar{\gamma}_{t+1}^i \lambda \mathbf{z}_{t-1}^i + \boldsymbol{s}_t^i)$
        $\boldsymbol{\nu}_{t+1}^i \leftarrow \boldsymbol{\nu}_t^i + \alpha^\nu \delta_t^i \mathbf{z}_t^i$
        $v_t'^{,i} \leftarrow \boldsymbol{\nu}_{t+1}^{i\top} \boldsymbol{s}_t^i$
    $\boldsymbol{v}_{t+1} \leftarrow v_{t+1}^{1\cdots n}$ # Collect value estimates in a vector
    $\boldsymbol{s}_{t+1} \leftarrow \phi(\boldsymbol{o}_{t+1}, \boldsymbol{v}_{t+1})$ # Compute current agent-state
    # **Control Agent Update**
    $\delta_t^C \leftarrow r_{t+1} + \gamma^C \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{w}_t) - Q(\boldsymbol{s}_t, a_t; \boldsymbol{w}_t)$
    $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t + \alpha^{\mathrm{c}} \delta_t^{\mathrm{C}} \nabla_{\boldsymbol{w}} Q(\boldsymbol{s}_t, a_t; \boldsymbol{w}_t)$ # Update control agent weights $\boldsymbol{w}_t$
    # **Meta-gradient Update**
    $\boldsymbol{v}_t' \leftarrow v_t'^{,1\cdots n}$
    $\boldsymbol{s}_t' \leftarrow \phi(\boldsymbol{o}_t, \boldsymbol{v}_t')$ # compute agent-state for MGD update
    $\delta_t'^{\mathrm{C}} \leftarrow r_{t+1} + \gamma^C \max_a Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{w}_t) - Q(\boldsymbol{s}_t', a_t; \boldsymbol{w}_t)$
    for $i \in$ GVFs **do**
        for each parameterized GVF component $j \in \{c, g, \rho\}$ **do**
            $\boldsymbol{\omega}_{t+1}^{j,i} \leftarrow \boldsymbol{\omega}_t^{j,i} + \alpha^j \delta_t'^{\mathrm{C}} \nabla_{\boldsymbol{\omega}^{j,i}} Q(\boldsymbol{s}_t', a_t; \boldsymbol{w}_t)$ # Update meta-weights
    Choose a_{t+1} according to $Q(\boldsymbol{s}_{t+1}, a; \boldsymbol{w}_{t+1})$, and exploration rate ϵ.

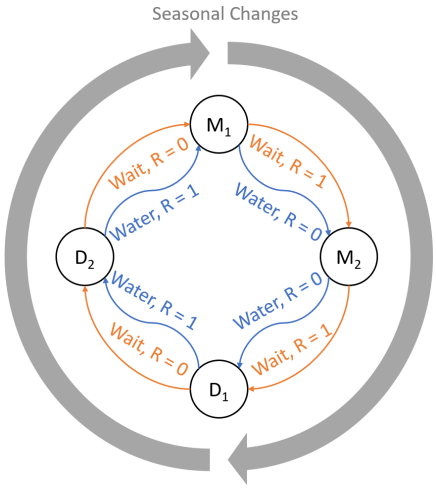


Figure 5.2: The monsoon problem. On each time-step the agent observes a binary value that determines whether the crops have grown. Growth of crops is determined by both the action an agent takes (to water crops or not), and the unobserved underlying season. There are four phases of the season that an agent can exist in: two monsoon and two drought (inner circles). The outer arrows indicate how the seasons change as the agent transitions through the cycle.

5.3 Can an Agent Learn What to Predict?

Using the meta process introduced can an agent find useful GVFs for use as predictive input features? I evaluate the meta specification of GVFs on a partially observable control problem, Monsoon World (Figure 5.2). Monsoon world is a small, clear example of a situation where temporal abstraction is necessary to solve the problem. This enables a clear assessment whether meta-gradient descent (MGD) is capable of specifying useful predictions, and a clear examination of the kinds of predictions the agent learns to specify.

In Monsoon World, there are two seasons: monsoon and drought. The underlying season determines whether the agent receives reward for its chosen action; however, the underlying season is not directly observable. Although the agent cannot directly observe seasons, it can observe the result of a given action: something impacted by the seasons. The agent tends to a field by choosing to either water, or not water their farm. Watering the field during a drought will result in a reward of 1; watering the field during monsoon season

does not produce growth and results in a reward of 0, and vice versa during a drought. If the agent chooses the right action corresponding to the underlying season, a reward of 1 can be obtained on each time-step. Regardless of the action chosen by the agent, time progresses and the agent transitions.

Agent	ϕ	ϕ^i	$\phi^{\boldsymbol{\omega}}$
Observations	`agg`	`agg`	-
Expert	`agg`	`agg`	-
MGD	`lin`	`agg`	`lin`

(a) Function approximators used for each agent component.

Agent	ϵ	α^Q	α^V	α^ρ	α^c
Observations	0.1	0.01	0.1	-	-
Expert	0.1	0.01	0.1	n/a	n/a
MGD	0.5	0.0001	0.1	0.001	0.1

(b) Hyperparameters for each agent.

Figure 5.3: Parameter settings for different agent configurations

This monsoon world problem can be solved, and an optimal policy found, if the agent reliably estimates how long until watering produces a particular result. Such estimates can be phrased as *echo GVFs* (c.f. Schlegel et al., 2021). Echo GVFs estimate the time to an event using a state-conditioned discount and cumulant. In plain terms, by estimating "How long until watering produces growth"? and "How long until not watering produces growth"? the agent can resolve the partially observable aspects of the environment. Indirectly, these capture the time until either the monsoon or drought. These two predictions can be described as *off-policy* estimates: predictions that are conditioned on a particular behaviour. Given the agent has two actions where a_0 is not watering and a_1 is watering, "if the agent waters" can be described as a deterministic policy $\pi = [0, 1]$. The signal of interest is, $c_{t+1} = 1$ if $r_{t+1} = 1$ & 0 otherwise. Similarly, a state-dependent discounting function terminates the accumulation, $\bar{\gamma}_t = 0$ if $c_{t+1} = 1$ & 0.9 otherwise. Off-policy GVFs can be estimated online, incrementally, while the agent is engaging in behaviours that do not strictly match the target policies of the prediction (Maei, 2011). Having constructed the

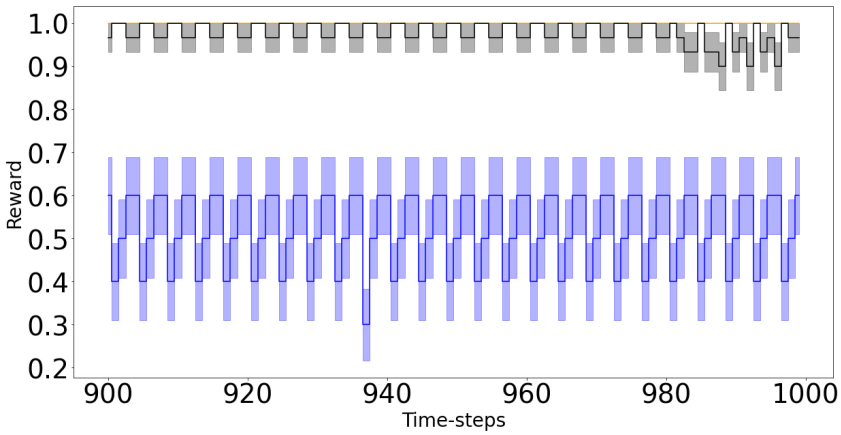


Figure 5.4: Three different learners that use 1) only the environmental observations as inputs (Agent Observations in blue), 2) two additional inputs that express the seasons (Agent Expert in orange), 3) two additional predictions that are updated using meta-gradient descent (Agent MGD in black). Each independent agent's reward is averaged over 30 independent trials. Error bars are standard error.

aforementioned GVFs, an agent can express what is hidden from its observation stream: how long until the next season. While no information was given about the season, the agent can learn about the seasons indirectly by making action-conditional predictions about its observations.

5.4 Learning to Specify GVFs in Monsoon World

GVF estimates can resolve the partial observability of monsoon world. Through MGD, can an agent specify such GVFs? I compare three different agent configurations (Figure 5.4): 1) a baseline agent that only receives environmental observations as inputs (in blue), 2) an agent that in addition to the environmental observations, two inputs that capture underlying seasons ('oracle', in orange), and 3) an agent that has two GVFs whose cumulants and policies are learned through meta-gradient descent (in black).

Learning by MGD to specify GVFs introduces two additional sets of meta-weight vector to initialise: weights $\boldsymbol{\omega}^{\pi}$ that specify the policy a prediction is conditioned on, and weights $\boldsymbol{\omega}^{c}$ that determine the signal of interest from the environment that is being learnt about. Policy weights $\boldsymbol{\omega}^{\pi}$ are initialised to an

a Log transform of prediction estimates

```
# Where v holds the value estimate from n GVFs.
# Where max γ_t is determined by the g function.

procedure TRANSFORM(v)
    v ← clip(log(v)/ log(max γ_t), 0, 1)
    return v
```

b State aggregation of predictions, `agg`

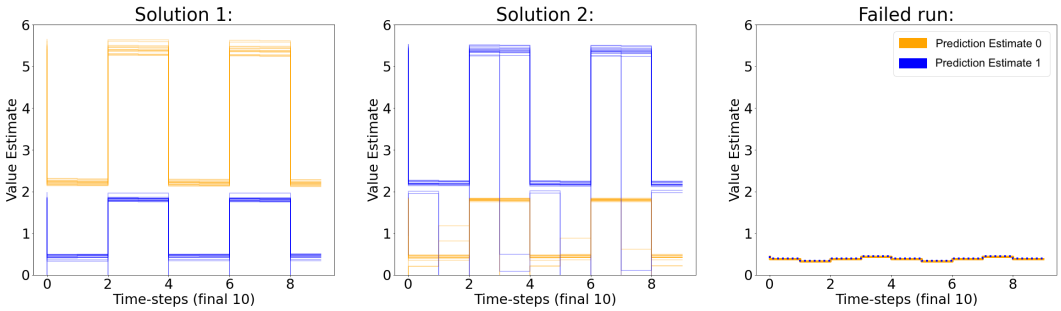
```
# Where v are the value estimates from n GVFs.
# Where v_i < 10
# Where memsize is the allocated length for the binary feature vector.

procedure STATE(v)
    s = zeros(memsize)
    i ← ⌊v_0 + v_1 * 10⌋
    s[i] = 1
    return s
```

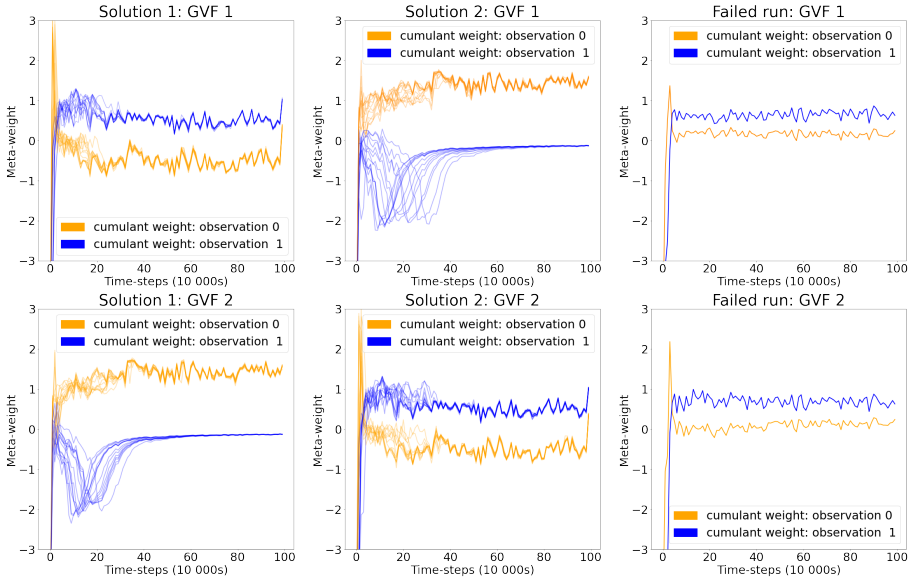
Figure 5.5: Standard function approximation steps for Echo GVFs.

equiprobable weighting of actions, and use a Softmax activation function $v_t^i = \texttt{softmax}(\boldsymbol{o}_t^\top \boldsymbol{w}_t^\pi)$ so that their sum is between $[1, 0]$. The cumulant meta-weight vectors $\boldsymbol{\omega}^c$ are initialised to -5, and a sigmoid activation $\texttt{sigmoid}(x) = \frac{1}{1+e^{-x}}$ is applied such that $c_{t+1} = \texttt{sigmoid}(\boldsymbol{o}_{t+1}^\top \boldsymbol{\omega}_t^c)$ bounding the cumulant between [0,1]. An L2 regulariser is applied to the meta-loss with $\lambda = 0.001$. The GVF's weights are initialised to 0. The control learner is a linear Q-learner that uses ϵ-greedy action selection with $\epsilon = 0.5$. The control agent's weights are initialised uniformly randomly between [0,1). Agents ran for a total of 1 million time-steps, and were evaluated during the final 1000 time-steps following a greedy policy with $\epsilon = 0$.

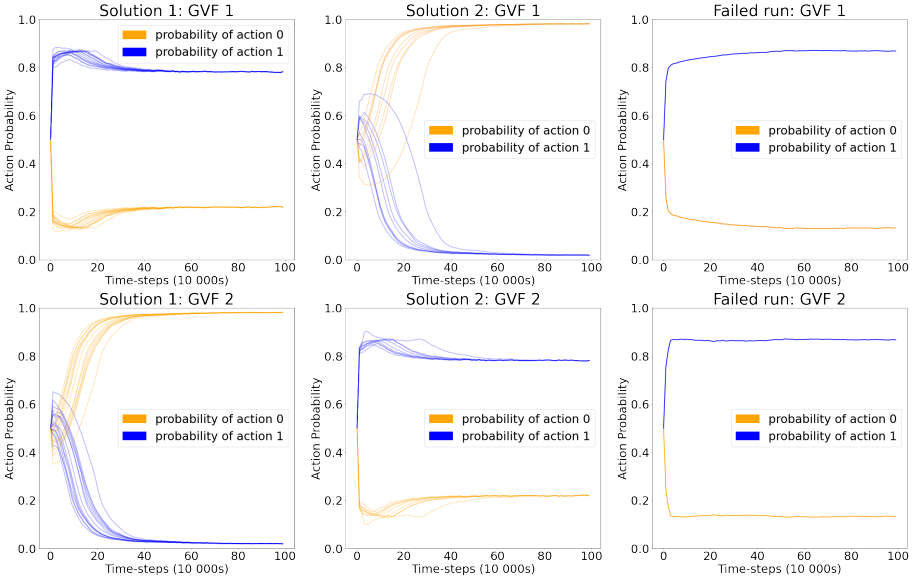
Different function approximators are used to transform the given inputs to an *agent-state* $\boldsymbol{s}_t^i = \phi(\boldsymbol{o}_t, \boldsymbol{v}_{t-1})$. Echo GVF estimates are in log-space (c.f. (Schlegel et al., 2021) for more information on echo GVFs). Before using an echo GVF's estimate as inputs, a log transformation is applied to them (see Algorithm 5.5a). A simple state-aggregation algorithm, `agg`is given in Algorithm 5.5b. Linear function approximation is abbreviated as `lin`.



(a) Value estimates from each GVF during all independent trials.



(b) The meta-weight vector ω_c for cumulants learned.



(c) The meta-weight vector ω_π during all independent trials.

Figure 5.6: Predictions learned, and the meta-parameters specified each GVF. There are two distinct GVF specifications that produced prediction estimates which enabled the agent to solve the problem.

Three agents are compared in the Monsoon environment: an agent that learns a value solely over the observations, **Observations**; an agent that learns over both observations and expert-specified GVFs, **Expert**; and an agent that learns over both observations and GVFs discovered through meta-gradient descent, **MGD**.

Each agent has a specific function-approximator for each of its components. All RL agents have a function approximator used by the control unit, ϕ. A GVF based agent also has a choice of function approximator for the GVF predictions, ϕ^i. Finally, the meta-gradient agents may also have a function approximator used for the meta-parameter update.

Parameters were chosen by performing a sweep across different values, choosing the best performing combination for each agent during the final 1000 evaluation steps in the experiment.

5.4.1 Meta-parameter specification

The GVF's policy π is a fixed policy: the meta-weight vector determines the policy a GVF is conditioned on, but they are not a function of the observations: $\pi \leftarrow \text{softmax}(\boldsymbol{\omega}^\rho)$. The cumulant c is a function of the observations, $c_{t+1} = \sigma(\boldsymbol{o}_{t+1}^\top \boldsymbol{\omega}_t^c)$, where $\boldsymbol{\omega}^c$ is the meta-weight vector for the cumulant, and $\boldsymbol{o}_{t+1}$ is the most recent environment observation.

In Figure 5.4, the average reward per time-step is plotted during the final 100 time-steps during which agent performance is evaluated given greedy behaviour. Although the agents deterministically follow their policy during the evaluation phase, learning still occurs during the evaluation phase: updates are made to the GVFs, which affect the input observations to the control agent (in the case of the MGD agent), and the control learner continues to update its action-value function. This continued learning accounts for irregularity in the oscillations.

The policy learnt using only environment observations similar to randomly choosing an action: the learned policy is no better than a coin-toss (Figure 5.4, depicted in blue). This is as expected, given observations alone are insufficient to determine the optimal action on a given time-step. When the underlying season is provided as input (depicted in orange), the learned policy is approximately

optimal. By augmenting environmental observations with predictive features that estimate the time to each season, an agent can solve the problem. Using meta-gradient descent, the agent can solve the task with performance on-par with the hand-crafted solution without being given what to predict.

5.4.2 What GVFs are specified by meta-gradient descent?

If an agent can find GVFs to solve the monsoon world, what are the useful predictions the agent found and are they the same as expert specified GVFs? In Figure 5.6a, the value-estimates on the final 10 time-steps of each run are presented, as well as the meta-weight vector for the cumulants (Figure 5.6b) and policies (Figure 5.6c). Two distinct types of policy and cumulant were learned for each of the two GVFs. In one of the 30 independent trials, the agent failed to solve the problem; the learned meta-weight vector of this failed trial is depicted independently. The third column illustrates how different the relationship of the two value estimates are in this failure case (from both successful and expert-specified GVFs). In the remaining 29 of 30 runs, the agent succeeds.

Over the course of successful runs, GVFs found by MGD do not look exactly like the echo GVFs introduced in Section 5.3. Some characteristics are similar: i.e., one of the policies approaches $\pi \approx [1, 0]$; however, the other policy $\pi \approx [0.8, 0.2]$ looks different from the deterministic policies. Even when the value estimates output by the self-supervised GVFs are similar to those from expert-specified GVFs, the cumulant and policies can be quite different.

For the best parameter settings in the sweep conducted, one of 30 independent trials failed, achieving an average reward per-step of 0.5 (note this performance is similar to that of the agent with only environmental observations). For this failed run, the learned policy and cumulant do not fit the categorisations of cumulants or policies learned in successful trials. Importantly, the learned value estimates presented to the control agent as features do not share the same cyclic values that capture the underlying seasons of the environment. From this failed run, it is clear that simply adding *any* prediction does not enable the agent to solve the problem: in successful trials, the policies and

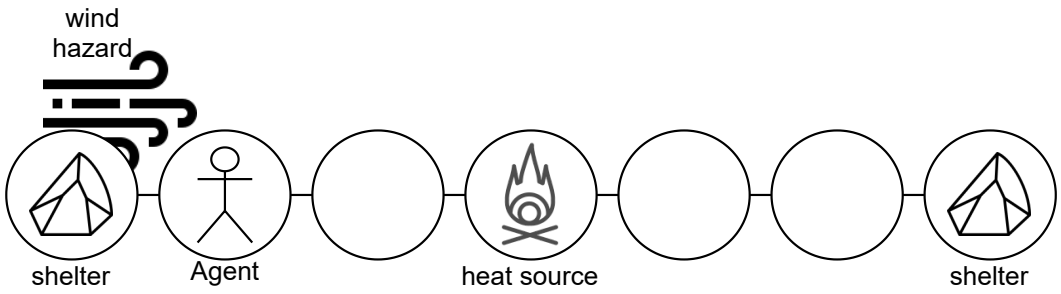


Figure 5.7: A depiction of the frost hollow problem. Frost hollow is a linear walk where an agent collects a unit of heat by standing at the fire in the centre state. Once the agent accumulates 6 units of heat, it receives a reward of 1. The only way to receive reward in frost hollow is by accumulation of internal heat points that are converted in to reward once the threshold of 6 heat units is reached. Every 8 time-steps, a wind hazard gusts for two consecutive time-steps, removing all the agent's accumulated heat if it is exposed to the hazard. To avoid losing its heat, the agent can take shelter in either of the end states. On each time-step, the agent observes its own location, the amount of heat it has accumulated, and whether the wind hazard is present.

cumulants learned by MGD are meaningful and specific to the environment and enable the agent to solve the problem.

5.5 Learning to Specify GVFs in Frost Hollow

The previous example explored whether using MGD an agent could learn to specify predictions to resolve the partial-observability of its environment. In this section, two additional complications are added: 1) instead of a linear control agent, a more complex function approximator is used; 2) the agent is in a domain with sparse reward, complicating the GVF specification process. Frost Hollow (depicted in Figure 5.7) (Brenneis et al., 2021; Butcher et al., 2022) was first proposed as a joint-action problem where a learned GVF is passed as an input feature to another agent, making it well suited for assessing whether via MGD an agent can autonomously choose what GVFs to learn.

Agent	α^V	α^Q
Observations	n/a	n/a
Expert	0.001	n/a
MGD	0.001	0.0001

Table 5.1: Best parameter settings for different agent configurations

Agent Type	**Cumulative Reward: Evaluation Phase**
Environment observations	7 ± 2.9
Expert Specified GVF	3.3 ± 1.6
GVF specified via MGD	18.7 ± 4.2

Figure 5.8: Average cumulative reward and standard error of the mean during final 1000 evaluation steps for best configuration of each agent. The mean and standard error of the mean are reported over 30 independent trials. The maximum possible cumulative reward is 50.

While simple, Frost Hollow poses a difficult learning problem. The reward is sparse: an agent can only observe a reward after successfully accumulating heat and dodging regular hazards. It takes at a minimum 50 time-steps, or 5 successive cycles of dodging the hazard successfully, before the agent can acquire a single reward. While the hazard itself is observable to the agent when active, the agent must preemptively take shelter before the hazard's onset to avoid losing its accumulated heat. All of this must be learnt from a sparse reward signal. Learning an additional GVF and using its estimate as an additional predictive feature can enable both humans, and value-based agents to successfully gain reward (Brenneis et al., 2021; Butcher et al., 2022).

In the frost-hollow setting, the control agent receives a single *on-policy* prediction as an additional input feature: the GVF is conditioned on the agent's behaviour rather than a policy specified by MGD. The discount γ is a constant value of 0.9, following Butcher et al. (2022). The meta-gradient learning process is compared to two baselines: one where the control agent

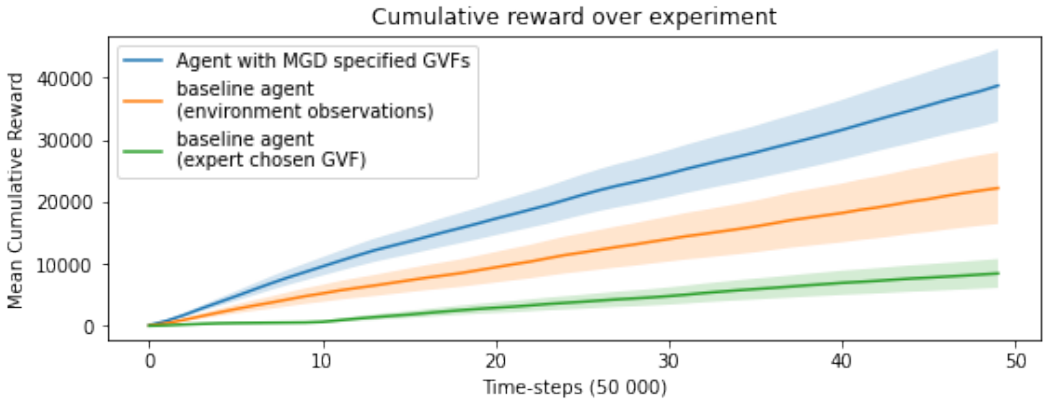


Figure 5.9: The mean cumulative reward for each agent during all of learning in Frost Hollow. Error bars are the standard error of the mean over 30 independent trials.

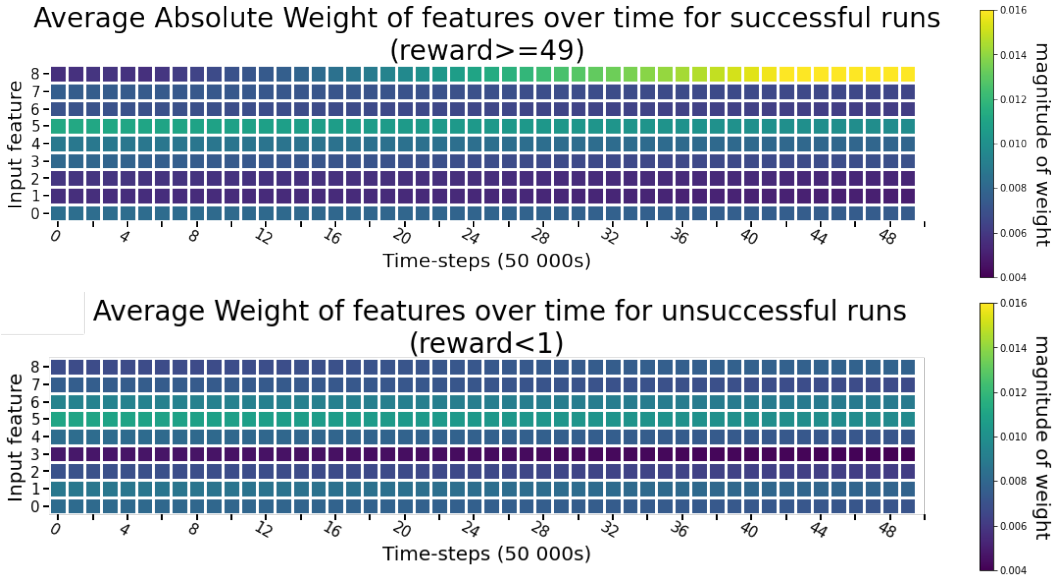


Figure 5.10: A depiction of the weights of the meta-weight vector ω_c for cumulants learned. Inputs $0-6$ are a binary encoding of the agent's location in the linear walk. Input 7 is a binary feature that encodes whether the wind hazard is present. Input 8 is the accumulated heat.

receives an expert defined GVF, as specified in Butcher et al. (2022), and one where an agent that receives only the environment observations—no additional predictive features or information. In this setting, the control learner is a DQN agent (Mnih et al., 2015) adapted from Dopamine (Castro et al., 2018). The control agent has two feed-forward layers with 512 units, and weights were initialised with a uniform Golorot initialiser (Glorot & Bengio, 2010) with a scale of 1. All agents are trained for 249 000 time-steps, and their performance is evaluated on the final 1000 time-steps. During the evaluation phase the ϵ is set to 0.01, limiting non-greedy actions. All reported results are averaged over 30 independent trials.

As in the Monsoon World experiments, the meta-weight vector that specifies the cumulants is a linear combination of features and weights with no activation function. For a single GVF the cumulant at time-step t is: $c_t = \boldsymbol{\omega}_t^\top \boldsymbol{o}_t$. The meta-weights that specify the cumulant $\boldsymbol{\omega}^c$ are initialized uniformly randomly. GVF weights are initialised at with a value of 0. The GVF in the frost-hollow experiment uses a *bit-cascade* representation, as in Butcher et al. (2022). The control agent uses an artificial neural network as its function approximator. In particular, this experiment uses a `JaxDQNAgent` from Castro et al. (2018) with the default parameter configurations for all control agents. Additional reported parameters were chosen by performing a sweep across different values, choosing the best performing combination for each agent during the final 1000 evaluation steps in the experiment.

In Figure 5.8, the average cumulative reward is reported during the final evaluation steps. If an agent is deterministically following the optimal policy during the evaluation phase, the maximum possible cumulative reward is 50. In Butcher et al. (2022), the best performing agent with a highly specialised representation was able to achieve a cumulative reward of around 40; however, many of the agents with hand-selected predictions received a cumulative reward of approximately 30.

In this experiment, the baseline agent without any additional predictions achieves an average cumulative reward of 7. By adding another predictive input feature that is specified by MGD, the agent was able to achieve an average

cumulative reward of 18.7. Of note, the MGD agent learned to specify a cumulant that is different from those chosen in Butcher et al. (2022). Successful runs learn a cumulant that predominantly weights the accumulated heat, input 8 (as depicted in Figure 5.10). In Figure 5.10 the average meta-weight vector specifying the cumulant over the course of the entire experiment is reported. In Butcher et al. (2022), the expert chosen prediction is of the oncoming hazard: input 7. There is a logic to the prediction selected by MGD: the heat an agent accumulates is directly related to reward; reward is received after an agent acquires 12 heat points. Moreover, the heat accumulated is indirectly related to the hazard: if the agent is unprotected before the hazard, it should anticipate a drop in its accumulated heat. By predicting accumulated heat, the agent is dually capturing information about both the sparse reward signal and the original aspect of the environment that the expertly defined prediction sought to predict.

Interestingly, the baseline agent that used an expert-specified prediction from Butcher et al. (2022) performed less well than the MGD agent, receiving an average cumulative reward of 3.3. This is a revealing example: while the expert-specified GVF was well suited to the tabular setting explored in prior work (Butcher et al., 2022), its effectiveness as a predictive feature did not generalise to this function approximation setting. This highlights a challenge that is present across domains and environmental settings. What predictive features may be useful to an agent is influenced not only by the environment, but also by differences in state construction and the underlying learning method. Together, these factors all influence what GVFs may be useful for decision-making.

5.6 Limitations & Future Work

In this chapter, I demonstrate for the first time that end-to-end learning of GVFs used as predictive inputs is possible. Moreover, I demonstrate this within the setting where GVFs were originally proposed: continual life-long learning. This has been an open challenge in GVF research since their introduction over ten years ago. However, this chapter is not the final word on MGD discovery

of GVFs. Assessing how well the method presented scales in environments with non-stationarity, or greater observational complexity, is important future work. In this chapter, I decided to fix the discount γ—the time-horizon over which a prediction is estimated. Previous work has enabled GVFs to be learned over multiple timescales at the same time, enabling inference over arbitrary horizons (Sherstan et al., 2020). Future work could explore how Sherstan *et al.*'s γ-net formulation could improve the scalability and flexibility of the meta-learning process introduced. This chapter makes progress in enabling agents to choose what to predict. How an agent may decide the number of predictions to learn remains an important open question. Similarly, it remains to be explored how an agent can incrementally increase its capacity by adding new predictions during lifelong learning. One possibility is to arrange predictions in multiple layers, similar to GVF networks (Schlegel et al., 2021). Questions concerning GVF structure and scale are exciting open frontiers, and this thesis provides a foundation that enables such questions to be asked in future work.

5.7 Conclusion

In this chapter, I introduced a process that enables agents to meta-learn the specifications of predictions in the form of GVFs. This process enabled agents to learn what aspects of the environment to predict to use as additional input features, while also learning the predictions themselves and learning a control policy. This meta-learning process was developed in an online, incremental fashion, making it possible for long-lived continual learning agents to self-supervise the specification and learning of their own GVFs. An agent with no prior knowledge of the environment was able to select predictions that yielded performance equitable to, or better than agents using expertly chosen predictive features. Even in an environment with a sparse reward, an agent was still able to learn to specify useful predictions to use as additional features based on the control-learner's error.

It has long been suggested that predictions of future experience in the form of GVFs can provide useful features to support decision-making in computational Reinforcement Learning. Requiring system designers to re-specify the GVFs for every permutation of an agent and its environment is a burden for applications of predictive features, and yet it is still the norm in both research and applied settings. Fundamentally, the requirement of designers to hand-specify GVFs prevents the development and the application of Predictive Knowledge agents, especially in long-lived continual learning domains that may exhibit non-stationarity. In this chapter, I contributed a meta-gradient descent processes by which agents were able to find GVFs that improved decision-making relative to environment observations alone.

Chapter 6

Future Horizons

This thesis concerned itself with how an agent may develop knowledge of its world through interaction with its environment. In particular, I explored and refined the subfield of Predictive Knowledge (Koop, 2008; Ring, 1994; Sutton, 2009; White, 2015), a growing collection of work that seeks to express an agent's knowledge through collections of predictions of future sensations. This approach to machine knowledge draws inspiration from a wide body of work in biological intelligence that has shown that humans and animals build sensorimotor models of the world and that these models form the basis of perception (Rao & Ballard, 1999; Wolpert et al., 1995). Similarly, Predictive Knowledge enables an agent to construct knowledge of its world from the agent's subjective perspective, independent of human intervention.

In this thesis, I explore predictions as value functions (Sutton & Barto, 2018). Value functions estimate the discounted sum of some future signal observed under a target agent behaviour. Value functions can be learned incrementally, online, in a self-supervised fashion. Based on a strong foundation, predictive approaches to machine knowledge have faced two key challenges: how do agents decide what to predict, and how do agents learn their predictions? This thesis addressed these two questions.

Let us now reflect on the contributions of this thesis and on how these questions were approached.

6.1 Adapting Step Sizes & Bias

In this thesis, I contributed a step-size adaptation method for temporal difference learning that I call TIDBD. Through TIDBD, an agent can incrementally adapt its step sizes while interacting with its environment based on its performance on a prediction task. To do so, TIDBD introduces a new *meta step size* that must be chosen. TIDBD was empirically compared to ordinary TD learning on a stationary prediction problem. I demonstrated that when used to adapt a single step size shared amongst features, TIDBD outperforms TD for a wide variety of meta step size values. Similar to ordinary TD, this thesis presented evidence that TIDBD is sensitive to its meta step size. Like many machine learning methods that take steps in the direction of a gradient, or error, the best choice of meta step size varies from one problem to the next. To address this sensitivity, I incorporated AutoStep style normalisation into the update of TIDBD, which I name AutoTIDBD. I found that AutoTIDBD can outperform ordinary TD learning for a broader set of meta step size values on a stationary prediction task. On this stationary problem, AutoTIDBD performs better than TD for a broader set of step size and meta step size combinations than TIDBD without AutoStep's normalisation. I then empirically evaluated TIDBD on a non-stationary prediction task where step sizes are assigned on a per-feature basis. Similar to prior results, I found that AutoTIDBD is less sensitive than TIDBD to meta-step size selection and that it outperformed many existing step-size adaptation methods. A promise of adapting step sizes on a per-feature basis is the ability to perform representation learning by scaling features autonomously. On the same non-stationary prediction task, a portion of the input features were assigned to be noisy, and demonstrated that AutoTIDBD tuned the step sizes of noisy features down.

One goal of this thesis was to develop a method of adapting the step sizes and features used to learn General Value Functions. In particular, I was concerned with developing a method of adapting step sizes suited for the real-world: a means of adapting how GVF predictions are learned in a never-ending and non-stationary learning problem with noisy inputs. On robotic prediction

tasks, I evaluated existing step-size adaptation methods originally presented in episodic, stationary learning environments. In doing so, this thesis provides new insights on how existing step-size methods perform in real-world learning settings.

6.2 Evaluation

In this thesis, I critically evaluated error metrics that form the basis of GVF evaluation. The primary motivation of learning predictions in Predictive Knowledge is their use: ultimately, predictions are a way for agents to conceptualise the world, and those conceptualisations should be in service to decision-making. How does an agent determine whether to rely on a prediction? It is a common belief that predictions should be chosen based on their error. I argue that error is not always correlated with whether a prediction is useful for decision-making. Using a simple binary prediction problem, I demonstrated how conventional online evaluation methods do not always rank predictions effectively. Using a network of predictions, I then demonstrated that poor methods of quantifying performance for these initial estimates can lead to catastrophic impacts on the performance of additional learning processes that depend on them. By examining how agent performance is quantified, this thesis reveals a challenge for Predictive Knowledge: determining how an agent disambiguates between the good and the bad when differentiating between GVF estimates. Returning to my first contribution, I generalise AutoTIDBD for off-policy policy evaluation. Using off-policy AutoTIDBD, I demonstrated that designers and agents can gain greater insight into the quality of a GVF estimate by examining the relevance of the features a GVF uses in addition to the error estimate.

6.3 Meta-descent

Finally, this thesis turns its attention to GVF selection. How does an agent identify which predictions will best support decision-making? Inspired by Incremental Delta-Bar-Delta and meta-descent, I parameterise the components of a GVF that determine *what* a GVF is about. I constructed a loss such that

what a GVF estimated was incrementally updated based on feedback from a control learner solving a task. I demonstrated that an agent was able to find predictions that—when used as features—enabled the agent to solve a partially observable control task. On a sparse reward problem, an agent that selected its own GVFs was able to outperform an agent using expert-chosen predictions. Moreover, the meta-gradient agent found predictions that differed from those chosen by domain experts. In my experiments, the control policy, GVF estimates, and the specification of GVFs could all be learned online and incrementally. Although the number of GVFs learned in each case were modest, this work provides a new perspective on enabling agents to self-direct the construction of Predictive Knowledge.

6.4 Future Directions

In this thesis, I contributed a new approach to self-organisation of predictions; however, there is still important work to be done to bring this line of work to its full potential. In this section, select areas of particular interest that remain for future work are highlighted.

Meta-learning both *what* and *how*: This thesis opens up new directions for the development of Predictive Knowledge agents. To analyse each learning method independently, I did not use TIDBD when meta-learning the specification of GVFs. The combined usage step-size adaptation and autonomous GVF selection would be a further step in the development of truly autonomous Predictive Knowledge agents.

Meta-gradient descent vs Generate and Test: While this thesis made oblique references to using feature relevance as a means of augmenting generate and test approaches to GVF selection, this remains possible future work. Instead, I chose to focus on meta-descent approaches to specifying GVFs. Furthermore, there are many possible avenues to enable GVF selection; further exploration of this area is likely to yield improvements.

Hierarchical Collections of Predictions: In this thesis, I briefly touched on collections of predictions that are structured as a network in Chapter 4, following the thought experiment from (Ring, 2021). In this thesis, I do not address how an agent could autonomously develop its own hierarchical predictions; this interrelation of predictions is left for future work.

Constructive agents: The meta-descent method presented in this thesis enabled agents to change the specification of a fixed set of predictions over time. How such an agent may incrementally add or remove predictions over time—or construct a network of predictions—remains to be explored.

Life-long environments: A motivation of Predictive Knowledge is the development of agents that can autonomously and continuously learn about their environment over a very long lifetime—agents that can continuously learn and improve their performance with respect to a goal over time. I explored continuing learning problems in this thesis; however, I did not explore large learning environments where an agent could learn forever. How well a Predictive Knowledge agent can grow and develop its understanding of the world it inhabits during a long lifespan is a question that remains unanswered.

The final word: Prediction is integral to biological perception: humans and animals continually anticipate what they may see next at any given moment. Inspired by these biological systems, it has long been suggested that artificial agents could use temporal-difference learning methods to learn predictive models of their world. In this thesis, I present an approach to two main challenges of Predictive Knowledge: enabling agents to modify *how* they learn, and enabling agents to select *what* they learn about. In doing so, this thesis presents an approach to removing two fundamental impediments to research in Predictive Knowledge. This thesis both expands our understanding of how predictive agents can learn autonomously and how Predictive Knowledge agents can be applied to decision-making problems. It is my hope that this work enables the further development of artificial agents that can continuously develop knowledge of their world by autonomously learning from their subjective stream of experience.

Bibliography

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., & De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. *Advances in Neural Information Processing Systems*, 3981–3989.

Bagheri, S., Thill, M., Koch, P., & Konen, W. (2016). Online adaptable learning rates for the game connect-4. *IEEE Transactions on Computational Intelligence and AI in Games*, *8*(1), 33–42. https://doi.org/10.1109/TCIAIG.2014.2367105.

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

Baranes, A., & Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, *61*(1), 49–73.

Barbieri, M. (2007). *Introduction to Biosemiotics: The New Biological Synthesis*. Springer Science & Business Media.

Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, *23*(2), 357–365.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., & Silver, D. (2017). Successor features for transfer in reinforcement learning. *Advances in Neural Information Processing Systems*, 4055–4065.

Becker, J. D. (1973). A model for the encoding of experiential information. *Computer Models of Thought and Language*, 396–434.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bengio, Y., Goodfellow, I., & Courville, A. (2017). *Deep Learning* (Vol. 1). MIT press Cambridge, MA, USA.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, *24*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, *13*(2).

Bhandari, J., Russo, D., & Singal, R. (2018). A finite time analysis of temporal difference learning with linear function approximation. *Conference On Learning Theory*, *75*, 1691–1692.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D. J., van Hasselt, H., Munos, R., Silver, D., & Schaul, T. (2019). Universal successor features approximators. *International Conference on Learning Representations*.

Bowling, M., & Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215–250.

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

Brenneis, D. J. A., Parker, A. S., Johanson, M. B., Butcher, A., Davoodi, E., Acker, L., Botvinick, M. M., Modayil, J., White, A., & Pilarski, P. M. (2021). Assessing human interaction in virtual reality with continually learning prediction agents based on reinforcement learning algorithms: A pilot study. *arXiv:2112.07774*.

Bridges, M. M., Para, M. P., & Mashner, M. J. (2011). Control system architecture for the modular prosthetic limb. *Johns Hopkins APL Technical Digest*, *30*(3), 217–222.

Brown, T., Mane, D., Roy, A., Abadi, M., & Gilmer, J. (2017). Adversarial patch. *Machine Deception Workshop. arXiv:1712.09665*.

Butcher, A., Johanson, M. B., Davoodi, E., Brenneis, D. J. A., Acker, L., Parker, A. S. R., White, A., Modayil, J., & Pilarski, P. M. (2022). Pavlovian signalling with general value functions in agent-agent temporal decision making. *arXiv:2201.03709*.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., & Bellemare, M. G. (2018). Dopamine: A research framework for deep reinforcement learning. *arXiv:1812.06110*.

Chaput, H. H., Kuipers, B., & Miikkulainen, R. (2003). Constructivist learning: A neural implementation of the schema mechanism. *Workshop on Self-Organizing Maps*.

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

Colunga, E., & Smith, L. B. (2005). From the lexicon to expectations about kinds: A role for associative learning. *Psychological Review*, *112*(2), 347.

Comanici, G., Precup, D., Barreto, A., Toyama, D. K., Aygün, E., Hamel, P., Vezhnevets, S., Hou, S., & Mourad, S. (2018). *Knowledge Representation for Reinforcement Learning using General Value Functions*. DeepMind.

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

Dabney, W. (2014). *Adaptive Step-Sizes for Reinforcement Learning* (Doctoral dissertation). University of Massachusetts, Amherst.

Dabney, W., & Barto, A. G. (2012). Adaptive step-size for online temporal difference learning. *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Dalrymple, A. N., Roszko, D. A., Sutton, R. S., & Mushahwar, V. K. (2020). Pavlovian control of intraspinal microstimulation to produce over-ground walking. *Journal of Neural Engineering*, *17*(3).

Dawson, M. R., Sherstan, C., Carey, J. P., Hebert, J. S., & Pilarski, P. M. (2014). Development of the Bento Arm: An improved robotic arm for myoelectric training and research. *Myoelectric Controls Symposium*, *14*, 60–64.

Dayan, P. (1993). Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, *5*(4), 613–624.

Degris, T., & Modayil, J. (2012). Scaling-up knowledge for a cognizant robot. *AAAI Spring Symposium Series*.

Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Drescher, G. L. (1991). *Made-up Minds: A Constructivist Approach to Artificial Intelligence*. MIT press.

Edgar An, P., Miller, W. T., & Parks, P. (1991). Design improvements in associative memories for cerebellar model articulation controllers (CMAC). *International Conference on Artificial Neural Networks*, 1207–1210.

Edwards, A. L., Dawson, M. R., Hebert, J. S., Sherstan, C., Sutton, R. S., Chan, K. M., & Pilarski, P. M. (2016). Application of real-time machine learning to myoelectric prosthesis control: A case series in adaptive switching. *Prosthetics and Orthotics International*, *40*(5), 573–581.

Edwards, A. L., Dawson, M. R., Hebert, J. S., Sutton, R. S., Chan, K. M., & Pilarski, P. M. (2014). Adaptive switching in practice: Improving myoelectric prosthesis performance through reinforcement learning. *Myoelectric Controls Symposium*, 18–22.

Edwards, A. L., Hebert, J. S., & Pilarski, P. M. (2016). Machine learning and unlearning to autonomously switch between the functions of a myoelectric arm. *Biomedical Robotics and Biomechatronics (BioRob)*, 514–521.

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning*, 1126–1135.

Forestier, S., Portelas, R., Mollard, Y., & Oudeyer, P.-Y. (2022). Intrinsically motivated goal exploration processes with automatic curriculum learning. *Journal of Machine Learning Research*, *23*(1), 6818–6858.

Ghiassian, S., Patterson, A., White, M., Sutton, R. S., & White, A. (2018). Online off-policy prediction. *arXiv:1811.02597*.

Ghiassian, S., & Sutton, R. S. (2021). An empirical comparison of off-policy prediction learning algorithms on the collision task. *arXiv:2106.00922*.

Gilbert, D. (2009). *Stumbling on Happiness*. Vintage Canada.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, 249–256.

Goldman, A. I. (1976). Discrimination and perceptual knowledge. *The Journal of Philosophy*, *73*(20), 771–791.

Graves, D., Günther, J., & Luo, J. (2021). Affordance as general value function: A computational model. *Adaptive Behavior*.

Guerin, F., & Starckey, A. (2009). Applying the schema mechanism in continuous domains. *Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*.

Günther, J., Ady, N. M., Kearney, A., Dawson, M. R., & Pilarski, P. M. (2020). Examining the use of Temporal-Difference Incremental Delta-Bar-Delta for real-world predictive knowledge architectures. *Frontiers in Robotics and AI*, *7*, 34.

Günther, J., Kearney, A., Ady, N. M., Dawson, M. R., & Pilarski, P. M. (2019). Meta-learning for Predictive Knowledge Architectures: A Case Study Using TIDBD on a Sensor-rich Robotic Arm. *International Conference on Autonomous Agents and MultiAgent Systems*, 1967–1969.

Günther, J., Kearney, A., Dawson, M. R., Sherstan, C., & Pilarski, P. M. (2018). Predictions, surprise, and predictions of surprise in general value function architectures. *AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*, 22–29.

Günther, J., Pilarski, P. M., Helfrich, G., Shen, H., & Diepold, K. (2016). Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning. *Mechatronics*, *34*, 1–11.

Ha, D., & Schmidhuber, J. (2018). Recurrent world models facilitate policy evolution. *Advances in Neural Information Processing Systems*, *31*, 2450–2462.

Hackman, L. (2012). *Faster Gradient-TD Algorithms* (Master's thesis). University of Alberta.

Hallak, A., Tamar, A., Munos, R., & Mannor, S. (2016). Generalized emphatic temporal difference learning: Bias-variance analysis. *Thirtieth AAAI Conference on Artificial Intelligence*.

Harb, J., & Precup, D. (2017). Investigating recurrence and eligibility traces in deep Q-networks. *arXiv:1704.05495*.

Huang, S., Papernot, N., Goodfellow, I., Duan, Y., & Abbeel, P. (2017). Adversarial attacks on neural network policies. *arXiv:1702.02284*.

Hutter, M., & Legg, S. (2008). Temporal difference updating without a learning rate. *Advances in Neural Information Processing Systems*, 705–712.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. *5th International Conference on Learning Representations*.

Jaeger, H. (2000). Observable operator models for discrete stochastic time series. *Neural Computation*, *12*(6), 1371–1398.

Johannes, M. S., Bigelow, J. D., Burck, J. M., Harshbarger, S. D., Kozlowski, M. V., & Van Doren, T. (2011). An overview of the developmental process for the modular prosthetic limb. *Johns Hopkins APL Technical Digest*, *30*(3), 207–216.

Johannes, M. S., Faulring, E. L., Katyal, K. D., Para, M. P., Helder, J. B., Makhlin, A., Moyer, T., Wahl, D., Solberg, J., Clark, S., et al. (2020). *The modular prosthetic limb*.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The Malmo platform for artificial intelligence experimentation. *International Joint Conference on Artificial Intelligence*, 4246–4247.

Kansky, K., Silver, T., Mély, D. A., Eldawy, M., Lázaro-Gredilla, M., Lou, X., Dorfman, N., Sidor, S., Phoenix, S., & George, D. (2017). Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *International Conference on Machine Learning*, 1809–1818.

Kearney, A., Koop, A., Sherstan, C., Gunther, J., Sutton, R. S., Pilarski, P. M., & Taylor, M. E. (2018). Evaluating predictive knowledge. *AAAI Fall Symposium on Reasoning and Learning In Real-World Systems For Long-Term Autonomy*.

Kearney, A., Veeriah, V., Travnik, J., Pilarski, P. M., & Sutton, R. S. (2019). Learning Feature Relevance Through Step Size Adaptation in Temporal-Difference Learning. *arXiv:1903.03252*.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Koop, A. (2008). *Investigating Experience: Temporal Coherence and Empirical Knowledge Representation* (Master's thesis). University of Alberta.

Krizhevsky, A., & Hinton, G. (2009). *Learning multiple layers of features from tiny images*. University of Toronto.

Li, X.-J., & Yang, G.-H. (2012). Fault detection for linear stochastic systems with sensor stuck faults. *Optimal Control Applications and Methods*, *33*(1), 61–80.

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*. Carnegie Mellon University.

Linke, C., Ady, N. M., White, M., Degris, T., & White, A. (2020). Adapting behavior via intrinsic reward: A survey and empirical study. *Journal of Artificial Intelligence Research*, *69*, 1287–1332.

Littman, M. L., Sutton, R. S., & Singh, S. (2002). Predictive representations of state. *Advances in Neural Information Processing Systems*, 1555–1561.

Liu, B., Liu, J., Ghavamzadeh, M., Mahadevan, S., & Petrik, M. (2016). Proximal gradient temporal difference learning algorithms. *International Joint Conference on Artificial Intelligence*, 4195–4199.

Lydia, A., & Francis, S. (2019). Adagrad—an optimizer for stochastic gradient descent. *International Journal of Information And Computing Science*, *6*(5), 566–568.

Ma, C., Wen, J., & Bengio, Y. (2018). Universal successor representations for transfer reinforcement learning. *arXiv:1804.03758*.

Maei, H. R. (2011). *Gradient Temporal-difference Learning Algorithms* (Doctoral dissertation). University of Alberta.

Mahmood, A. R. (2017). *Incremental Off-policy Reinforcement Learning Algorithms* (Doctoral dissertation). University of Alberta.

Mahmood, A. R., & Sutton, R. S. (2013). Representation search through generate and test. *AAAI Workshop: Learning Rich Representations from Low-Level Sensors*.

Mahmood, A. R., Sutton, R. S., Degris, T., & Pilarski, P. M. (2012). Tuning-free step-size adaptation. *IEEE International Conference On Acoustics, Speech and Signal Processing*, 2121–2124.

Makino, T., & Takagi, T. (2008). On-line discovery of temporal-difference networks. *International Conference on Machine Learning*, 632–639.

McCarthy, J., & Hayes, P. J. (1981). Some philosophical problems from the standpoint of artificial intelligence. *Readings in Artificial Intelligence*, 431–450.

Milan, K., Veness, J., Kirkpatrick, J., Bowling, M., Koop, A., & Hassabis, D. (2016). The forget-me-not process. *Advances in Neural Information Processing Systems*, *29*.

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

Modayil, J., & Kuipers, B. (2008). The Initial Development of Object Knowledge by a Learning Robot. *Robotics and Autonomous Systems*, *56*(11), 879–890.

Modayil, J., & Sutton, R. S. (2014). Prediction driven behavior: Learning predictions that drive fixed responses. *The AAAI-14 Workshop on Artificial Intelligence and Robotics*.

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

Mugan, J., & Kuipers, B. (2008). *Towards the application of reinforcement learning to undirected developmental learning*. Carnegie Mellon University.

Ni, K., Ramanathan, N., Chehade, M. N. H., Balzano, L., Nair, S., Zahedi, S., Kohler, E., Pottie, G., Hansen, M., & Srivastava, M. (2009). Sensor network data fault types. *ACM Transactions on Sensor Networks*, *5*(3), 25.

Nöe, A. (2004). *Action in Perception*. MIT press.

Oudeyer, P.-Y., & Kaplan, F. (2009). What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, *1*, 6.

Oudeyer, P.-Y., Kaplan, F., Hafner, V., & Whyte, A. (2005). The playground experiment: Task-independent development of a curious robot. *AAAI Spring Symposium on Developmental Robotics*.

Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: An ASR corpus based on public domain audio books. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 5206–5210.

Pezzulo, G. (2011). Grounding procedural and declarative knowledge in sensorimotor anticipation. *Mind & Language*, *26*(1), 78–114.

Pezzulo, G., Donnarumma, F., & Dindo, H. (2013). Human sensorimotor communication: A theory of signaling in online social interactions. *PloS one*, *8*(11).

Piaget, J. (1954). *The Construction of Reality in the Child*. New York: Basic Books.

Piaget, J., & Cook, M. (1952). *The Origins of Intelligence in Children* (Vol. 8). International Universities Press New York.

Piaget, J., & Duckworth, E. (1970). Genetic epistemology. *American Behavioral Scientist*, *13*(3), 459–480.

Pierce, D., & Kuipers, B. (1997). Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, *92*(1), 169–227.

Pilarski, P. M., Butcher, A., Davoodi, E., Johanson, M. B., Brenneis, D. J., Parker, A. S., Acker, L., Botvinick, M. M., Modayil, J., & White, A. (2022). The frost hollow experiments: Pavlovian signalling as a path to coordination and communication between agents. *arXiv:2203.09498*.

Pilarski, P. M., Dawson, M. R., Degris, T., Carey, J. P., Chan, K. M., Hebert, J. S., & Sutton, R. S. (2013). Adaptive artificial limbs: A real-time approach to prediction and anticipation. *IEEE Robotics & Automation Magazine*, *20*(1), 53–64.

Pilarski, P. M., Dawson, M. R., Degris, T., Carey, J. P., & Sutton, R. S. (2012). Dynamic switching and real-time machine learning for improved human control of assistive biomedical robots. *4th IEEE International Conference on Biomedical Robotics and Biomechatronics*, 296–302.

Pilarski, P. M., Dawson, M. R., Degris, T., Fahimi, F., Carey, J. P., & Sutton, R. S. (2011). Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. *IEEE International Conference on Rehabilitation Robotics*, 1–7.

Pilarski, P. M., Dick, T. B., & Sutton, R. S. (2013). Real-time prediction learning for the simultaneous actuation of multiple prosthetic joints. *IEEE 13th International Conference on Rehabilitation Robotics (ICORR)*, 1–8.

Pilarski, P. M., & Sherstan, C. (2016). Steps toward knowledgeable neuroprostheses. *6th IEEE International Conference on Biomedical Robotics and Biomechatronics*, 220–220. https://doi.org/10.1109/BIOROB.2016.7523626.

Precup, D. (2000). *Temporal Abstraction in Reinforcement Learning* (Doctoral dissertation). University of Massachusetts, Amherst.

Qin, Y., Carlini, N., Cottrell, G., Goodfellow, I., & Raffel, C. (2019). Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. *International Conference on Machine Learning*, 5231–5240.

Radulescu, A., Niv, Y., & Ballard, I. (2019). Holistic reinforcement learning: The role of structure and attention. *Trends in Cognitive Sciences*, *23*(4), 278–292.

Rafiee, B. (2018). *Predictive Knowledge in Robots: An Empirical Comparison of Learning Algorithms* (Master's thesis). University of Alberta.

Rao, R. P., & Ballard, D. H. (1999). Predictive coding in the visual cortex: A functional interpretation of some extra-classical receptive-field effects. *Nature Neuroscience*, *2*(1), 79–87.

Ring, M. (1994). *Continual Learning in Reinforcement Environments* (PhD Thesis). University of Texas at Austin.

Ring, M. (1997). CHILD: A first step towards continual learning. *Machine Learning*, *28*(1), 77–104.

Ring, M. (2021). Representing Knowledge as Predictions (and State as Knowledge). *arXiv:2112.06336*.

Russell, S., & Norvig, P. (2010). *Artificial Intelligence a Modern Approach*. Pearson Education, Inc.

Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal value function approximators. *International Conference on Machine Learning*, *37*, 1312–1320.

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In Y. Bengio & Y. LeCun (Eds.), *4th International Conference on Learning Representations, ICLR*.

Schlegel, M., Jacobsen, A., Abbas, Z., Patterson, A., White, A., & White, M. (2021). General value function networks. *Journal of Artificial Intelligence Research*, *70*, 497–543.

Schlegel, M., White, A., & White, M. (2018). A baseline of discovery for general value function networks under partial observability. *NeurIPS Workshop on Reinforcement Learning under Partial Observability*.

Schlegel, M., & White, M. (2022). Predictions predicting predictions. *The 5th Multi-disciplinary Conference on Reinforcement Learning and Decision Making*.

Schraudolph, N. N. (1999). Local gain adaptation in stochastic gradient descent. *9th International Conference on Artificial Neural Networks*, 569–574.

Schultz, W., Dayan, P., & Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, *275*(5306), 1593–1599.

Schweighofer, N., & Arbib, M. A. (1998). A model of cerebellar metaplasticity. *Learning & Memory*, *4*(5), 421–428.

Seijen, H. (2016). Effective multi-step temporal-difference learning for non-linear function approximation. *arXiv:1608.05151*.

Seijen, H., & Sutton, R. (2014). True online TD(λ). *International Conference on Machine Learning*, *32*(1), 692–700.

Sellars, W. (1956). Empiricism and the philosophy of mind. *Minnesota Studies in the Philosophy of Science*, *1*(19), 253–329.

Shapiro, L. G., & Stockman, G. C. (2001). *Computer vision* (Vol. 3). Prentice Hall New Jersey.

Sherstan, C., Dohare, S., MacGlashan, J., Günther, J., & Pilarski, P. M. (2020). Gamma-nets: Generalizing value estimation over timescale. *AAAI Conference on Artificial Intelligence*, *34*(04), 5717–5725.

Sherstan, C., Machado, M. C., & Pilarski, P. M. (2018). Accelerating learning in constructive predictive frameworks with the successor representation. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2997–3003.

Sherstan, C., Modayil, J., & Pilarski, P. M. (2015). A collaborative approach to the simultaneous multi-joint control of a prosthetic arm. *IEEE International Conference on Rehabilitation Robotics*, 13–18. https://doi.org/10.1109/ICORR.2015.7281168.

Sherstan, C., White, A., Machado, M. C., & Pilarski, P. M. (2016). Introspective agents: Confidence measures for general value functions. *Artificial General Intelligence*, 258–261.

Silver, D. (2013). Gradient temporal difference networks. *Tenth European Workshop on Reinforcement Learning*, *24*, 117–130.

Sinclair, A. H., & Barense, M. D. (2018). Surprise and destabilize: Prediction error influences episodic memory reconsolidation. *Learning & Memory*, *25*(8), 369–381.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, *3*(1), 9–44. https://doi.org/10.1007/BF00115009.

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Tenth National Conference on Artificial Intelligence*, 171–176.

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

Sutton, R. S. (2011). Beyond reward: The problem of knowledge and data. *International Conference on Inductive Logic Programming*, 2–6.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press Cambridge.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., & Precup, D. (2011). Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. *International Foundation for Autonomous Agents and Multiagent Systems AAMAS*, 761–768.

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1), 181–211.

Sutton, R. S., & Tanner, B. (2004). Temporal-Difference Networks. *Advances in Neural Information Processing Systems*.

Tanaka, S. C., Doya, K., Okada, G., Ueda, K., Okamoto, Y., & Yamawaki, S. (2016). Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. *Behavioral Economics of Preferences, Choices, and Happiness*, 593–616.

Thill, M. (2015). *Temporal Difference Learning Methods With Automatic Step-size Adaption for Strategic Board Games: Connect-4 and Dots-and-Boxes* (Master's thesis). Cologne University of Applied Sciences.

Tieleman, T., & Hinton, G. (2012). *Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning*. University of Toronto.

Travnik, J. B., & Pilarski, P. M. (2017). Representing high-dimensional data to intelligent prostheses and other wearable assistive robots: A first comparison of tile coding and selective Kanerva coding. *IEEE International Conference on Rehabilitation Robotics*, *2017*, 1443–1450. https://doi.org/10.1109/ICORR.2017.8009451.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., & Singh, S. (2019). Discovery of useful questions as auxiliary tasks. *Advances in Neural Information Processing Systems*, *32*.

Veeriah, V., Zhang, S., & Sutton, R. S. (2017). Crossprop: Learning representations by stochastic meta-gradient descent in neural networks. *Machine Learning and Knowledge Discovery in Databases*, 445–459.

White, A. (2015). *Developing a Predictive Approach to Knowledge* (PhD Thesis). University of Alberta.

Wilson, R. C., & Niv, Y. (2012). Inferring relevance in a changing world. *Frontiers in Human Neuroscience*, *5*, 189.

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. *International Conference on Machine Learning*, 980–987.

Wolpert, D. M., Ghahramani, Z., & Jordan, M. I. (1995). An internal model for sensorimotor integration. *Science*, *269*(5232), 1880–1882.

Xu, K., Ba, J. L., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R. S., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *Journal of Machine Learning Research*, 2048–2057.

Xu, Z., van Hasselt, H. P., & Silver, D. (2018). Meta-gradient reinforcement learning. *Advances in Neural Information Processing Systems*, *31*.

Young, K., Wang, B., & Taylor, M. E. (2019). Metatrace actor-critic: Online step-size tuning by meta-gradient descent for reinforcement learning control. *International Joint Conference on Artificial Intelligence*, 4185–4191.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method. *arXiv:1212.5701*.