Continual Auxiliary Task Learning

by

Matthew McLeod

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Learning auxiliary tasks, such as multiple predictions about the world, can provide many benefits to reinforcement learning systems. A variety of off-policy learning algorithms have been developed to learn such predictions, but as yet there is little work on how to adapt the behavior to gather useful data for those off-policy predictions. In this thesis, we investigate a reinforcement learning system designed to learn a collection of auxiliary tasks, with a behavior policy that learns to take actions to improve the auxiliary predictions. We highlight the inherent non-stationarity in this continual auxiliary task learning problem, for both prediction learners and the behaviour learner. We develop an algorithm based on successor features that facilitates tracking under non-stationary rewards and propose how behaviour can be specialized to learn areas of interest for a prediction learner. We conduct an in-depth study into the resulting multi-prediction learning system.

Preface

My thesis is based on my joint paper (McLeod et al., 2021) which is currently under review for NeurIPS 2022. The experiments shown in this thesis are covered in McLeod et al. (2021) with minor modifications made by me. The chapters consist of revisions and extensions made by me on the paper under review. Matthew Schlegel and Chunlok Lo made significant contributions to the code base used in this research and Matthew Schlegel conducted the replay experiment shown in Section 8.2. The LaTeX template used in the writing of this thesis was provided by the University of Alberta.

Acknowledgements

I extend my deepest thanks to my supervisor, Adam White, who taught me how to be a better researcher and a more critical thinker. He showed me how to make meaningful progress on problems that otherwise would feel too daunting and inspired me to persevere through the highs and lows of research. I would also like to thank Martha White whose insight and guidance through out this research endeavour was invaluable.

During my Master's, I worked along side with many talented graduate students. Matthew Schlegel was a great mentor who helped me with challenging problems. Working with him felt like I had a personal superhero to call on when times got tough. Chunlok Lo was an absolute joy to work with and he made my Master's much more enjoyable. I'd also like to thank Raksha Kumaraswamy, Andrew Jacobsen and Banafsheh Rafiee who helped me along my research journey.

Finally, I would like to thank my wife, family and friends for their support. Being surrounded with love and encouragement is a blessing I am lucky to have. My wife, Mikyla Foreht, was my biggest cheerleader through my studies and I appreciate so much everything she did. Thank you to my parents, Ian and Maree, and my brother, Jonathan, who instilled in me a love of science and inspired me to pursue a Master's degree.

Contents

List of Figures

Chapter 1

Introduction to Multiple Prediction Learning

Determining which algorithms best support an agent in learning auxiliary tasks and how an agent can adapt its behaviour to enhance its learning is an under researched problem in reinforcement learning. Building our understanding of this area is important to reinforcement learning for two reasons. In many cases agents are improved by learning auxiliary tasks. Moreover, an agent's ability to learn many auxiliary tasks in parallel holds promise in representing an agent's general purpose knowledge since auxiliary tasks are an expressive way to capture their knowledge. The following introductory paragraphs explain these two reasons in more detail and summarize my contributions.

There are a plethora of uses for agents learning information beyond what is described by the rewards the agent is receiving. In deep reinforcement learning, there are many self-supervised signals that can be leveraged by an agent. Jaderberg et al. (2017) found that allowing an agent to predict the sign of the reward that it would experience improved its efficiency in learning and added robustness to its hyperparameters across a range of Atari environments. Predicting terminal states is another domain independent signal that an agent can learn. Kartal et al. (2019) found that having an agent make terminal predictions dramatically improved its performance in some video game environments. Mirowski et al. (2017) formulated additional tasks involving depth prediction for an agent which resulted in improved navigation policies for an agent's main task. Passively learning auxiliary tasks can improve an agent's

performance even if the final predictions are not used.

Auxiliary tasks forming a predictive model of the environment can guide an agent's exploration of large environments because the predictions can be used to identify novel or interesting states (Burda, Edwards, Pathak, et al., 2019; Burda, Edwards, Storkey, et al., 2019; Pathak et al., 2017; Stadie et al., 2015). Predictions by an agent can also be used as part of state representation itself. This can improve the generalization (Schaul & Ring, 2013) as well as performance of an agent on partially observable problems (Schlegel et al., 2021). An agent uses successor representation to efficiently transfer knowledge between tasks and dynamically synthesize complex new behaviours (Barreto et al., 2019; Barreto et al., 2018; Barreto et al., 2017; Barreto et al., 2020). Successor representation enables an agent to accomplish this in part by learning many different prediction tasks that model future feature visitations. Riedmiller et al. (2018) shows an agent can leverage a given set of auxiliary tasks and policies to efficiently explore the environment and learn complex behaviours.

Auxiliary tasks can generally be formulated as general value functions (GVFs). GVFs enable the agent to formulate and answer specific questions about their environment. Capturing a large body of knowledge in an agent's GVFs will likely require the agent to learn GVFs efficiently. For example, Modayil et al. (2014) studied a robot that follows a fixed data collection policy and learns about its future experiences. The robot learns through short term predictions about its future sensorimotor experience ranging from 0.1 seconds to 8 seconds into its future. Thinking more broadly, Mark (2016) outlines an architecture for a generally intelligent agent in a thought experiment. In his thought experiment, the agent becomes generally intelligent by building layers of abstraction beginning with basic GVF questions and progressing towards ever more complex questions about its environment. The efficiency of learning GVFs in parallel will undoubtedly play an important role in future developments.

Recognizing the growing need for agents to be able to effectively adapt their behaviour to learn auxiliary tasks, we address this problem in the multi-

prediction setting. We build off of the work of Linke et al. (2020) who studied the multi-prediction problem in the bandits setting. We use their framing of the problem where each auxiliary task learner generates an intrinsic reward approximating their learning progress. By having a behaviour that maximizes the intrinsic reward, we can have a learner that adapts its behaviour to complement parallel auxiliary task learning. We leverage the key finding in Linke et al. (2020) that simple intrinsic rewards like absolute weight change can be effective if the auxiliary task learners are introspective. We build on this work by moving beyond the bandits setting and addressing the challenges that are raised in Markov Decision Processes (MDPs).

The combination of off-policy learning, function approximation, non-stationary rewards, multiple learners, intrinsic rewards and the dilemma of exploration versus exploitation are challenging problems that have interacting effects. This work highlights the importance of building and analyzing complete reinforcement learning systems. The contributions of this thesis are:

1. The identification of how successor features can address the non-stationary problems faced by a multi-prediction system.

2. An interest reweighting scheme to allow specificity in which regions of the environment are valuable to learn for each auxiliary task.

3. An empirical evaluation across a range of environments to test the ideas presented.

4. The first demonstration of a multi-prediction learning system where the agent's objective is to adapt its behaviour to improve parallel prediction learning in the MDP setting under function approximation.

This thesis is organized into nine chapters. Chapter 1, our introduction, provides motivation and outlines the contributions of this thesis. Chapter 2 reviews the necessary concepts to understand the work discussed in this thesis. Chapter 3 formalizes the multi-prediction problem and offers a high level approach to the solution method. Chapter 4 discusses the related works.

Chapter 5 discusses the inherent non-stationarities in learning faced by an agent in multi-prediction learning. It also highlights strategies to mitigate the negative repercussions of these non-stationarities. Chapter 6 gives a detailed description of the different environments. Chapter 7 demonstrates the experimental results central to this thesis. Chapter 8 discusses extensions of the ideas outlined to make our approach more practical in a larger scale system. Chapter 9 provides a summary of contributions and possible future research directions to take this work.

Chapter 2

Background on Reinforcement Learning

2.1 Basics of Reinforcement Learning

Reinforcement learning is a framework where the decision maker learns in an environment through trial and error. The decision maker typically attempts to maximize a reward signal found in the environment. Its learning process is formalized by considering the environment as a Markov Decision Process (MDP). The decision maker, commonly referred to as an agent, learns to interact with the environment over a set of discrete timesteps, $t \in \mathbb{N}$. An MDP is described by ($\mathcal{S}$,$\mathcal{A}$,$\mathcal{R}$,p): $\mathcal{S}$ is the set of possible states the agent may find itself in, $\mathcal{A}$ is the set of possible actions the agent may take, $\mathcal{R}$ is the set of rewards the agent may receive, and p governs the dynamics of how one state leads to another.

The agent learns as it interacts with its environment by taking action $A_t \in \mathcal{A}$ in state $S_t \in \mathcal{S}$ at timestep t and receiving a reward $R_{t+1} \in \mathcal{R}$. The probability of taking action A_t in state S_t is governed by the policy $\pi(s, a)$, where $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ and $\sum_{a \in \mathcal{A}} \pi(s, a) = 1, \forall s \in \mathcal{S}$. After the agent takes action A_t, the agent receives a reward R_{t+1} and the environment transitions to S_{t+1} according to $p : \mathcal{S} \times \mathcal{A} \times \mathcal{R} \times \mathcal{S} \to [0, 1]$ where $p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a\}$. The Markov property asserts that the conditional probability distribution of the next state and reward only depends on the current state and action. The dynamics can be fully summarized by $p(s', r|s, a)$

because no additional information can be gained from previous states and actions. The reward is a signal provided by the environment and relates to the goal that the agent is trying to accomplish. Often, the agent attempts to maximize the reward signal according to some transition-based discount function $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ (Martha, 2017). The sum of rewards received by the agent and weighted by the transition-based discount function is called the return and is defined as,

$$G_t \stackrel{\text{def}}{=} \sum_{k=0}^{\infty} \left(\prod_{i=0}^{k-1} \gamma(S_{t+i}, A_{t+i}, S_{t+i+1}) \right) R_{t+k+1}.$$

For the ease of readability, we will shorten $\gamma(S_t, A_t, S_{t+1})$ to γ_{t+1}. The transition-based discounting function reflects the trade-off in value of receiving a reward immediately or at some time in the future. The expected discounted return is the value function,

$$v_\pi(s) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s].$$

The value function is conditioned on the agent taking actions according to policy π from state S_t. The value function gives an estimate of how beneficial it is for the agent to be in a state. States that have a higher expected return are more beneficial to the agent. The value function does not tell the agent which action to take to reach high value states. The action-value function enables the agent to learn the expected return based on taking an action A_t and subsequently following policy π. This helps the agent determine which action to take by allowing the agent to select the action with the highest state-action value. The state-action value function is defined as,

$$q_\pi(s, a) \stackrel{\text{def}}{=} \mathbb{E}_\pi[G_t | S_t = s, A_t = a].$$

2.2 Linear Function Approximation

In Section 2.1, we did not consider the particulars of how the value function is estimated. This is an important consideration that has theoretical and prac-

tical implications. The simplest case is to uniquely represent every state and learn the value function for each state. This is called a tabular representation and can be visualized as though the value estimate for each state can be directly looked up in a table. Tabular representation is not suitable for large or continuous MDPs as generalization between states is required in order to have estimates across the state space. Function approximation for the value estimate is a way to induce generalization across states. A state is represented by a feature vector $\mathbf{x}(S_t) \in \mathbb{R}^d$, where d is the number of features. In the case of learning action values, the feature vector may be a state-action feature vector $\mathbf{x}(S_t, A_t) \in \mathbb{R}^d$. Linear function approximation enables generalization by calculating the value estimate, $v_\pi(s)$, as a linear combination of the feature vector for a state. The weights, $\mathbf{w} \in \mathbb{R}^d$, are parameters learned by the agent to perform the linear transform of the state feature vector to the value estimate. The value function is estimated as $\hat{v}(s, \mathbf{w}) \stackrel{\text{def}}{=} \langle \mathbf{x}(s), \mathbf{w} \rangle$. Correspondingly, the action value function is estimated as $\hat{q}(s, a, \mathbf{w}) \stackrel{\text{def}}{=} \langle \mathbf{x}(s, a), \mathbf{w} \rangle$. For simplicity in the algorithm updates, we will not show the weight vector as an inputted argument to the value function estimate or action-value function estimate as it is understood that the weights, $\mathbf{w}$, are used in value estimation in the linear setting.

2.3 Tile Coding

Tile coding is a method for feature construction from a multi-dimensional continuous observation. The observation space is partitioned into individual receptive fields called tiles. If the observation is within the receptive field, the corresponding feature is considered active and has a value of 1. Each of these partitions of tiles is called a tiling. When only one tiling is used for feature construction, this is called state aggregation as many observations are clustered into a single feature that is represented with one active unit. The strength of tile coding is when multiple tilings are offset by one another. This allows generalization in the feature representation and specificity in the representable functions. Figure 2.1 shows a sample tile coding visualization

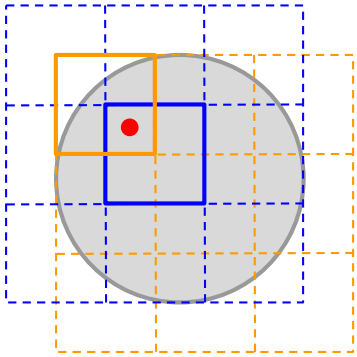


Figure 2.1: Sample tile coding of a 2D circle with two tilings. The red dot is the agent location and the bolded square boxes around the agent demonstrates the active tiles in each tiling.

where the environment is the shaded grey circle and two tilings are used to build the feature representation.

2.4 Temporal-Difference Learning

To get an estimate of G_t, we do not need to wait for the trajectory to end. Instead, we can make use of temporal difference (TD) learning where the estimate of G_t is completed by using value estimates in other states. The use of updating value estimates by their successor states is called bootstrapping. Bootstrapping can enable learning at each time step since the return can be estimated by the reward of the transition and value estimate of the successor states. The estimated return is called the TD target. The error between TD target and current value estimate is called the TD error and is,

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} v(S_{t+1}) - v(S_t)$$

and the case of action values,

$$\delta_t \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1} q(S_{t+1}, A_{t+1}) - q(S_t, A_t).$$

In this thesis, we will show the algorithm updates under the linear function approximation setting. The reduction of algorithm updates to the linear function approximation setting is done since all of the environments studied were

either in the tabular setting or linear function approximation setting. The algorithm updates for the tabular setting can be trivially retrieved from the linear function approximation updates by considering the state feature vector $\mathbf{x}(S_t)$ as a one hot encoding of the state space, where the active feature is the current state.

TD(0) is a learning algorithm that uses the TD error to learn per time step.

Algorithm 1 Linear TD(0) Update

$\delta_t = R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}) - \hat{v}(S_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{x}(S_t)$

While this algorithm converges to the TD fixed point, it is biased by the use of bootstrapping (Sutton & Barto, 2018). Instead of backing up our TD target from the immediate next state, we can backup from a return estimate that uses a mix of the Monte Carlo return and the bootstrapped return. The trace parameter $\lambda : \mathcal{S} \times \mathcal{A} \to [0, 1]$ assigns the amount of bootstrapping in the return estimate per state and action. For the simplicity of notation, let $\lambda_t \stackrel{\text{def}}{=} \lambda(S_t, A_t)$. The λ-return, G_t^λ, is the weighted return with the recursive relationship,

$$G_t^\lambda \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\left((1 - \lambda_{t+1})\,\hat{v}(S_{t+1}) + \lambda_{t+1}G_{t+1}^\lambda\right).$$

In the case of an action value return estimate in Expected Sarsa (ESARSA) form,

$$G_t^{\lambda,a} \stackrel{\text{def}}{=} R_{t+1} + \gamma_{t+1}\left((1 - \lambda_{t+1})\sum_{a' \in \mathcal{A}} \pi(a'|S_{t+1})\hat{q}(a'|S_{t+1}) + \lambda_{t+1}G_{t+1}^{\lambda,a}\right).$$

.

To enable learning at every time step, we can use an eligibility trace vector, $\mathbf{z}_t$. The eligibility trace maintains a decaying history of past gradients to enable the agent to approximate the λ-return such that the final weights updated through eligibility trace approximate the final weights by updating with the λ-return.

The TD(λ) algorithm uses the eligibility trace parameter and is shown below.

Algorithm 2 Linear TD(λ) Update

$\mathbf{z}_t = \gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}(S_t)$
$\delta_t = R_{t+1} + \gamma_{t+1} \hat{v}(S_{t+1}) - \hat{v}(S_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t$

ESARSA(λ) is an online learning algorithm that uses the λ-return and eligibility traces to learn the state-action values.

Algorithm 3 Linear ESARSA(λ) Update

$\mathbf{z}_t = \gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}(S_t, A_t)$
$\delta_t = R_{t+1} + \gamma_{t+1} \sum_{a' \in \mathcal{A}} \pi(a'|S_{t+1}) \hat{q}(a'|S_{t+1}) - \hat{q}(S_t, A_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t$

The algorithms discussed so far learn the value function for the reward provided by the environment. The reward is derived from a source external to the agent, and can appropriately be called the extrinsic reward, r_{ext}. An intrinsic reward, r_{int}, is a scalar reward that is generated by a process of the agent itself. The usage of the two reward signals are not mutually exclusive and can be incorporated into any value function learning method by adding the two rewards $R_t = R_{\text{ext},t} + \beta R_{\text{int},t}$, where β is an optional scaling parameter.

2.5 Off-Policy Learning Algorithms

The agent may want to learn value functions conditioned on a policy that the agent is not acting on. Off-policy learning addresses this restriction by letting the agent learn about a policy different than the one being enacted. The policy the agent acts on is called the behaviour policy, denoted by $b(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$, and the policy that is different than the behaviour policy is called a target policy, denoted by $\pi(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$. The two policies can be unrelated as long as the behaviour policy has coverage over the target policy; i.e: $b(s, a) > 0 \ \forall s \in \mathcal{S}$ if $\pi(s, a) > 0$.

While we can easily substitute the target policy for the behaviour policy in the updates, we still need to correct for the action distribution according

to the expectation of the value function. This correction is called a posterior correction and it is commonly done through importance sampling. The importance sampling ratio, ρ_t for the transition $(S_t, A_t, R_{t+1}, S_{t+1})$ is

$$\rho(A_t|S_t) \stackrel{\text{def}}{=} \frac{\pi(A_t|S_t)}{b(A_t|S_t)}.$$

The TD(λ) learning algorithm can be converted into an off-policy learning algorithm by scaling the eligibility trace by the importance sampling ratio (Sutton & Barto, 2018).

Algorithm 4 Off-Policy Linear TD(λ) Update

$\mathbf{z}_t = \rho_t(\gamma_t \lambda_t \mathbf{z}_{t-1} + \mathbf{x}(S_t))$
$\delta_t = R_{t+1} + \gamma_{t+1}\hat{v}(S_{t+1}) - \hat{v}(S_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t$

In this thesis, we use three off-policy learning algorithms; Tree-Backup(λ) (TB(λ)), a variation on Emphatic TD(λ) that uses TB(λ) updates instead of the traditional TD(λ) updates, and a variation on Least Squares TD(λ) called LSTB(λ) which uses a Tree Backup(λ) update. We will present each algorithm in the linear function approximation setting.

Tree-Backup(λ) is an off-policy learning algorithm that does not use importance sampling directly (Precup, 2000). This can help ensure the variance updates are not too large. Algorithm 5 shows the learning update.

Algorithm 5 Linear TB(λ) Update

$\mathbf{z}_t = \gamma_t \pi(A_t|S_t)\lambda \mathbf{z}_{t-1} + \mathbf{x}(S_t, A_t)$
$\delta_t = R_{t+1} + \gamma_{t+1} \sum_{a'} \pi(a'|S_{t+1})\hat{q}(S_{t+1}, a') - \hat{q}(S_t, A_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t$

The ETB(λ) algorithm is from Sutton et al. (2016) but modified to use TB(λ) instead of TD(λ). The strategy to derive this is to rely on the correspondence between TB and TD, where TB is simply a version of TD with variable trace parameter, $\lambda_t = b(a_t|s_t)\lambda$, as highlighted in Mahmood et al. (2017). Here, we use the same replacement of λ_t in ETD(λ) to get ETB(λ). F_0 is initialized to 0 at the start of an episode.

Algorithm 6 Emphatic Linear TB(λ) Update

$F_t = \rho_{t-1}\gamma_t F_{t-1} + I_t(S_t, A_t)$
$M_t = \rho_t\Big[\lambda b(A_t|S_t)I_t(S_t, A_t) + \big(1 - \lambda b(A_t|S_t)\big)F_t)\Big]$
$\mathbf{z}_t = \gamma_t\pi(A_t|S_t)\lambda\mathbf{z}_{t-1} + M_t\mathbf{x}(S_t, A_t)$
$\delta_t = R_{t+1} + \gamma_{t+1}\sum_{a'}\pi(a'|S_{t+1})\hat{q}(S_{t+1}, a') - \hat{q}(S_t, A_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t$

The Least Squares TD(λ) (LSTD(λ)) algorithm works a bit differently than the TB(λ) or ETB(λ) algorithm. Instead of iteratively applying a gradient estimate to converge to the TD fixed point, LSTD(λ) directly computes the TD fixed point from the set of data experienced over the lifetime of the agent (Sutton & Barto, 2018). We use the linear LSTD(λ) algorithm (Bradtke & Barto, 1996) for off-policy learning that uses the Tree-Backup estimate rather than TD.

Algorithm 7 LSTD(λ) with TB trace updates

$\rho \leftarrow \pi(S_t, A_t)$
$\mathbf{z} \leftarrow \gamma_t\lambda_t\rho\mathbf{z} + \mathbf{x}(S_t, A_t)$
$\mathbf{b} \leftarrow \mathbf{b} + \frac{1}{t+1}(R_{t+1}\mathbf{z} - \mathbf{b})$
$\mathbf{u} \leftarrow \sum_{a'\in\mathcal{A}}\pi(S_{t+1}, a')\mathbf{x}(S_t, a')$
$\mathbf{v} \leftarrow \Big((\mathbf{x}(S_t, A_t) - \gamma_{t+1}\mathbf{u})^T A^{-1}\Big)^T$
$\mathbf{A}^{-1} \leftarrow \frac{t}{t-1}\left(\mathbf{A}^{-1} + \frac{\mathbf{A}^{-1}\mathbf{z}\mathbf{v}^T}{t-1+\mathbf{v}^T\mathbf{z}}\right)$
$\mathbf{w}_{t+1} \leftarrow \mathbf{A}^{-1}\mathbf{b}$

2.6 General Value Functions

GVFs are an abstraction of the traditional value function found in reinforcement learning. GVFs separate the learning of a value function from the environment's reward and discount function (Sutton et al., 2011). The abstraction from value function to GVFs requires three components: a termination function, a target policy and a cumulant. These components do not need to be related to the task of the main problem so that an expressive range of predictions can be formulated. This section shows how the components of a value function are abstracted to create GVFs.

A value function tries to estimate the expected sum of discounted rewards from a given state. This definition prevents the agent from learning any auxiliary variable found in its sensorimotor experience because it is tied to the environment reward. A GVF extends the idea of reward in a value function to being any signal found in the agent's sensorimotor experience. In GVFs, the value function can be an expected discounted sum of any function of the sensorimotor experience based on the state transition (S_t, A_t, S_{t+1}). This offers versatility in what the agent can learn. The signal that is being learned is called a cumulant. A cumulant, $c : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, maps state transitions to a scalar value, where $C_{t+1} \stackrel{\text{def}}{=} c(S_t, A_t, S_{t+1})$.

The modified value function is still unnecessarily restricted to the base problem's termination function. In order to express a larger set of learnable functions, we can relax this restriction by defining a termination function, $\gamma : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ specific to the GVF. The termination function of the GVF is not constrained by the base problem's discount function. It retains the same subscript notation with $\gamma_{t+1} \stackrel{\text{def}}{=} \gamma(S_t, A_t, S_{t+1})$. To completely shift to GVFs, the policy should not be constrained by the behaviour of the agent. With π being the target policy of the GVF, the value function the GVF is learning is,

$$v_{\pi,c,\gamma}(S_t) \stackrel{\text{def}}{=} \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \left(\prod_{i=1}^{k} \gamma_{t+i} \right) C_{t+k+1} | S_t = s, A_{t:\infty} \sim \pi \right]. \tag{2.1}$$

2.7 Successor Features

A value function expresses the discounted sum of rewards from the states visited according to policy π. The core idea of successor features is to disentangle the return estimate of a value function into an estimation of the rewards from a state and the expected future state visitations. While the end goal is to estimate a value function, separating this knowledge can be a powerful tool. Successor representation was first proposed by Dayan (1993) for the tabular setting and was extended into the function approximation setting (Barreto et al., 2017). The successor features $\boldsymbol{\psi}(s, a)$ are the expected discounted sum of

the feature vector $\phi(s, a, s')$ that the agent will experience according to policy π. Any TD learning method can be used to learn $\boldsymbol{\psi}(s, a)$. Expressing $\boldsymbol{\psi}(s, a)$ recursively shows the connection to TD learning methods as follows,

$$\boldsymbol{\psi}(s, a) = \mathbb{E}_\pi[\phi(S_t, A_t, S_{t+1}) + \gamma_{t+1}\boldsymbol{\psi}(S_{t+1}, A_{t+1})|S_t = s, A_t = a].$$

To use successor features in estimating value functions, the reward needs to be estimated for a given state. Let $r(s, a, s')$ be the reward from transition s to s', then the estimation of the expected reward from state $\phi(s, a, s')$ is $r(s, a, s') = \langle \phi(s, a, s'), \mathbf{w} \rangle$, where $\mathbf{w}$ is the learned weights through any supervised learning method. To estimate the value function at $\phi(s, a, s')$, it is calculated as: $Q(s, a) = \langle \boldsymbol{\psi}(s, a), \mathbf{w} \rangle$. If parts $\psi(s, a)$ and $\mathbf{w}$ are well learned, it is equivalent to $Q(s, a)$. For convenience of notation, let $\phi_t \stackrel{\text{def}}{=} \phi(S_t, A_t, S_{t+1})$ and $\boldsymbol{\psi}_t \stackrel{\text{def}}{=} \boldsymbol{\psi}(S_t, A_t)$. The equivalence to $Q(s, a)$ can then be shown as,

$$\begin{aligned}
&\langle \boldsymbol{\psi}(s, a), \mathbf{w} \rangle = \mathbb{E}_\pi[\langle \phi_t, \mathbf{w} \rangle | S_t = s, A_t = a] + \mathbb{E}_\pi[\gamma_{t+1} \langle \boldsymbol{\psi}_{t+1}, \mathbf{w} \rangle | S_t = s, A_t = a] \\
&= r(s, a, s') + \mathbb{E}_\pi[\gamma_{t+1} \langle \phi_{t+1}, \mathbf{w} \rangle | S_t = s, A_t = a] + \mathbb{E}_\pi[\gamma_{t+1}\gamma_{t+2} \langle \boldsymbol{\psi}_{t+2}, \mathbf{w} \rangle | S_t = s, A_t = a] \\
&= r(s, a) + \mathbb{E}_\pi[\gamma_{t+1} r_{t+1} | S_t = s, A_t = a] + \mathbb{E}_\pi[\gamma_{t+1}\gamma_{t+2} \langle \boldsymbol{\psi}_{t+2}, \mathbf{w} \rangle | S_t = s, A_t = a] \\
&= \quad \ldots \quad = \mathbb{E}_\pi[r(s, a) + \gamma_{t+1} r_{t+1} + \gamma_{t+1}\gamma_{t+2} r_{t+2} + \ldots | S_t = s, A_t = a] \quad = Q(s, a).
\end{aligned}$$

Therefore, the value function can be expressed through the reward weights, $\mathbf{w}$, and the successor features, $\boldsymbol{\psi}$. There are two things to highlight. Firstly, the successor features can be a function of s' as that is a quantity known during learning. Secondly, ϕ_t is a feature vector for estimating the reward. A different feature representation can be used to estimate $\boldsymbol{\psi}(s, a)$.

2.8 Step Size Adaptation

Many learning algorithms work by receiving a gradient sample of the objective they are trying to minimize or maximize. The learning occurs as parameters are adjusted according to the sampled gradient. A prototypical example is semi-gradient TD(0) update shown in Algorithm 1.The step size α directly scales the magnitude of the change of the weights.

A simple choice for α is to use a fixed step size for all features over the entire learning process. However, we can do better if we allow the step size to change over time. Meta-descent methods adjust the step size in order to further improve performance. Jacobsen et al. (2019) showed the effectiveness of meta-descent methods in the linear function approximation setting. A key insight gained from Linke et al. (2020) is that step size adaptation in the GVF learners can be essential for enabling learners to be more introspective. In Linke et al. (2020), they use a meta-descent method called Auto for the bandits setting. The Auto algorithm, shown in Algorithm 8, is an iteration of the Autostep algorithm by Mahmood et al. (2012) that is adapted to the reinforcement learning setting. The modifications from the Autostep algorithm are from personal communications with an author of the original work Mahmood et al. (2012) and are not a contribution of this thesis.

Algorithm 8 Auto Update

$\mathbf{n} = \mathbf{n} + \frac{1}{\tau}\alpha\,|\phi| \cdot (|\mathbf{h} \cdot \delta\phi| - \mathbf{n})$
for all i such that $\phi_i \neq 0$ do
  $\triangle\beta_i = \text{clip}\left(-M_{\triangle}, \left|\frac{\mathbf{h}_i \delta\phi_i}{\mathbf{n}_i}\right|\right)$
  $\alpha_i = \text{clip}(\kappa, \alpha_i e^{\mu\triangle\beta_i}, \frac{1}{|\phi_i|})$
end
if $\alpha^T\mathbf{z} > 1$
  $\forall i$ such that $\mathbf{z}_i \neq 0$: $\alpha_i = \min(\alpha_i, \frac{1}{|\mathbf{z}|_1})$
end
$\theta = \theta + \alpha \cdot \delta\phi$
$\mathbf{h} = \mathbf{h}(1 - \alpha \cdot |\phi|) + \alpha\delta\dot{\phi}$

where:

- μ is the meta-step size parameter.
- α is the step sizes.
- δ is the scalar error.
- ϕ is the feature vector.
- $\mathbf{z}$ is the step-size truncation vector.
- θ is the weight vector.

- $\mathbf{h}$ is the decaying trace.
- $\mathbf{n}$ maintains the estimate of $|\mathbf{h} \cdot \delta\phi|$.
- τ is the step size normalization parameter.
- $M_{\triangle}$ is the maximum update parameter of α_i.
- κ is the minimum step size.

In all experiments, $M_{\triangle} = 1$, $\tau = 10^6$, $\kappa = 10^{-6}$. To use for the reinforcement setting, ϕ is the eligibility trace, δ is the td error of the algorithm and $\mathbf{z}$ is the overshoot vector calculated as $|\mathbf{e}| \cdot \max(|\mathbf{e}|, |\mathbf{x}_t - \gamma \mathbf{x}_{t+1}|)$.

Chapter 3

Problem Formulation and Solution Approach

3.1 Problem Formulation

We formalize the multi-prediction problem as the following setting: an agent acts in an MDP to more accurately learn N predictions. Each prediction is a GVF with a fixed policy as described in Sutton et al. (2011) and explained in Section 2.6. The error of the GVF learner is quantified by a weighted root mean-squared value error (RMSVE) across all states. The weighting distribution is according to some state distribution $d(S, A)$. At each timestep, the GVF learner, j, can be evaluated by the RMSE as,

$$\text{RMSVE}(\hat{Q}^{(j)}, Q^{(j)}) \stackrel{\text{def}}{=} \sqrt{\sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} d^{(j)}(s, a)(\hat{Q}^{(j)}(s, a) - Q^{(j)}(s, a))^2}. \tag{3.1}$$

The total error (TE) across all GVF learners is then,

$$\text{TE} \stackrel{\text{def}}{=} \sum_{i=1}^{t} \sum_{j=1}^{N} \text{RMSVE}(\hat{Q}_i^{(j)}, Q_i^{(j)}). \tag{3.2}$$

The goal of the agent is to minimize the TE by sculpting a stream of experiences that benefits the learning of the GVF learners in aggregate. The best agent is one that minimizes the TE by the last 10% of the steps.

In order to evaluate the agent on the TE, we calculated the true value for states offline using thousands of Monte Carlo rollouts. While there are ways to get accurate value estimates offline in complex environments (Sajed et al.,

2018), scaling the calculation of TE to complex environments that are difficult to simulate remains as future work.

3.2 Solution Method

The agent's goal is to have a data collection policy that results in the lowest TE. Creating a data collection policy from strictly following the GVF policies may not be good. Simply following the target policy of the GVF is insufficient for ensuring the value function is well learned. To understand why this would not work, imagine a long vertical hallway where the policy of the GVF is to always move forwards. If the agent starts in the middle of the hallway, it never learns what is behind it. Additional actions beyond what are in the GVF policy are required to learn the value function. Using the GVF directly by maximizing the cumulant of the GVF is also an unrealistic solution as it requires changing the policy in a way that may not result in informative behaviour. Consider a GVF predicting how many timesteps it will take to reach a wall by producing a cumulant value of 1 if the agent is touching the wall and a cumulant value of 0 if it is not touching the wall. If we attempt to maximize the cumulant, the agent will stay glued to a wall without learning how many timesteps it will take to reach a wall from states not touching it. Therefore, maximizing the cumulant could deliver an uninformative and inefficient data collection policy. The problem becomes harder when the parallel learning of many GVFs is required. Some GVFs may be easier to learn and require fewer observations to get an accurate value estimate while some GVFs are more difficult and require more observations. The data collection policy will need to balance the needs of the different prediction learners. Another consideration for the data collection policy is that some actions may be informative for a larger number of GVFs. In order to have an efficient multi-prediction system, the data collection policy should be aware of actions that benefit the learning of multiple GVFs. Lastly, there may be non-stationarities in the GVFs that result in an accurate value estimate no longer being accurate. The possible non-stationarities are described further in Section 5.1. The data collection policy needs to be able

recognize when learning is possible again for the prediction learners and adjust accordingly.

If the learning progress across the prediction learners can be quantified in an intrinsic reward, then the data collection policy can be the policy that maximizes the intrinsic reward. This fits neatly into the reinforcement learning framework and addresses the problems raised earlier on how an agent should behave to learn the value function of a policy. A possible intrinsic reward is the negative TE after the agent takes an action A_t in state S_t and transitions to S'. This rewards the agent for directly reducing the TE. Unfortunately, the total error is unknown to the agent because it requires the true value function to compute. Linke et al. (2020) performed a survey of possible intrinsic rewards in the bandits setting for multi-prediction learning and found the l_1-norm in the change of weights is an effective intrinsic reward if the prediction learners are *introspective*. Introspective learners are capable of autonomously modulating their rate of learning depending on the possible learning progress. In our research, we leverage this finding and use weight change as the intrinsic reward. In order to extend weight change to the MDP setting, we associate the intrinsic reward induced by the tuple (S_t,A_t,S_{t+1}) to state S_t.

The intrinsic reward can be calculated as,

$$r_{int,t+1} = \sum_{j=1}^{N} \|w_{t+1}^{(j)} - w_t^{(j)}\|_1. \tag{3.3}$$

In equation 3.3, $w^{(j)}$ are the weights parameterizing the GVF learner j.

For the experiments in this thesis, we calculate the intrinsic reward according to equation 3.4 where a small value of ϵ is used to encourage the agent to seek out new experiences,

$$r_{int,t+1} = \sum_{j=1}^{N} \|w_{t+1}^{(j)} - w_t^{(j)}\|_1 - \epsilon. \tag{3.4}$$

The final agent architecture is a behaviour learner learning a policy that maximizes the intrinsic reward of weight change generated by the N prediction learners. We use the Auto optimizer to encourage our agents to be more

introspective. A summary of our multi-prediction agent architecture is shown in algorithm 9.

Algorithm 9 Multi-Prediction Learning System

Input: N GVF questions

Initialize behavior policy parameters θ_0
and GVF learners $w_0^{(1)}, \ldots, w_0^{(N)}$
Obtain initial observation S_0
for $t = 0, 1, \ldots$ **do**
  Choose an action A_t according to $\pi_{\theta_t}(\cdot|S_t)$
  Observe next state vector S_{t+1} and γ_{t+1}
  // Update predictions with new data
  for $j = 1$ **to** N **do**
    $c \leftarrow c^{(j)}(S_t, A_t, S_{t+1})$
    $\gamma \leftarrow \gamma^{(j)}(S_t, A_t, S_{t+1})$
    Update $w_t^{(j)}$ with $(S_t, A_t, c, S_{t+1}, \gamma)$
  // Compute intrinsic reward, update behavior
  $r \leftarrow 0$
  for $j = 1$ **to** N **do**
    $r \leftarrow r + \|w_{t+1}^{(j)} - w_t^{(j)}\|_1 - \epsilon$
  Update θ_t with $(S_t, A_t, r, S_{t+1}, \gamma_{t+1})$

3.3 Summary

In this chapter we formalize the multi-prediction problem as the setting where there are N independent prediction learners. The goal of the agent is to reduce the TE, which is the sum of RMSVE across GVFs. We leverage the findings of Linke et al. (2020) who studied this problem in the bandits setting. We use their approach and cast it as a behaviour learning problem where the intrinsic reward is a proxy for learning progress. We rely on the conclusion of Linke et al. (2020) that weight change is an effective intrinsic reward if the N GVF learners are introspective. We present the general layout for a multi-prediction learning system in algorithm 9.

Chapter 4

Related Works

Linke et al. (2020) studied the multi-prediction problem in the bandits setting. Their agent consists of N learners estimating the value of a possibly non-stationary arm. The behaviour learner selects which arm to pull based on the intrinsic reward generated by the learners. Their work performed an extensive study of intrinsic rewards on different cumulant schedules. Linke et al. (2020) use several of their key findings (discussed in Section 3.2) and extend the problem setting to MDPs.

In the playground experiments of Chentanez et al. (2005), the agent creates and learns options in parallel through intrinsic reward. The intrinsic reward is a hand-crafted intrinsic reward signal specialized for the tabular environment. The agent moves in the playground and plays with different items and is evaluated on if it learns to perform a complex set of actions. The behaviour of the agent is guided by the intrinsic reward in addition to the extrinsic reward.

Riedmiller et al. (2018) proposes an agent based on scheduled auxiliary control (SAC) that utilizes GVFs to efficiently explore the environment in the sparse reward setting. A scheduler chooses which GVF should be the priority for learning over some time period. The agent acts to maximize the cumulant of the specific GVF until a new GVF is chosen. The scheduler chooses the GVF based on the extrinsic reward generated while learning the policy for the GVF.

Veeriah et al. (2019) develops an agent that discovers and answers GVF questions. While the agent acts to maximize the extrinsic reward, the agent

also learns on-policy GVFs and their associated predictions. The efficacy of the learned GVFs are evaluated based on the state representation they help craft for the behaviour of the agent.

Universal value function approximators (UFVA) take a different approach to learning many predictions in parallel (Schaul et al., 2015). Instead of explicitly learning a different value function per learner, the responsibility of learning different value functions is placed on the function approximator itself. If the different tasks can be parameterized by a vector, then the parameterized task can be used as part of the input space to the function approximator. The function approximator learns the different tasks and how to generalize between parameterized tasks, thereby effectively learning many different auxiliary tasks and allowing the interpolation of tasks never seen before.

Learning a model of the world around the agent can be viewed as an auxiliary task learning problem. In Kim et al. (2020), the agent learns a world model by rotating and observing the world around itself. The learning of the world model generates an intrinsic reward and the behaviour acts to maximize this signal. The intrinsic reward generated by the learning of a world model can also be used in conjunction with the extrinsic reward to aid in exploration (Burda, Edwards, Pathak, et al., 2019; Burda, Edwards, Storkey, et al., 2019; Pathak et al., 2017; Stadie et al., 2015).

Chapter 5

Non-Stationarity in a Multi-Prediction Setting

The world the agent finds itself in can be large and complex. In order to be an effective multi-prediction learning system, the agent needs to be capable of adapting to the non-stationarity that occurs throughout the system. In this chapter, we will highlight where sources of non-stationarity occur, why they pose a problem and provide strategies to mitigate their effects.

5.1 Sources of Non-Stationarity

There are many sources of non-stationarity found in a multi-prediction setting. In this section, we detail three of these sources: non-stationarity in the intrinsic rewards, cumulants, and state distribution faced by GVF learners.

The behaviour learner faces non-stationarity in the intrinsic rewards. To demonstrate this, let us consider the reward stream experienced by the behaviour learner and what it means for an intrinsic reward to be based on learning progress. In order for the rewards of the behaviour learner to be stationary, the distribution of rewards the agent receives remains the same for the agent's lifetime. An intrinsic reward based on learning progress is non-stationary as it depends on the agent's knowledge and environment. The non-stationarity of an intrinsic reward system based on learning progress is evident when the intrinsic reward is weight change. For example, let's consider an agent in a tabular grid world with a GVF learner learning a constant

cumulant. At first, the GVF learner changes its weights significantly because the prediction error is high. This results in a large intrinsic reward. Over time, the GVF learner learns to fully express the value function. Once this happens, there will be no more intrinsic reward or weight change. This shows that the reward distribution for a given state has inherently changed for the agent over time. If the agent is unable to adapt to this non-stationarity, it will continue to visit areas that are not informative anymore to the GVF learners. We want an agent that can effectively adapt its behaviour to maximize learning, so it is critical that the behaviour learner can handle this kind of non-stationarity in intrinsic rewards.

Another source of non-stationarity are the cumulants the GVF learners are trying to learn. Cumulants can be non-stationary due to their nature or can appear non-stationary because they depend on a hidden variable. Consider an animal living in the desert with a GVF learner modelling the amount of water in an oasis. The cumulant is the amount of water in the oasis. Due to evaporation or other animals drinking the water, water levels gradually change. The agent would not observe the animals drinking water, so the water in the oasis is a non-stationary target. Shifts in the underground water sources are a hidden variable that could also affect water levels. For a large scale system, it is important that the GVF learners themselves can adapt properly to a non-stationary cumulant.

An additional source of non-stationarity faced by the GVF learners is caused by behaviour policy changes (Patterson et al., 2021). In off-policy reinforcement learning, there are many common algorithms that weight their learning objective by the behaviour state distribution. Some of these algorithms include off-policy TD(λ), TB(λ) and GTD(λ) (Patterson et al., 2021). Recall that the mean squared projected bellman error (MSPBE) can be written as,

$$\text{MSPBE}(w) = \mathbb{E}_b[(\hat{v}_\pi(w) - \prod T\hat{v}(w))^2]. \tag{5.1}$$

In equation 5.1, $\prod$ is the projection operator which projects any value function to the nearest value function representable by the function approximator. The

Bellman operator, T, calculates the backup value estimate $R+\gamma P\hat{V}(w)$ where P summarizes the transition dynamics in matrix form. Notice that since the updates are according to state distribution d_b, the off-policy learner is learning an objective weighted by the behaviour state distribution. In the tabular setting, this is not an issue since any value function can be representable. In the function approximation setting, this leads to a non-stationarity in which the optimal solution is within the set of learnable functions. Different weightings on value estimates can change the optimal solution. In fact, Kolter (2011) found that weightings placed on the learning objective due to the behaviour state distribution can induce arbitrarily high value error. This occurs when there is a significant mismatch between the behaviour state distribution and the state distribution induced by the target policy. Since there are many off-policy learners in multi-prediction learning, it is important to consider the state distribution weighting the objective and the desired state distribution of the GVF learner.

In summary, we identify three sources of non-stationarity for our multi-prediction agent. The first is the non-stationarity inherent in an intrinsic reward representing learning progress. The second is that non-stationarity in the cumulants is a reasonable problem to occur. The third is that changing state distributions alters the learning objective of the off-policy learners.

5.2 Addressing Non-Stationarity in the Cumulants

To address the non-stationarity of the cumulants, we use successor features. As described in Section 2.7, successor features separate the learning of the cumulant estimate and the future state visitations according to policy π. This separation provides two distinct advantages.

Firstly, the successor features $\boldsymbol{\psi}(s,a)$ only need to be learned once as they summarize the transition dynamics of the MDP according to policy π. As long as there are no changes to the underlying dynamics of the MDP or the policy π on which $\boldsymbol{\psi}(s,a)$ is conditioned on, the successor features are reusable

regardless of changes to the reward or cumulant. This reusability is a primary reason successor features effectively transfer knowledge between tasks (Barreto et al., 2019). Successor features remain relevant for an entire experiment so they can be learned to a high accuracy.

The second advantage of successor features is that the effects of the cumulant distribution changing is isolated to updating the estimate for the immediate 1-step cumulant value. Generally, a small change in the cumulant can have a significant effect on the value estimate across the MDP. With successor features, change is localized to the specific states and is estimated by a regression problem. While the value estimate still changes across the MDP, isolating the non-stationarity into a supervised learning problem makes it an easier problem for tracking with faster convergence rates (McLeod et al., 2021). This also enables us to use optimizers designed for the supervised learning setting. Supervised learning setting is a large field of research which may provide additional performance increases to the updating of the cumulant estimate.

We clarify in algorithm 10 how successor features can be used for the nonstationary setting. We call this algorithm Successor Features for Nonstationary Rewards (SF-NR).

Algorithm 10 Successor Features for Nonstationary Rewards (SF-NR)

Input: $(S_t, A_t, S_{t+1}, C_{t+1}, \gamma_{t+1}), \pi, w_\psi, w_c$

$\phi \leftarrow \phi(S_t, A_t)$
$\hat{\boldsymbol{\psi}} \leftarrow \hat{\boldsymbol{\psi}}(S_t, A_t; w_\psi)$
$\hat{\boldsymbol{\psi}}' \leftarrow \sum_{a'} \pi(a'|S_{t+1}) \hat{\boldsymbol{\psi}}(S_{t+1}, a'; w_\psi)$
$\Delta \leftarrow 0$
for $m = 1$ **to** d **do**
  $\delta_m \leftarrow \mathbf{x}_m + \gamma_{t+1} \hat{\boldsymbol{\psi}}'_m - \hat{\boldsymbol{\psi}}_m$
  $\Delta \leftarrow \Delta + \delta_m \nabla \hat{\boldsymbol{\psi}}_m$
$w_\psi \leftarrow w_\psi + \alpha \Delta$
$w_c \leftarrow w_c + \alpha (C_{t+1} - \langle \phi, w_c \rangle) \phi$

5.3 Addressing Non-Stationarity in the Behaviour Learner

Successor features for non-stationary rewards is a promising way to combat non-stationarity in the cumulants or rewards for policy evaluation. To extend SF-NR for control, we apply General Policy Improvement (GPI) (Barreto et al., 2017). GPI works by greedifying over policies rather than over the actions of a specific policy. The set of policies for the behaviour can come externally or from the policies of the GVF learners themselves. Given a set of policies $\Pi = \{\pi_1, \ldots, \pi_N\}$, we learn the associated successor features for each policy, $\hat{\boldsymbol{\psi}}(s, a; w_\psi^{(j)})$, where j is the jth policy. We learn the weights $w_r \in \mathbb{R}^d$ such that $\langle \phi(S_t, A_t, S_{t+1}), w_r \rangle \approx \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$ where R_{t+1} is the intrinsic reward for that timestep. This allows the action value function for policy j at state s to be evaluated as $Q^{(j)}(s, a) = \langle \hat{\boldsymbol{\psi}}(s, a; w_\psi^{(j)}), w_r \rangle$. On each step, the behaviour policy takes the greedy action across policies,

$$\pi_b(s) = \operatorname*{argmax}_a \max_{j \in \{1,\ldots,N\}} \hat{Q}^{(j)}(s, a) = \operatorname*{argmax}_a \max_{j \in \{1,\ldots,N\}} \langle \hat{\boldsymbol{\psi}}(s, a; w_\psi^{(j)}), w_r \rangle.$$

The policy created by following this greedification across policies is guaranteed to be at least as good as any policy in the set of policies (Barreto et al., 2017). By using successor features for the policy evaluation, we leverage the finding that successor features enable faster convergence for non-stationary rewards. The intrinsic rewards are non-stationary, so the improved estimation is likely to facilitate more accurate value estimates.

5.4 Addressing Non-Stationarity Due To Changing State Distribution

We can use prior corrections to address the non-stationarity that changing behaviour policy induces in the GVF learners. Without prior corrections, the MSPBE learning objective is weighted by the behaviour distribution, $d_b(s, a)$ and is shown in Equation 5.1. This weighting happens because the behaviour state distribution governs the frequency of updates for the states.

If we reweight the updates, we can change the distribution by which the objective is weighted by. Alternative Life TD(λ), a type of prior corrections, reweights the updates by likelihood of reaching the state from the beginning of the episode under the target policy (Patterson et al., 2021). This completely removes the non-stationarity induced by the changing behaviour distribution because the MSPBE objective is now weighted by the state distribution induced by the target policy. In practice, this is not used as the updates are often zero or near zero when there are differences between target policy and behaviour policy. Emphatic methods are another approach that provides a third type of state distribution for the objective to be weighted by. While emphatic methods do not completely eliminate the non-stationarity, they may help reduce its effects because they include information of the target policy in their updates. Through prior corrections, emphatic methods weight the objective by the state distribution induced by target policy π with a starting distribution of states according to $d_b(s, a)$.

Reweighting the GVF learner updates according to distributions enables a direct relationship between the magnitude of the intrinsic reward and the distribution of interest. This has broad implications for how the learning of GVFs can be further specialized, which is discussed in Section 8.1. Emphatic methods use an interest function to guide the magnitude of the weight updates implicitly through the follow-on trace. In this thesis, we explore more directly manipulating the magnitude of the updates by explicitly using an interest function $I(S_t, A_t)$ to scale the update. This approach specifies the interest the GVF learner places on getting the value estimate correct. For convenience, we let $I_t \stackrel{\text{def}}{=} I(S_t, A_t)$. The following algorithm demonstrates how TB(λ) can be adapted to use an interest function.

Algorithm 11 Linear TB(λ) with Interest Update

$\mathbf{z}_t = \gamma_t \pi(A_t|S_t)\lambda_t \mathbf{z}_{t-1} + I(S_t, A_t)\mathbf{x}(S_t, A_t)$
$\delta_t = C_{t+1} + \gamma_{t+1} \sum_{a'} \pi(a'|S_{t+1})\hat{q}(S_{t+1}, a') - \hat{q}(S_t, A_t)$
$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha\delta_t \mathbf{z}_t$

Similar to off-policy TD(λ) or TB(λ), this update rule is not guaranteed

to converge in the function approximation setting with bootstrapping.

The derivation for online updating follows the same procedure used by Sutton and Barto (2018) to derive the TB update. This ignores the changes in approximate value function and can be written as,

$$G_t \approx \hat{q}(S_t, A_t, \mathbf{w}_t) + \sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i). \tag{5.2}$$

A forward view update that considers the interest at each time-step can be written as,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha I_t (G_t - \hat{q}(S_t, A_t, \mathbf{w}_t)) \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).$$

Substituting Equation 5.2 for G_t, we get

$$\mathbf{w}_{t+1} \approx \mathbf{w}_t + \alpha I_t \sum_{k=t}^{\infty} \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i) \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).$$

The sum of forward view update over time is,

$$\begin{aligned}
\sum_{t=1}^{\infty} (\mathbf{w}_{t+1} - \mathbf{w}_t) &\approx \sum_{t=1}^{\infty} \sum_{k=1}^{\infty} \alpha I_t \delta_k \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i) \\
&= \sum_{k=1}^{\infty} \sum_{t=1}^{k} \alpha I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \delta_k \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i) \\
&= \sum_{k=1}^{\infty} \alpha \delta_k \sum_{t=1}^{k} I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i).
\end{aligned}$$

This can be a backward-view TD update if the entire expression from the second sum can be estimated incrementally as an eligibility trace. This results in the eligibility trace being written as,

$$\begin{aligned}
\mathbf{z}_k &= \sum_{t=1}^{k} I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i) \\
&= \sum_{t=1}^{k-1} I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k} \gamma_i \lambda_i \pi(A_i|S_i) + I_k \nabla \hat{q}(S_k, A_k, \mathbf{w}_k) \\
&= \gamma_k \lambda_k \pi(A_k|S_k) \sum_{t=1}^{k-1} I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t) \prod_{i=t+1}^{k-1} \gamma_i \lambda_i \pi(A_i|S_i) + I_k \nabla \hat{q}(S_k, A_k, \mathbf{w}_k) \\
&= \gamma_k \lambda_k \pi(A_k|S_k) \mathbf{z}_{k-1} + I_k \nabla \hat{q}(S_k, A_k, \mathbf{w}_k).
\end{aligned}$$

Changing the index from k to t, the accumulating trace update can be written as,

$$\mathbf{z}_t = \gamma_t \lambda_t \pi(A_t|S_t)\mathbf{z}_{t-1} + I_t \nabla \hat{q}(S_t, A_t, \mathbf{w}_t).$$

Since we are considering the linear case, $\nabla \hat{q}(S_t, A_t, \mathbf{w}_t) = \mathbf{x}(S_t, A_t)$, the incremental update for estimating $\mathbf{w}_{t+1}$ reduces to what is shown in Algorithm 11.

5.5 Summary

In this chapter, we highlight three sources of non-stationarity faced by a multi-prediction system. The non-stationarity in the cumulants can be mitigated by using the SF-NR algorithm for GVF learning. Using the SF-NR algorithm, we can develop a control policy using GPI that addresses the non-stationarity inherent in the intrinsic reward of weight change. Lastly, we discuss that a changing behaviour state distribution induces non-stationarity in the optimal weight vector for the GVF learners in the function approximation setting. We explain why emphatic methods could be used to limit this effect.

Chapter 6

Environments

In this chapter, we explain the environment dynamics and the GVF specifications. Tabular TMaze and Continuous TMaze are used to test the multi-prediction learning system for a fixed behaviour policy and a learned behaviour policy. The mountain car is used as an environment not specifically designed for our experiments and is intended to demonstrate that complex behaviour policies can be learned as a byproduct in a multi-prediction setting. The Open 2D World is a large 2D open space that has similar GVFs to the Continuous TMaze.

6.1 Tabular TMaze

Tabular TMaze is a deterministic tabular gridworld with four actions: up, down, left and right. The state representation is a tabular representation for each cell block shown in Figure 6.1. The agent begins at the bottom of the maze, which is depicted in Figure 6.1.

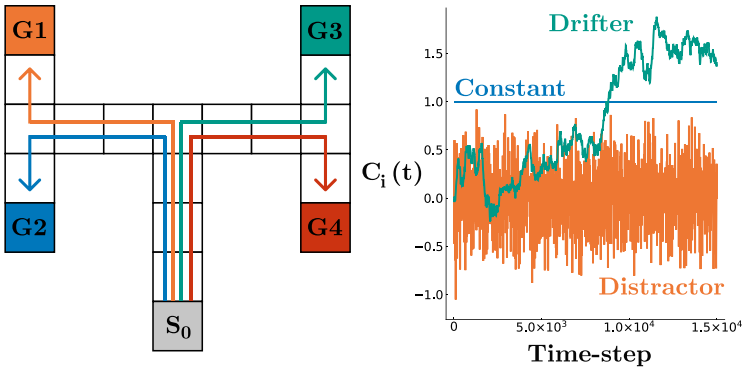


Figure 6.1: Tabular TMaze with 4 GVFs, with cumulants of zero except in the goals. The right plot shows the cumulants in the goals. G2 and G4 have constant cumulants, G1 has a distractor cumulant and G4 a drifter.

There are four GVFs being learned by the agent in this environment. Each of these GVFs has one of three possible cumulants schedules: drifter, distractor or constant. The *constant* cumulant is 0 everywhere except for at the goal state. At the goal state, it takes on a constant value uniformly selected between $[-10, 10]$. This value is randomized at the start of each run. In Figure 6.1, the cumulants corresponding to the lower left goal and lower right goal are constant goals. The *drifter* cumulant is 0 everywhere except for at the goal state. Starting at a value of 1.0 at the goal state, it drifts by a value of $N(\mu = 0, \sigma^2 = 0.01)$ at each timestep for the duration of the run. At the end of a run, it is reset back to 1. The drifter is located at the top right goal, G3, as depicted in Figure 6.1. The drifter cumulant value and the distractor cumulant value are capped between [-50,50]. The distractor cumulant is 0 everywhere except for at the goal state. At the goal state, the distractor cumulant is sampled from a normal distribution $N(\mu = 1, \sigma^2 = 25)$. The distractor goal is the top left goal as depicted in Figure 6.1. The following summarizes the cumulant schedules at the goal state for each GVF.

- Constant: $C_i^t = C_i$
- Distractor: $C_i^t = N(\mu_i, \sigma_i)$
- Drifter: $C_i^t = C_i^{t-1} + N(\mu_i, \sigma_i), C_i^0 = 1$

To define a GVF, we also need to specify the policy and the termination function. The policy for the GVF is the shortest path to the GVF's corresponding goal. The termination function is 0.9 for all transitions to non-goal states and 0 for all transitions to goal states.

In addition to the state representation, the SF-NR algorithm requires a feature representation that is used for estimating the cumulant values. In the Tabular TMaze, this feature representation is the same as the tabular representation.

6.2 Continous TMaze

The *Continuous TMaze* follows a similar design as the Tabular TMaze and is depicted in Figure 6.2. The GVFs are defined similarly in the Continuous TMaze as they were in the Tabular TMaze. The GVF cumulant schedules for each GVF are the same and the goals are in the matching corners. The policy is the shortest path to the GVF's respective goal. The termination function is 0.9 for all transitions to non-goal states and 0 for all transitions to goal states.

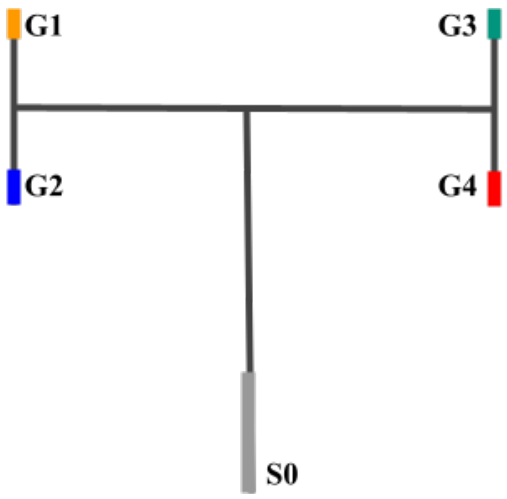


Figure 6.2: Continuous TMaze with 4 GVFs, with cumulants of zero except in the goals. G2 and G4 have constant cumulants, G1 has a distractor cumulant and G4 a drifter cumulant.

One major difference between the TMazes is that the Continuous TMaze is embedded in a continuous 2D plane between 0 and 1 on both axes. Each

hallway is a line with no width, allowing the agent to go along the hallway but not perpendicular to it. The main vertical hallway spans between [0, 0.8] on the y-axis and is located at x = 0.5. The main horizontal hallway spans [0, 1] on the x axis and is located at y = 0.8. Finally, the two vertical side hallways span between [0.6, 1.0] on the y-axis and are situated at x = 0 and x = 1 respectively. To switch between junctions, the agent must be within ± 0.05 unit distance from the junction. For example, to begin moving up or down on the left vertical hallway at x = 0, the agent must be in the left vertical hallway or within 0.05 units along the x-axis.

The goal states for each GVF match the corresponding corners of the Tabular TMaze. Goal 1 is at coordinates [0,1], goal 2 is at coordinates [0,0.6], goal 3 is at coordinates [1,1] and goal 4 is at coordinates [1,0.6]. Similar to the junctions, the agent must be within 0.05 unit distance from the (x,y) coordinates in order for them to be considered goal state. The agent's actions (up, down, left or right) moves the agent $0.08 \pm Uniform(-0.01, 0.01)$ units in the direction of the corresponding action.

Due to the Continuous TMaze being a continuous MDP, the feature representations are different between TMazes. The state feature is constructed with a tilecoder of 2 tilings of 8 tiles applied to the (x,y) coordinates of the agent. The feature representation for cumulant estimation is an indicator function for if $s' \in G_i$ from the tuple (s, a, s'). The reward feature vector is four dimensional since the Continuous TMaze has 4 goals. The reward feature vector is a reasonable feature vector because it is related to rewarding transitions. For the behaviour learner using SF-NR with GPI, it is unclear what is a rewarding transition apriori. Therefore, we take a state aggregation approach. The reward feature is the state-action vector of state-aggregation applied to the Continuous TMaze. Each line segment described above for the Continuous TMaze is broken up into thirds for state aggregation.

6.3 Open 2D World

The Open 2D World is an open continuous grid world with the dimensions 10×10 and is depicted in Figure 6.3. The GVF goals are located in each of the four corners and follow schedules similar to those used in the TMaze experiments. While the constants remain the same, the drifter parameters and the distractor parameters are different. In the Open 2D World, the cumulant values at the goal state are sampled by the distribution $N(\mu = 1, \sigma^2 = 1)$. The drift rate of the drifter is parameterized by the distribution $N(\mu = 0, \sigma^2 = 0.005)$. On each step, the agent can select between the four compass actions (up, down, right and left) and move 0.5 in the chosen direction with uniform noise $[-0.1, 0.1]$. A uniform $[-0.001, 0.001]$ orthogonal drift is also applied to each movement. The goals are squares in the corners with a size of 1×1. Similar to the TMaze variants, the GVF policies are defined as the shortest path to their respective goal. In the Open 2D World, there may be multiple actions that are part of the shortest path. These actions are weighted with equal probability in the target policy. The termination for the GVFs is $\gamma = 0.95$ for all transitions to non-goal states and 0 for transitions to goal states.

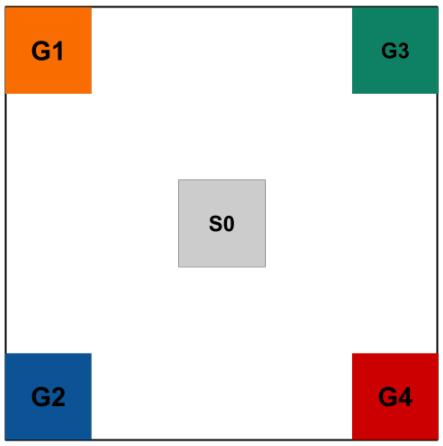


Figure 6.3: Open 2D World with 4 GVFs, with cumulants of zero except in the goals. G2 and G4 have constant cumulants, G1 has a distractor cumulant and G4 a drifter cumulant.

The state feature representation used by the agent is a tilecoded represen-

tation of 16 tilings of 2 tiles applied to the (x,y) coordinates of the agent. The reward features for the GVFs using successor features are defined similarly to the Continuous TMaze. The ith reward feature is an indicator function for if $s' \in G_i$. For the behaviour learner using successor features, it is unclear apriori what an optimal feature representation for estimating intrinsic rewards would be. We opt for a tilecoded representation of 8 tilings of 2 tiles on the (x,y) observation.

6.4 Mountain Car

In the mountain car environment, a car is situated at the bottom of a valley between two hills. Situated on each hill is a wall. The wall on the right hill is situated higher up the hill than the wall on the left hill. Therefore, an interesting aspect of the environment is that the car's throttle forwards action does not provide enough torque to reach the top of the right side of the hill. Thus, the car must build up momentum to successfully reach the right hilltop. The dynamics of the mountain car environment used in this experiment are from Sutton and Barto (2018) and are summarized below,

$$\dot{x}_{t+1} = \dot{x}_t + 0.001A_t - 0.0025\cos(3x_t)$$
$$x_{t+1} = x_t + \dot{x_{t+1}}.$$

The state can be described by a 2 dimensional representation, where x is the position of the car and $\dot{x}$ is its velocity. There are three actions available to the agent,

$$A_t \in [\text{Reverse} = -1, \text{Neutral} = 0, \text{Throttle} = 1]$$

. At the start of each episode, the agent begins in a position between [-0.2,0.2] and a velocity between [-0.04,0.04]. An episode terminates when the agent reaches the right wall located at 0.5. The left wall is located at the position of -1.2.

In the mountain car environment, we define two GVFs. The first GVF receives a cumulant value 1 when the agent reaches the left wall and is otherwise zero. It has a discount $\gamma = 0.99$, which terminates when the left wall is

touched. The second GVF is similar, but receives a cumulant value of 1 when reaching the top of the hill (i.e. the typical goal state). For each GVF, the policy is learned offline with ESARSA(λ) and an epsilon greedy policy with $\epsilon = 0.1$. It runs for 500k steps on a transformed problem where the cumulant is -1 per step. This results in a denser reward signal that enables learning a high quality policy. Note that the optimal policy that maximizes a cumulant of -1 per time step for all states is the same policy that maximizes the cumulant of the GVFs. The state representation used for these policies is an independent tile coder of 16 tilings of 2 tiles. The final policy that the GVFs are conditioned on is greedy with respect to the learned offline action values.

The state feature representation used by the agent is a tilecoded representation of 16 tilings of 2 tiles applied to the position and velocity of the agent. The reward features for the GVFs using successor feature is an indicator function for if $s' \in G_i$ from the tuple (s, a, s'). The reward feature vector is two dimensional since the Mountain Car environment has 2 goals.

Chapter 7

Experiments

In this chapter, we experimentally validate a multi-prediction learning system. This is the first system to adapt its behaviour to maximize learning in the function approximation setting. We seek to demonstrate that it is possible to adapt behaviour to improve multi-prediction learning. We aim to demonstrate the theoretical findings in McLeod et al. (2021) that show SF-NR enables faster correction to value estimates due to shifts in reward distributions. The use of successor features in both the GVF learners and the behaviour learner should each independently improve the agent.

A key component in evaluating a multi-prediction setting is to establish the prediction problems. In the TMaze experiments, we use four prediction problems. As discussed in Section 5.1, there may be non-stationarity in the cumulants of the GVFs that our learners need to adapt to. Therefore, one prediction problem includes a non-stationary cumulant. Since there are some prediction problems that are easier for an agent to learn, it is important for our agent to recognize its own competency and avoid focusing on familiar prediction problems. Thus, two prediction problems include a constant cumulant. In self-learning systems, the agent also needs to be able to recognize when something is unlearnable or simply noise (Schmidhuber, 2008) to avoid succumbing to the noisy-tv problem. As such, our final prediction problem includes a noisy cumulant. These four cumulant schedules are from Linke et al. (2020) and adapted into the MDP setting. Section 6.1 contains the details of GVF learning problems. The drifter signal models the non-stationarity the

agent must be able to adapt to, the distractor signal simulates the unlearnable signal the agent must learn to ignore, and the constant cumulants represent the easy to learn prediction tasks.

We also evaluate the full learning system where the behaviour is being learned. It is important to consider what kind of behaviour the agent should exhibit in this setting. The GVF learners for the constant cumulant should be quickly learned and not revisited. For the distractor cumulant, the agent should learn the mean of the Gaussian distribution and learn to ignore the high error signal from that GVF learner. It should stop visiting the distractor goal as it is unlearnable. Lastly, the agent should continue to visit the drifter goal as it always presents a learnable signal. By the end of the experiment, the agent should primarily visit the drifter goal.

7.1 Evaluating Off-Policy Learners

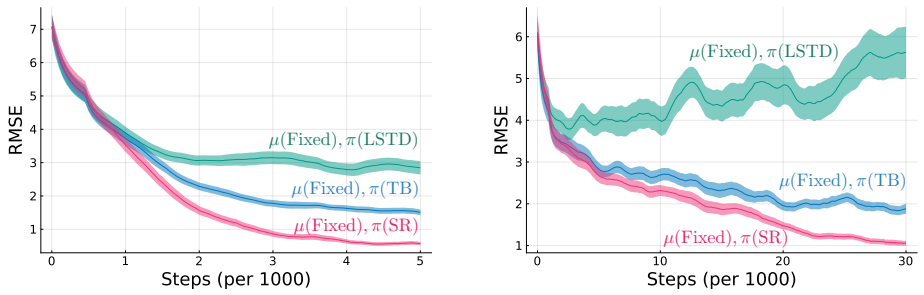
The purpose of this experiment is to evaluate the off-policy learners in the TMaze environments to better understand their effectiveness under a reasonable data collection policy. The data collection all agents follow is called the *fixed behaviour policy*. The fixed behaviour policy involves the agent taking the shortest path towards the closest goal. If two or more goals are the same number of steps away but require a different action, the probability of taking the respective actions is equally weighted. This only occurs at the three junctions of the Tabular TMaze and Continuous TMaze. To ensure that all state action pairs are visited, we use exploring starts. Except for the initial random action due to exploring starts, each episode follows precisely one GVF policy to termination and that all GVF policies are equally likely to be executed in expectation.

The three different off-policy learners evaluated were TB(λ), LSTD(λ), and SF-NR. TB(λ) was chosen to be a stand-in for a typical value based learner. The LSTD(λ) learner was used to highlight how well the stationary targets could be learned and show the detrimental effects of applying an assumption of stationarity to the drifter cumulant. The SF-NR learner was the proposed

method and was intended to perform better at tracking the drifter cumulant. The RMSVE for each GVF learner was weighted according to the state distribution induced by the fixed behaviour policy. This was estimated offline by performing a large number of rollouts and randomly selecting 500 state-action pairs from the set of trajectories that were used as an evaluation set for calculating the RMSVE. The true value of each state-action pair was calculated by averaging over Monte Carlo rollouts starting from those state-action pairs.

The Auto optimizer described in Section 2.8 was used for the GVF learners. For the Tabular TMaze environment, the metastep size parameter for the GVF learners was swept over the range of $[5^{-4}, ..., 5^{0}]$. The Auto optimizer also had an initial step size parameter. We initialized with two different initial step sizes: $[0.1, 1.0]$. A low initial step size of 0.1 represented the idea that it is better for learning to start small and grow as the learning rate is adjusted by the Auto optimizer. An initial step size of 1.0 represented the idea of learning aggressively and letting the Auto optimizer tune down the learning rate as needed. The optimal parameters in the tabular case was a metastep size of 5^{-1} and an initial step size of 1.0 for both SF-NR learner and TB(λ) learner. For the LSTD(λ) learner, the η parameter scaled the initialization of the $\mathbf{A}$ matrix. The $\mathbf{A}$ matrix was swept along powers of 10 in the range of $[10^{-2}, ..., 10^{3}]$. The optimal η value was 10^{3}. When testing on the Continuous TMaze, the initial step size of Auto took the range of values [0.01,0.1,0.2] and then was scaled down by the number of tilings used in the representation. For both TB(λ) learners and SF-NR learners, the optimal meta-step size was 5^{-2} and the initial step size was 0.2. For the LSTD(λ) learner, the η parameter was swept along powers of 10 in the range of $[10^{-2}, ..., 10^{3}]$. The optimal η value was 10^{1}.

Figure 7.1 shows the TE of each agent averaged over 30 runs on the Tabular TMaze and Continuous TMaze environments with the shaded region being the standard error. While LSTD(λ) initially performs well, the TE plateaus by the end of the experiment. Figure 7.2 shows the TE broken down into the individual RMSE contribution per GVF learner of each agent. The top left plot is the RMSVE of the GVF learner for the distractor cumulant, the bottom



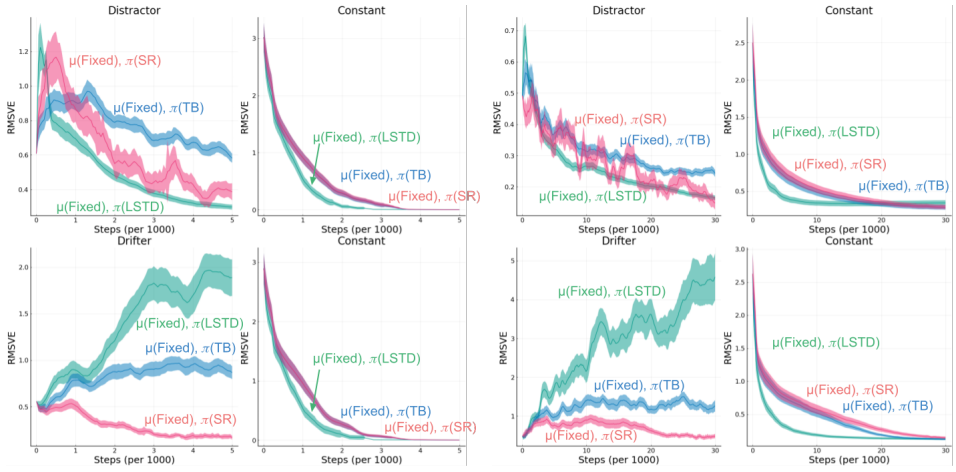
(a) Tabular TMaze environment. (b) Continuous TMaze environment

Figure 7.1: Total Error while following a fixed behaviour policy averaged over 30 runs with shaded region being standard error. RMSE weighting is the state distribution induced by the fixed behaviour.

left plot is the RMSVE of the GVF learner for the drifter cumulant and the two right side plots are for the RMSVE of the GVF learners for the constant cumulants.

Of the three learners in the Tabular TMaze, the LSTD(λ) learner has the highest TE and performs the worst on the constant GVFs in addition to the drifter GVF. This is evident in Figure 7.2a. It is possible that it performs poorly due to the hyperparameter selection reducing error across GVFs. The optimal η value for the drifter GVF is unlikely to be the optimal η value for the constant GVFs. The SF-NR learners perform significantly better than the TB(λ) learners as demonstrated in Figure 7.1a. The SF-NR learners' biggest advantage over the other learners is that they are significantly better at tracking the drifter GVF. This aligns with the theory that value function estimation with successor features should enable faster tracking if the successor features are well learned. Extending to the function approximation setting is critical for building any large scale system.

The findings remain consistent in the Continuous TMaze as shown in Figure 7.1b. The agent is able to learn accurate enough successor features so the SF-NR learners can more effectively propagate the drifter value throughout the MDP. Similar to the Tabular TMaze, the LSTD(λ) learners do not perform well on the drifter GVF as shown in Figure 7.2b.



(a) Tabular TMaze environment. (b) Continuous TMaze environment

Figure 7.2: RMSE error per GVF learner while following a fixed behaviour policy averaged over 30 runs with shaded region being the standard error. RMSE error weighting is the state distribution induced by the fixed behaviour.

7.2 Evaluating Learned Behaviour

In this set of experiments, we let the agent learn its own control policy from the intrinsic reward of weight change in two environments: Tabular TMaze and Continuous TMaze. In doing so, we aim to understand the effects that the SF-NR learning algorithm has on the learning system as a whole; this includes the effect that the SF-NR learning algorithm has as a GVF learner and the effect of the SF-NR learning algorithm used in conjunction with GPI. These experiments test two control algorithms; Expected Sarsa(λ) and GPI applied to SF-NR learners. Expected Sarsa(λ) is a stand-in for a generic control algorithm. GPI with successor features applies generalized policy improvement to derive the control algorithm. How GPI with the SF-NR learning algorithm is used as a control algorithm is explained in Section 5.3.

In this set of experiments, we combined the possible GVF learners with the possible behaviour learners to test the effects of the SF-NR learning algorithm on both learners. This resulted in four possible agents: GPI SF-NR learner for control and SF-NR for the GVF learners, GPI SF-NR learner for control and TB(λ) for the GVF learners, ESARSA(λ) for control and SF-NR learners for the GVF learners, and ESARSA(λ) for control and TB(λ) for the GVF

learners. For the remainder of Chapter 7, SF-NR learner refers to a GVF learner that uses the SF-NR learning algorithm, and GPI SF-NR refers to the control algorithm leveraging generalized policy improvement applied over learned successor features.

For the experiment run in the Tabular TMaze and the experiment run in the Continuous TMaze experiment, a hyperparameter sweep over the initial step size and meta-step size was conducted. The behaviour learner and the GVF learners shared the same initial step size and meta-step size parameters in the sweep. For the Tabular TMaze environment, the meta-step size swept was from $[5^{-4}, ..., 5^{0}]$ and initial step sizes were $[0.1, 1.0]$. All four agents had an optimal meta-step size of 5^{-2}. For agents using SF-NR learners, the optimal initial step size was 1.0. For agents using the TB(λ) GVF learners, the optimal initial step size was 0.1. For the Continuous TMaze experiment, the meta step size parameter was swept over the range of $[5^{-4}, ..., 5^{0}]$. The initial step size was swept over [0.01,0.1,0.2] and divided by the number of tilings in the representation. The optimal meta step size was 5^{-3} for the SF-NR learners and 5^{-2} for the TB(λ) GVF learners. The optimal initial step size was 0.2 (then divided by the number of tilings) for agents using SF-NR learners and 0.1 for the agents using TB(λ) learners.

In these experiments, the behaviour learner was optimistically initialized to ensure that the agent visited each of the four GVF goals at least once. To the best of our knowledge, no one has studied optimistic initialization for successor features. To perform optimistic initialization on successor features, we initialized the $w_{\boldsymbol{\psi}}$ estimates to be $\frac{1}{d}$, where d was the state feature length. We initialized the immediate reward estimates to be $\frac{\text{opt}}{m}$, where m was the length of the successor feature representation and opt was the optimistic threshold. This initialization resulted in each successor feature having an expected discounted value of 1 and the reward estimates scaled such that the final value estimate was equal to opt. Since the intrinsic reward is based on weight change, there was no maximum reward to calculate the appropriate optimistic initialization required. Empirically, we found that opt = 10 is sufficient to ensure all goals are visited. All behaviours applied ϵ-greedy policy with a fixed ϵ of 0.1. All

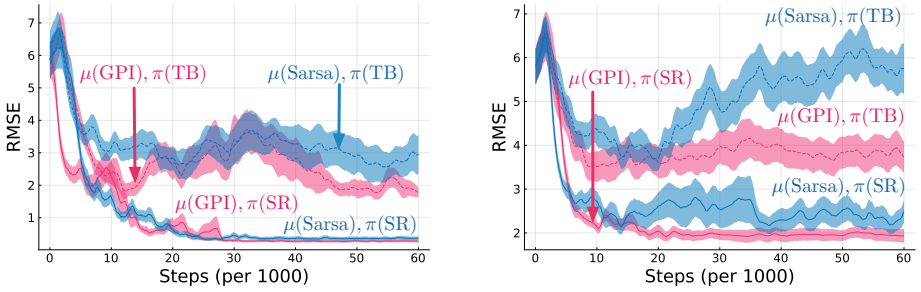
plots were selected for the lowest TE on the last 10% of the runs. The small penalty per step used in the intrinsic reward was 0.01.

Figure 7.3 shows the TE of each agent averaged over 30 runs on the Tabular TMaze and Continuous TMaze environments with the shaded region being the standard error. Figure 7.4 shows the TE broken down into the individual RMSE contribution per GVF learner of each agent. Each plot per environment contains four smaller plots. The top left plot is the RMSVE of the GVF learner for the distractor cumulant, the bottom left plot is the RMSVE of the GVF learner for the drifter cumulant and the two right side plots are for the RMSVE of the GVF learners for the constant cumulants. Figure 7.5 shows the goal visitation plots of the agent. For each environment, the top left plot is GPI SF-NR with TB(λ) GVF learners, the top right is GPI SF-NR with SF-NR learners, the bottom left is ESARSA(λ) with TB(λ) and the bottom left is ESARSA(λ) with SF-NR learners.

The SF-NR learners consistently improve overall performance of the agent for all behaviour learners across the Tabular TMaze and Continuous TMaze environments. The added benefit of SF-NR aligns with the findings of the fixed behaviour policy evaluation, where SF-NR learners perform significantly better than TB(λ) learners across environments. Figure 7.4 demonstrates that the agent using SF-NR has significantly lower error on the drifter GVF in particular. This is likely due to the SF-NR learners themselves being more efficient learners and more introspective learners. The efficacy of SF-NR learners is shown in the fixed behaviour policy experiments and supported by the theoretical findings of McLeod et al. (2021). The idea that SF-NR learners are more introspective learners is supported by the stark difference in behaviour between agents when SF-NR is used. Figure 7.5 shows the goal visitation plots for the Tabular TMaze and Continuous TMaze. When the SF-NR learner is substituted for TB(λ) learners, the distractor goal is ignored *much* more quickly and the drifter goal is correctly identified as the only long term source of reducing TE. As described in the introduction of Chapter 7 the agent should learn to prioritize visiting the drifter goal since it is only cumulant with non-stationarity. While the agents with TB(λ) GVF learners

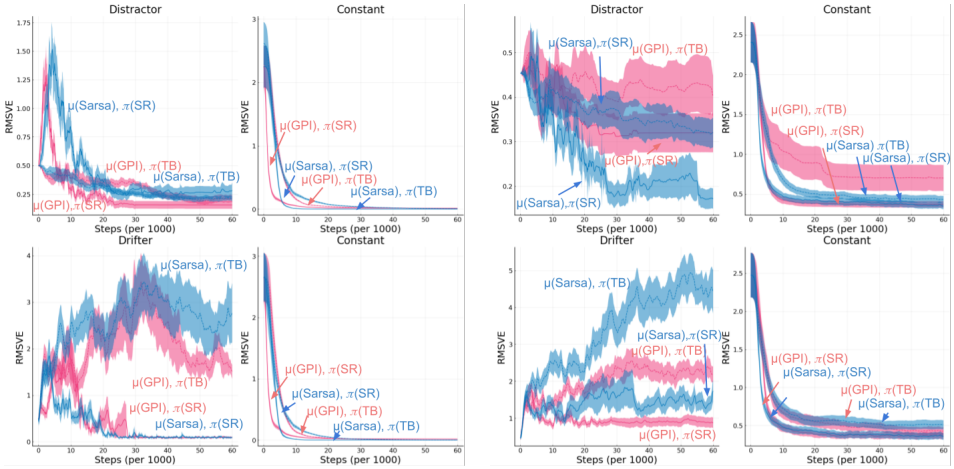
trend in the direction of favouring the drifter, the SF-NR learners achieve this desired behaviour more quickly and more consistently.

GPI SF-NR improves the agents as well. In the Tabular TMaze, the TE is lower for TB(λ) learners when GPI SF-NR is used over the ESARSA(λ) behaviour learner. Examining the goal visitation plot shown in Figure 7.5, GPI with TB(λ) GVF learners visit the drifter goal more frequently than the ESARSA(λ) behaviour learner. The negligible difference between the GPI SF-NR behaviour learner and the ESARSA(λ) behaviour learner when the SF-NR learners are used could be due to the tabular domain being too easy. With the SF-NR learners, both behaviour learners quickly identify the drifter and develop a near optimal behaviour policy. Thus, it is unlikely for the two behaviour learners to have significantly different TEs. The Continuous TMaze includes function approximation that makes the differences between behaviour learners more pronounced. The GPI SF-NR learner results in lower TE regardless of the GVF learners used. In addition, the overall behaviour in the goal visitation plots is significantly different when GPI is used. The GPI SF-NR learner results in persistent visitation to the drifter while the ESARSA(λ) behaviour learner struggles to develop a similar policy throughout the experiment. This suggests that the GPI SF-NR learner adapts to the non-stationary rewards more quickly and isolates a source of persistent intrinsic reward.



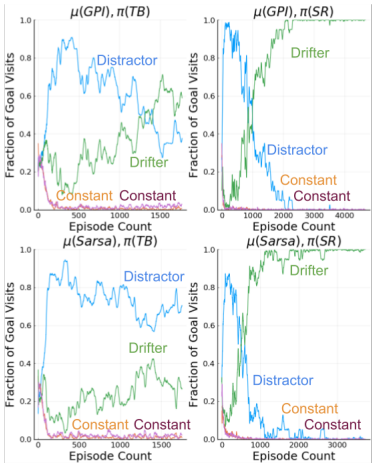
(a) Tabular TMaze environment. (b) Continuous TMaze environment

Figure 7.3: TE of GVF learner for a learned behaviour averaged over 30 runs with shaded region being standard error. RMSE error weighting is the uniform state distribution. Blue lines correspond to behaviour learned through ESARSA(λ) while the pink lines correspond to behaviour learned through GPI SF-NR. Solid lines are for the agents using SF-NR learners while the dashed lines are using TB(λ) GVF learners.

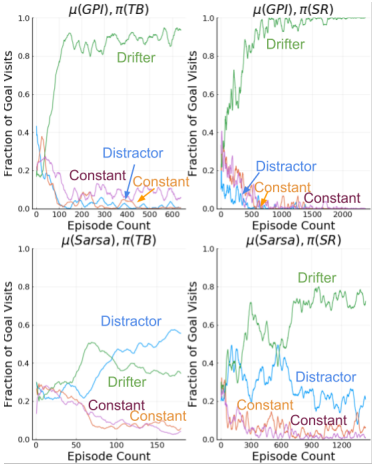


(a) Tabular TMaze environment. (b) Continuous TMaze environment

Figure 7.4: TE of GVF learners for a learned behaviour averaged over 30 runs with shaded region being standard error. RMSE error weighting is the uniform state distribution. Blue lines correspond to behaviour learned through ESARSA(λ) while the pink lines correspond to behaviour learned through GPI SF-NR. Solid lines are for the agents using SF-NR learners while the dashed lines are using TB(λ) GVF learners.



(a) Tabular TMaze environment



(b) Continuous TMaze environment

Figure 7.5: Goal visitations across 30 runs of 60k steps truncated at the shortest number of episodes. For each environment, the top left plot is GPI SF-NR with TB(λ) GVF learners, the top right is GPI SF-NR with SF-NR learners, the bottom left is ESARSA(λ) with TB(λ) and the bottom right is ESARSA(λ) with SF-NR learners.

7.3 Learning Complex Behaviours through Intrinsic Rewards

Continuous TMaze and Tabular TMaze are environments specifically designed to test our multi-prediction learning system. The GVF policies in both environments have significant overlap of actions in reaching their respective goals due to the narrow hallways. This results in experiences that are generally useful to all learners. Through the mountain car environment, we test if an agent is able to construct a policy to effectively learn its prediction tasks in a more challenging environment not specifically designed for multi-prediction learning.

The mountain car environment, described in section 6.4, is a common environment to test algorithms because of its dynamics. A complex policy of building up momentum is required to gain enough kinetic energy to reach the top right side of the hill. Thus, a random policy is highly unlikely to succeed in reaching the top. The dynamics result in value functions where states with only small differences in position or velocity can have a large effect. Mountain

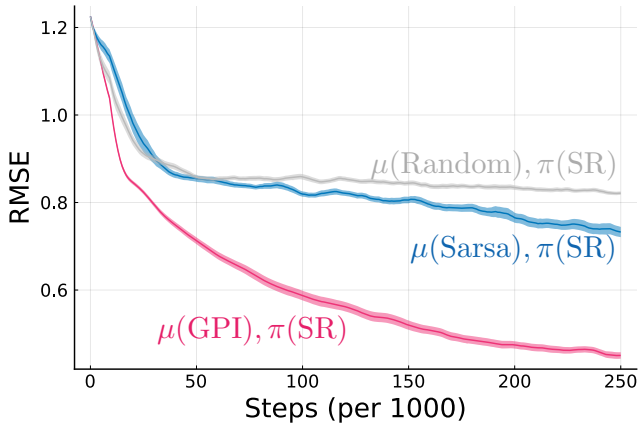


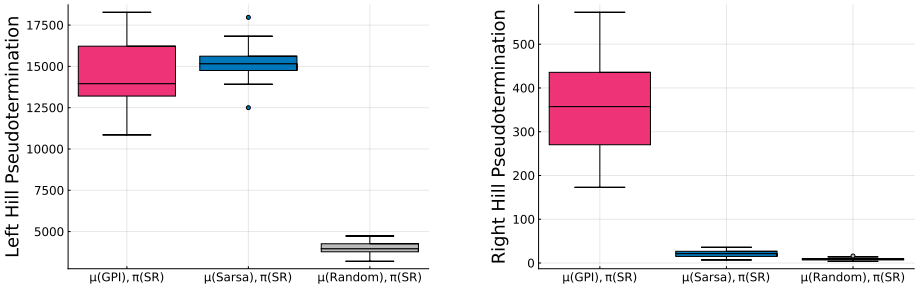
Figure 7.6: TE averaged over 30 runs with shaded region being the standard error on the mountain car environment. The state distribution used is a uniform sampling across the state space.

car can be adapted to have auxiliary tasks by placing goal locations in the environment.

To get a baseline, we ran a random behaviour policy with SF-NR GVF learners. We evaluated two agents in this domain: GPI with SF-NR GVF learners and ESARSA(λ) with SF-NR GVF learners. The behaviour learner and GVF learner were optimized separately with stochastic gradient descent and were swept over the range of $[3^{-3}, ..., 3^{0}]$. For GPI SF-NR with SF-NR learners, the GPI SF-NR learners' best performing step size was 3^{0} with the SF-NR learners using a step size of 3^{-2}. For the ESARSA(λ) agent with SF-NR learners, the ESARSA(λ) used a step size of 3^{0} while the SF-NR learners used a step size of 3^{-2}. The step size of the SF-NR learners for the random behaviour policy was 3^{-1}. For the two learned behaviour policies, a fixed ϵ-greedy exploration strategy was swept with ϵ over the range of $[0.1, 0.3, 0.5]$ with $\epsilon = 0.1$ being the best for both agents. The small penalty per step used in the intrinsic reward was 0.01.

Figure 7.6 shows the TE for the agents averaged over 30 runs for the 250k timesteps with the shaded region being the standard error. Figure 7.7 summarizes the behaviour of the agent by showing a boxplot of the GVF goal visits for the agent.

Both agents significantly outperform the random behaviour agent. This



(a) Left Wall Pseudo-terminations (b) Right Wall Pseudo-terminations

Figure 7.7: The goal visitation plot for the GVFs in the multi-prediction learning system averaged over 30 runs. The horizontal line of the box plot shows the median number of goal visits per algorithm. The whiskers are the maximum or minimum number of GVF goal visits up to the 1.5 times the inner quartile range.

is promising to the field because it demonstrates that an agent can actively adapt its behaviour to enhance learning across GVF learners. Figure 7.7 highlights the difference between a behaviour learned through ESARSA(λ) and a behaviour learned through GPI. Both behaviour learners are effectively able to learn to reach the left wall as it is situated on a lower hill. Reaching the right wall requires a more complex policy of energy pumping; only the GPI learner is able to effectively reach this goal by the end of the run. This indicates that the GPI SF-NR learner is able to more effectively learn from a series of non-stationary rewards and learn policies for the multi-prediction setting.

7.4 Summary

In summary, we examined the effects of SF-NR learners on multi-prediction learning by running three experiments. For a fixed behaviour policy, we find that SF-NR learners significantly outperform TB(λ) learners and LSTD(λ) learners. Their better performance is primarily due to their better estimate of the drifter goal, a result that carries over from the Tabular TMaze to the Continuous TMaze. In the case of learned behaviour, SF-NR learners continue to offer improvement over TB(λ) learners and each agent system is strictly improved when the SF-NR learning algorithm is used in the TMaze environments. Due to the distinct differences in behaviour induced by SF-NR learners,

we conclude that SF-NR learners are more introspective learners in the TMaze environments. When successor features are used for control, the GPI SF-NR behaviour learner improves the agent as a whole. The GPI SF-NR learner offers mild improvements in the Tabular TMaze and larger improvements in the Continuous TMaze. The lower TE and behavioural differences found in the experiment suggest that the GPI SF-NR learner is able to more effectively adapt to non-stationary intrinsic rewards. The GPI SF-NR behaviour learner being more effective at adapting to non-stationary rewards is further supported by the mountain car experiment. In the mountain car experiment, the GPI SF-NR learner performs significantly better than the ESARSA(λ) behaviour learner by reducing the TE and crafting a policy that consistently reaches both goals of the GVFs. Both behaviour learners create a better policy than a random policy for reducing the TE and reaching the GVF goals. An overarching result is that an agent can adapt its behaviour to maximize learning across its GVFs.

Chapter 8

Extensions

In this chapter, we investigate two different ways to extend our work to address more challenging problems. We explain why each extension is important and provide preliminary experimental evidence supporting the ideas discussed in each section. Section 8.1 discusses why we would want GVF learning to be areas of interest. We propose methods to address this challenge and perform a supporting experiment to test the efficacy. In Section 8.2, we discuss how experience replay can be properly added to the SF-NR learning method and perform a supporting experiment demonstrating its success.

8.1 Directing Areas of Interest

In the previous experiments, we valued the accuracy of an agent's value estimates in all states equally. There are circumstances where valuing the estimates equally in all states may not be desired. For example, a general purpose cleaning agent can have hundreds of different GVF learners modelling different components of cleaning - not all of these GVFs are applicable to all cleaning tasks. In vacuuming, for instance, an agent would not benefit from having a GVF modelling the chances of the agent breaking a plate.

Since the agent will learn the value function in the desired area of interest as a byproduct of learning it in all areas, it is tempting to overlook this matter. This oversight could harm the effectiveness of the agent for a few reasons. In large MDPs with a small area of interest, the agent is distracted by learning the value estimate in areas outside of its interest and thereby re-

duces its efficiency in learning. Another consequence of this oversight is due to the limited capacity of our function approximators. As discussed in Section 5.1, the objective that off-policy learners are optimizing is weighted by a state distribution, usually the behaviour state distribution. We should have the function approximator dedicate its resources to getting accurate value estimates in the states of interest rather than the estimator using its resources on unimportant states. We could improve our value estimates for the states of interest if we can inform our agents as to which states are important. Since our intrinsic reward is weight change, rescaling the weight updates shapes the intrinsic reward. If the updates are scaled down in uninteresting areas for the GVF, then the behaviour learner will have less incentive to visit those areas.

The Open 2D World, described in Section 6.3, is a good environment to test if an agent can leverage its knowledge of which states are important. Each GVF is only interested in the closest one fourth of the MDP to the GVF's respective goal. The shaded region of Figure 8.1 shows the area of interest for each GVF. The evaluation set used for calculating the total error reflects the areas of interest for the respective GVF. Each GVF learner is evaluated on a data set of randomly selected states within the shaded region. The true value function is calculated by Monte Carlo rollouts according to the target policy over a large number of iterations. Without an agent leveraging its knowledge of which states are important, its function approximation resources will be more constrained and the agent may waste time learning the action value function for areas that are not important.

We evaluated three different agents in the Open 2D World. Each agent used GPI SF-NR as the behaviour learner without any prior corrections on the learning of its successor features. The agents used different prior corrections for the GVF learners. The first agent did not use any prior corrections in the GVF learners and was learning the entire MDP with the learning objective weighted by the behaviour distribution. The second agent used Emphatic TB(λ) for learning the successor features in the GVF learners. The interest function for the emphatic method corresponded to 1 in the shaded region of interest and was otherwise 0. We used an emphasis clipping of 10 to correct issues with the

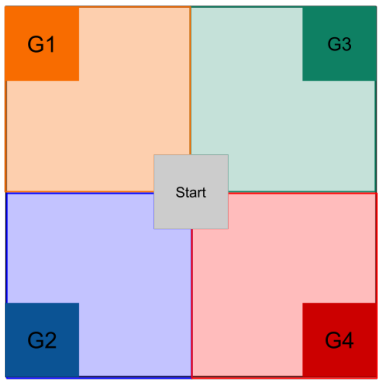


Figure 8.1: The state distribution used in calculating the total error. The shaded quadrant represents the state distribution used for each GVF learner.

variance of the updates. The last agent used Interest Reweighting as described in Section 5.4. The interest function was 1 if the state-action pair was in the shaded region and otherwise was 0.

All three agents used the Auto optimizer for the GVF learners and the behaviour learner. The initial step size was 0.1 divided by the number of tilings in the tile coded representation. The meta step size parameters were swept over the range of $[5^{-4}, ..., 5^{0}]$. This was done independently for the GVF learners and the behaviour learner. The results were averaged over 30 runs with the best hyperparameter being selected by the lowest TE on the last 10% of the run. The small penalty per step used in the intrinsic reward was 0.005.

Figure 8.2 shows the TE averaged over 30 runs with the shaded region being the standard error. ETB(λ) and Interest TB(λ) result in lower TE than using TB(λ) with no update reweighting. Figure 8.3 breaks the TE down by RMSVE per GVF. Lastly, Figure 8.4 summarizes the behaviour of the agents by plotting the goal visitations.

ETB(λ) and Interest TB(λ) perform at the same level or better than TB(λ) across the four GVFs. It is possible that the agents using interest learn more accurate successor feature estimates for the areas of interest. This could be due to the function approximator dedicating more resources to getting the

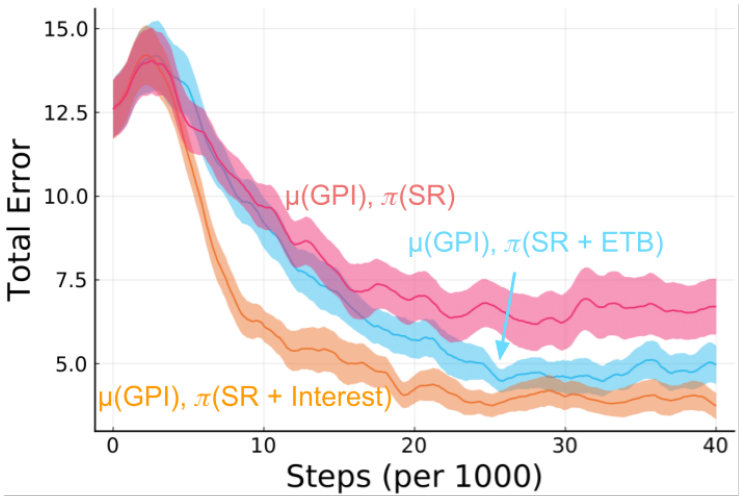


Figure 8.2: Comparison in 2D Open World where the agent uses the same behaviour learner but three different GVF learners. The red line is the agent learning successor features with TB(λ), the blue is learning the successor features with ETB(λ) and the orange is learning with Interest TB(λ)

successor feature estimate correct in those areas, or due to the intrinsic reward shaping the behaviour relevant to learning the successor features. Further investigation is needed to separate the corresponding effects. The results from this experiment suggest that agents can leverage knowledge of which states are of interest to improve their corresponding value estimates.

8.2 Enabling Replay

Experience replay is a common technique used in many state of the art reinforcement learning systems (Hessel et al., 2018). As the agent experiences the world, it stores its experiences in a buffer called a *replay buffer*. The agent periodically draws samples of experiences from its replay buffer to relearn on them. Storing real experiences in the replay buffer ensures that the samples are valid experiences that can be used to improve the agent's value function or policy (Lin, 2004). In the previous experiments, the agent has used online learning with eligibility traces. In order to move to the deep reinforcement learning setting, the agent will need to work with the more modern approach of experience replay.

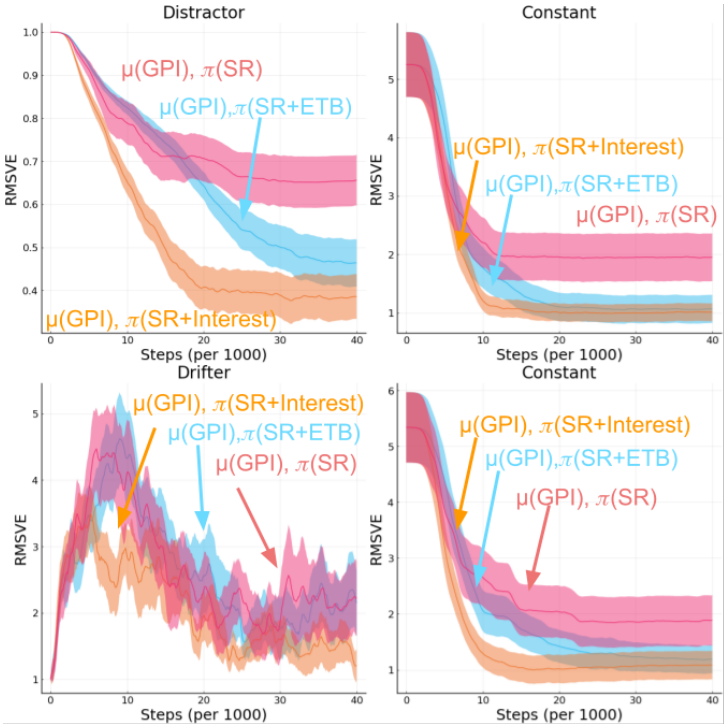


Figure 8.3: RMSE per GVF learner averaged over 30 runs with shaded region being the standard error.

A barrier to using a replay buffer for the multi-prediction setting is that it is difficult to adapt to the non-stationarity faced by the GVF learners. To understand this barrier, consider the effects of storing the drifter and distractor cumulants. Updating the GVF learners from samples randomly drawn from the buffer results in the estimate approaching the mean value stored in the buffer. This is beneficial for the distractor cumulant as the mean value of the buffer smooths out the noisy signal. This is detrimental for the drifter cumulant since the most recent sample is the best estimate and the mean value in the buffer is a lagging estimate. Figure 8.5 shows the detrimental effect of applying replay with the ADAM optimizer to the TB(λ) learners.

Separating SF-NR learners into a stationary component (the learning of successor features) and a potentially non-stationary component (the cumulant estimates) enables us to apply experience replay with ADAM to the stationary component of the learning problem without smoothing over the non-stationary component. Algorithm 12 shows how replay can be introduced to the SF-NR

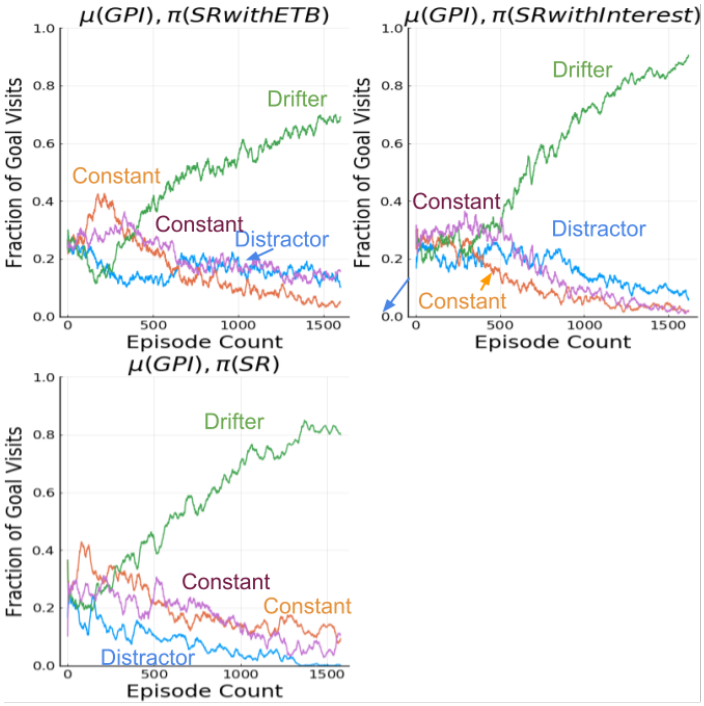


Figure 8.4: Goal visitation plot averaged over 30 runs truncated at the run with the shortest number of episodes. The top left plot is with the GVF learners using ETB(λ) to learn the successor features. The top right plot is with GVF learners using the proposed interest method applied to TB(λ) for learning the successor features. The bottom left plot is for an agent using TB(λ) for learning successor features.

algorithm.

In this experiment, we ran an agent using the fixed behaviour policy and compared the effects of experience replay with ADAM against the baseline agents using no experience replay and Auto optimizer. The buffer was set to store the last 10000 experiences and the samples for replay updates were randomly selected from the buffer. The number of replay updates per step was swept from $[2^0, ..., 2^4]$. For the TB(λ) learners, the agent used the ADAM optimizer to learn the value function. The SF-NR learners used the ADAM optimizer to learn the successor features. We swept ADAM's metastep size over the range of 2^{-2n}, with n ranging from [-10,...,0].

Figure 8.5 shows the TE of the algorithms using replay and overlays the best performing online learning algorithms using Auto. The difference in per-

Algorithm 12 Successor Features for Nonstationary Rewards (SF-NR) with Replay

Input: $(S_t, A_t, S_{t+1}, C_{t+1}, \gamma_{t+1}), \pi, w_\psi, w_c$

$buffer \leftarrow S_t, A_t, S_{t+1}, C_{t+1}, \gamma_{t+1}$
$\phi \leftarrow \phi(S_t, A_t, S_{t+1})$
$\hat{\boldsymbol{\psi}} \leftarrow \hat{\boldsymbol{\psi}}(S_t, A_t; w_\psi)$
$\hat{\boldsymbol{\psi}}' \leftarrow \sum_{a'} \pi(a'|S_{t+1}) \hat{\boldsymbol{\psi}}(S_{t+1}, a'; w_\psi)$
$\Delta \leftarrow 0$
for $m = 1$ **to** d **do**
  $\delta_m \leftarrow \phi_m + \gamma_{t+1} \hat{\boldsymbol{\psi}}'_m - \hat{\boldsymbol{\psi}}_m$
  $\Delta \leftarrow \Delta + \delta_m \nabla \hat{\boldsymbol{\psi}}_m$
$w_\psi \leftarrow w_\psi + \alpha \Delta$
$w_c \leftarrow w_c + \alpha (C_{t+1} - \langle \phi, w_c \rangle) \phi$
// Update predictions with new data
for $n = 1$ **to** batch size **do**
  $S, A, S', C \leftarrow$ buffer
  $\phi \leftarrow \phi(S, A, S')$
  $\hat{\boldsymbol{\psi}} \leftarrow \hat{\boldsymbol{\psi}}(S, A; w_\psi)$
  $\hat{\boldsymbol{\psi}}' \leftarrow \sum_{a'} \pi(a'|S') \hat{\boldsymbol{\psi}}(S', a'; w_\psi)$
  $\Delta \leftarrow 0$
  for $m = 1$ **to** d **do**
    $\delta_m \leftarrow \phi_m + \gamma_{t+1} \hat{\boldsymbol{\psi}}'_m - \hat{\boldsymbol{\psi}}_m$
    $\Delta \leftarrow \Delta + \delta_m \nabla \hat{\boldsymbol{\psi}}_m$
  $w_\psi \leftarrow w_\psi + \alpha \Delta$

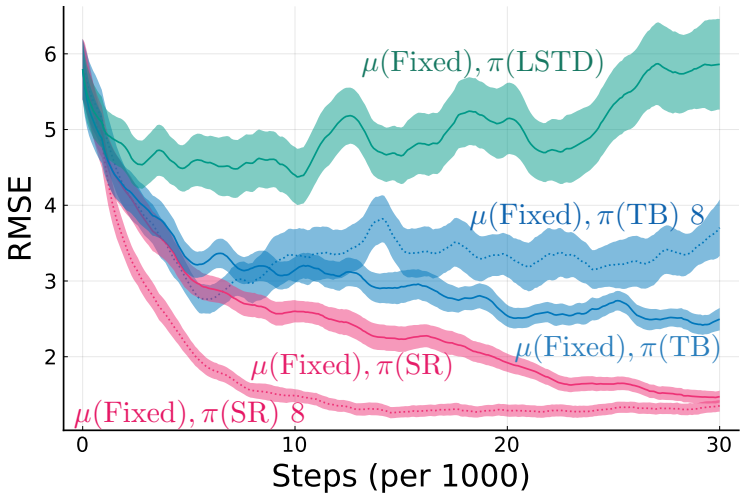


Figure 8.5: Experienced Replay with dotted lines being the replay version of the respective algorithm. Averaged over 30 runs with shaded region being the standard error. The number following the algorithm is the number of replay updates used.

formance between TB(λ) and its replay counterpart visually appears to increase the further into the run. With SF-NR learners, the gap in performance widens and then shrinks.

Experience replay harms the performance of the TB(λ) learners. This is likely due to the poor performance on the drifter GVF since the buffer contains out of date estimates for the drifter. This would explain why the difference in performance is initially smaller and grows as the buffer contains out of date samples. In contrast, the SF-NR GVF learners significantly improve their early learning by applying replay and ADAM to their learning of successor features. These results demonstrate how replay can be effectively used for successor features in a non-stationary setting. The performance gap between SF-NR learners and their replay counterparts fades away by the end of the run. This is likely because by the end of the run, the online SF-NR learners have adequately learned their successor features such that replay does not offer any more significant learning advantages.

8.3 Summary

In this chapter, we highlight two extensions necessary for moving to more complex environments. In Section 8.1, we highlight that GVF learners should be able to focus on learning a subset of states. We compare ETB(λ) and a proposed Interest TB(λ) against TB(λ) (a method that does not incorporate interest) on the Open 2D World environment. In the supporting experiment, we show that rescaling the updates is a promising method for enabling GVFs to specialize their learning in parts of the MDP. In Section 8.2, we demonstrate how replay can be effectively applied in a non-stationary setting for successor features and the limitations of applying it to TB(λ) learners.

Chapter 9

Conclusions and Future Work

In this work, we study what kinds of learners are necessary for multi-prediction learning and how the agent can adapt its behaviour to improve learning across the learners. We highlight the non-stationarity faced by both auxiliary task learners and the behaviour learner and identify how successor features are well suited for this problem domain. We provide empirical evidence of the advantage of successor features over other learning methods (TB(λ) and LSTB(λ)) for GVF learning and show how the same principles can be applied for control. We demonstrate that complex behaviours can be learned as a byproduct of the agent trying to maximize learning across its prediction tasks. In moving to larger MDPs, it is likely that each auxiliary task may not need to be learned accurately across the entire MDP. We propose an interest reweighting method and demonstrate empirically that it can help the agent focus its attention to specific sub-areas of the MDP. Finally, we highlight the issues of applying experience replay in a non-stationary setting and propose an alternative method of using experience replay with successor features that improves learning.

There are many possible directions for future work in this area. The behaviour policy is an ϵ-greedy policy atop action values. This has a limitation as this policy is ϵ deterministic and may not adapt well to situations where a stochastic behaviour policy is needed. It is easy to imagine that as the domains get more complicated, a stochastic behaviour policy will be more desirable. It remains open as to what kind of behaviour learners would be good in this setting and how successor features could be adapted to learn a stochas-

tic behaviour policy. Another area for investigation is the incorporation of planning. The extension with experience replay is a preliminary step for this direction as experience replay can be seen as 1-step planning. It is an exciting direction as the learning of a planner could be included as the set of auxiliary tasks and it would be interesting to investigate these effects. Investigation into making learners more introspective through optimizers or algorithms is another research avenue. More introspective learners enhance the efficacy of the multi-prediction system as a whole by providing more meaningful intrinsic rewards. The scaling of this approach to more complex environments is sure to reveal more research questions and is essential for being useful in modern state of the art systems. Lastly, in our multi-prediction learning problems, we only considered policy evaluation. It is unclear what are the effects on the learning system if the GVF policies are allowed to be learned.

This thesis presents an early step towards enabling agents to effectively learn about the world they find themselves in. The hidden journey behind this work involved successive failures using classic algorithms in RL, like replay, and discovering the utility in newer ideas. This work underscores the importance of considering the complete reinforcement learning system where the interacting components necessitate combining ideas across the field to create an effective agent.

References

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., hunt, J., Mourad, S., Silver, D., & Precup, D. (2019). The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D., Zidek, A., & Munos, R. (2018). Transfer in deep reinforcement learning using successor features and generalised policy improvement. *International Conference on Machine Learning*.

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*.

Barreto, A., Hou, S., Borsa, D., Silver, D., & Precup, D. (2020). Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, *117*(48), 30079–30087.

Bradtke, S., & Barto, A. (1996). Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., & Efros, A. A. (2019). Large-scale study of curiosity-driven learning. *International Conference on Learning Representations*.

Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2019). Exploration by random network distillation. *International Conference on Learning Representations*.

Chentanez, N., Barto, A., & Singh, S. (2005). Intrinsically motivated reinforcement learning. *Advances in Neural Information Processing Systems*.

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

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. *AAAI Conference on Artificial Intelligence*.

Jacobsen, A., Schlegel, M., Linke, C., Degris, T., White, A., & White, M. (2019). Meta-descent for online, continual prediction. *AAAI Conference on Artificial Intelligence*.

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

Kartal, B., Hernandez-Leal, P., & Taylor, M. E. (2019). Terminal prediction as an auxiliary task for deep reinforcement learning. *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Kim, K., Sano, M., De Freitas, J., Haber, N., & Yamins, D. (2020). Active world model learning with progress curiosity. *International Conference on Machine Learning*.

Kolter, J. Z. (2011). The fixed points of off-policy td. *Advances in Neural Information Processing Systems*.

Lin, L. (2004). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, *8*, 293–321.

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.

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*.

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

Mark, R. (2016). Unpublished work [Unpublished manuscript].

Martha, W. (2017). Unifying task specification in reinforcement learning. *International Conference on Machine Learning*.

McLeod, M., Lo, C., White, M., Schlegel, M., Jacobsen, A., Kumaraswamy, R., & White, A. (2021). Continual auxiliary task learning [Submitted to NeurIPS 2022].

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., & Hadsell, R. (2017). Learning to navigate in complex environments. *International Conference on Learning Representations*.

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

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. *International Conference on Machine Learning*.

Patterson, A., White, A., Ghiassian, S., & White, M. (2021). A generalized projected bellman error for off-policy value estimation in reinforcement learning. *arXiv preprint arXiv:2104.13844*.

Precup, D. (2000). Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, 80.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., & Springenberg, J. T. (2018). Learning by playing solving sparse reward tasks from scratch. *International Conference on Machine Learning*.

Sajed, T., Chung, W., & White, M. (2018). High-confidence error estimates for learned value functions. *Uncertainty in Artificial Intelligence*.

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

Schaul, T., & Ring, M. (2013). Better generalization with forecasts. *International Joint Conference on Artificial Intelligence*.

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.

Schmidhuber, J. (2008). Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Workshop on Anticipatory Behavior in Adaptive Learning Systems*, 48–76.

Stadie, B. C., Levine, S., & Abbeel, P. (2015). Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. A Bradford Book.

Sutton, R. S., Mahmood, A. R., & White, M. (2016). An emphatic approach to the problem of off-policy temporal-difference learning. *The Journal of Machine Learning Research*, *17*(1), 2603–2631.

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 Conference on Autonomous Agents and Multiagent Systems*.

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