The Thesis Committee for Brahma Suneil Pavse
certifies that this is the approved version of the following thesis:

Reducing Sampling Error in Batch Temporal Difference Learning

Committee:

Peter Stone, Supervisor

Scott Niekum

Reducing Sampling Error in Batch Temporal Difference Learning

by

Brahma Suneil Pavse

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computer Science

The University of Texas at Austin

May 2020

To my parents, Ashwini and Suneil Pavse

Acknowledgments

I would like to give my sincerest thanks to my advisor, Peter Stone. Over the last two years, Peter has been extremely supportive and has given me the freedom to explore a variety of ideas in reinforcement learning. I am particularly thankful for his immense patience, valuable feedback during our meetings, and for his assistance in my professional life. Peter has always made himself available whenever I needed help, and for that I am extremely grateful. In addition to robot soccer, it has been tremendously fun to play real-world soccer with Peter.

I would also like to thank my second reader, Scott Niekum. He provided valuable advice as the second reader of my undergraduate honors thesis, when I took his reinforcement learning class, and during the development of this thesis. I am also thankful for his help and advice about graduate school.

I am thankful to Kristen Grauman for her wisdom when answering many of my questions related to graduate school. She continues to be extremely patient when answering my non-sensical questions about the research world.

I am also very thankful to Michael Scott and Alison Norman for being extremely encouraging and nurturing figures during my time at UT.

I must also thank my co-authors on this work. In alphabetical order, Ishan Durugkar has been a source of unwavering support, for which I am grateful. I deeply appreciate every time he would put down whatever he was working on to entertain my questions and troubles. Thanks, Ishan. Alongside Peter, Josiah Hanna has practically served as my co-advisor. Thank you, Josiah for helping me select this important problem to explore, debug my experiments, improve my writing, organize my research thoughts, navigate the research literature, and for your kindness and patience whenever I was confused or hit a wall.

I am also grateful to my co-authors on previous research projects as well. In alphabetical

order, Patrick MacAlpine first got me into research two years ago. Thanks, Patrick for all the help when we collaborated on projects and even for your help when I worked on my undergraduate thesis. Faraz Torabi has been a great friend and fun collaborator. I am especially grateful for Faraz's never-ending readiness to listen to my roadblocks; it always gave me a sense of calm. I hope that our work together contributes well to your thesis. Finally, thanks to Garrett Warnell. Thank you, Garrett for your enthusiasm of our work and for teaching me how to be a better writer. I learned a great deal from you about selling a research paper.

I have benefited immensely from the Learning Agents Research Group (LARG) during my time at UT. I have also had the pleasure of making great friends. In alphabetical order, Alessandro Allievi, Sid Desai, Haresh Karnan, Sai Kiran, Brad Knox, Elad Liebman, Bo Liu, Sanmit Narvekar, Harel Yedidsion, Qiping Zhang, and Yifeng Zhu.

Thanks very much to Katie Dahm and Elizabeth Saras for being wonderful and helpful advisors. Thanks to Amy Bush for infrastructure support when running the experiments in this thesis.

Finally, much thanks to my friends. Prabhat Nagarajan gave me guidance on the structuring of this document and has always given me helpful advice on my research career. Darshan Thaker helped me debug my experiments and with parts of the theoretical analysis. Neha Srikanth has been a great source of encouragement and has taken joy in many of the milestones of this thesis. Lastly, Madhav Narayan has supported and encouraged me in every way he can whenever I have tried something ambitious.

Brahma Suneil Pavse

The University of Texas at Austin

May 2020

Reducing Sampling Error in Batch Temporal Difference Learning

Brahma Suneil Pavse, M.S.
The University of Texas at Austin, 2020

Supervisor: Peter Stone

Temporal difference (TD) learning is one of the main foundations of modern reinforcement learning. This thesis studies the use of TD(0), a canonical TD algorithm, to estimate the value function of a given *evaluation* policy from a batch of data. In this batch setting, we show that TD(0) may converge to an inaccurate value function because the update following an action is weighted according to the number of times that action occurred in the batch – not the true probability of the action under the evaluation policy. To address this limitation, we introduce *policy sampling error corrected*-TD(0) (PSEC-TD(0)). PSEC-TD(0) first estimates the empirical distribution of actions in each state in the batch and then uses importance sampling to correct for the mismatch between the empirical weighting and the correct weighting for updates following each action. We refine the concept of a certainty-equivalence estimate and argue that PSEC-TD(0) converges to a more desirable fixed-point than TD(0) for a fixed batch of data. Finally, we conduct a thorough empirical evaluation of PSEC-TD(0) on three batch value function learning tasks in a variety of settings and show that PSEC-TD(0) produces value function estimates with lower mean squared error than the standard TD(0) algorithm.

Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) (Sutton and Barto, 2018) algorithms have been applied to a variety of sequential-decision making problems such as robot manipulation (Kober et al., 2013; Gu et al., 2016) and autonomous driving (Sallab et al., 2017). Many RL algorithms learn an optimal control policy by relying on the *value function*, a function that estimates the expected return from each state when following a particular policy (Puterman and Shin, 1978; Bertsekas, 1987; Konda and Tsitsiklis, 2000). These algorithms require accurate value function estimation with finite data.

One of the most fundamental approaches to value function learning is the temporal difference (TD) algorithm (Sutton, 1988). In this work, we focus on improving the accuracy of the value function learned by *batch* TD. Batch TD computes temporal difference updates for a value function from a given batch of state-action-reward-next-state transitions.

In this thesis, we show that batch TD(0) may converge to an inaccurate value function because it ignores the known action probabilities of the evaluation policy. Consider a single state in which the evaluation policy selects between action a_1 or a_2 with probability 0.5. If, in the finite batch of observed data, a_1 actually happens to occur twice as often as a_2 then TD updates following a_1 will receive twice as much weight as updates following a_2, even though in expectation they should receive the same weight. We describe this finite-sample error in the value function estimate as *policy sampling error*.

To correct for policy sampling error we propose to first estimate the maximum likelihood policy from the observed data and then use importance sampling (Precup et al., 2000a) to

account for the mismatch between the frequency of sampled actions and their true probability under the evaluation policy. Variants of this technique have been successful in multi-armed bandits (Li et al., 2015; Narita et al., 2018; Xie et al., 2018), policy evaluation (Hanna et al., 2019), and policy gradient learning (Hanna and Stone, 2019). However, we are the first to study this technique for value function estimation. We call our new value function learning algorithm batch *policy sampling error corrected-TD(0) (PSEC-TD(0))*.

1.1 Research Question and Contributions

This thesis aims to answer the following question:

Given a fixed batch of data, generated by some policy and transition dynamics distribution, does TD learning compute an accurate value function?

In this work, we provide a theoretical and empirical analysis showing that single step temporal difference learning does *not* compute an accurate value function from a fixed batch of data. In doing so, we make the following contributions:

1. Show that the certainty-equivalence estimate, the fixed point that batch TD(0) converges to, is inaccurate with respect to the true value function.
2. Introduce the PSEC-TD(0) algorithm that reduces the policy sampling error in batch TD(0).
3. Refine the concept of a certainty-equivalence estimate for TD-learning (Sutton, 1988) and provide theoretical justification that PSEC-TD(0) converges to a more desirable fixed-point than TD(0).
4. Empirically show that the PSEC correction is applicable to a TD(0) variant, least squares TD(0) (LSTD) (Bradtke and Barto, 1996).
5. Empirically analyze PSEC-TD(0) in the tabular and function approximation setting.

1.2 Thesis Outline

The thesis is organized as follows: Chapter 2 goes over background and notation used throughout this document, and the related literature, Chapter 3 extends the proof of Sutton

(1988) and shows that batch linear TD(0) converges to the certainty-equivalence estimate in the per-step reward and discounted settings in a Markov reward process (MRP) and Markov decision process (MDP). Chapter 4 proves that PSEC-TD(0) converges to the new and more accurate fixed point, and empirically analyzes PSEC in the tabular and function approximation settings. Finally, Chapter 5 discusses future work and concludes.

Chapter 2

Background

This chapter introduces notation and terminology, formally specifies the batch value function learning problem, and discusses related work.

2.1 Notation and Definitions

Following the standard MDPNv1 notation (Thomas, 2015), we consider a Markov decision process (MDP) with state space, $\mathcal{S}$, action space $\mathcal{A}$, reward function, R, transition dynamics function, P, and discount factor γ (Puterman, 2014). In any state, s, an agent selects stochastic actions according to a policy π, $A \sim \pi(\cdot|s)$. After taking an action, a, in state s the agent transitions to a new state $s' \sim P(\cdot|s,a)$ and receives reward $R(s,a,s')$. We assume $\mathcal{S}$ and $\mathcal{A}$ to be finite; however, our experiments also consider infinite sized $\mathcal{S}$. We consider the episodic, discounted, and finite horizon setting. The policy and MDP jointly induce a *Markov reward process* (MRP), in which the agent transitions between states s and s' with probability $P(s'|s)$ and receives reward $R(s,s')$. Finally, $\boldsymbol{x}(s) : \mathcal{S} \to \mathbb{R}^d$ gives a column feature vector for each state $s \in \mathcal{S}$.

We are concerned with computing the value function, $v^\pi : \mathcal{S} \to \mathbb{R}$, that gives the *value* of any state. The value of a particular state is the expected sum of discounted rewards when following policy π from that state:

$$v^\pi(s) \coloneqq \mathbf{E}_\pi \left[\sum_{k=0}^{L} \gamma^k R_{t+k+1} \;\middle|\; S_t = s \right], \forall s \in \mathcal{S} \tag{2.1}$$

where L is the terminal time-step and the expectation is taken over the distribution of future states, actions, and rewards under π and P.

2.1.1 Matrix Notation for Proofs

In this section, we also introduce matrix-related notation, which is based on the above notation, specifically for the proofs in Chapters 3, 4, and Appendix A.1. Part of these notations are derived from Sutton (1988).

We refer to state features using vectors indexed by the state. So features $\boldsymbol{x}(i)$ for state i is referred to as $\boldsymbol{x}_i$. Reward $r(j|i,a)$ (the reward for transitioning to state j from state i after taking action a) is referred to by r_{ij}^a. The policy $\pi(a|i)$ is π_i^a and the transition function $P(j|i,a)$ is accordingly the value p_{ij}^a. $\mathcal{N}$ and $\mathcal{T}$ are the set of non-terminal and terminal states respectively, and $\mathcal{A}$ is the set of possible actions. $\mathcal{D}$ is the batch of data used to train the value function. For any transition from a non-terminal state to a (non-) terminal state, $i \to j$, $1 \leq i \leq |\mathcal{N}|$ and $1 \leq j \leq |\mathcal{N} \cup \mathcal{T}|$. Finally, the maximum-likelihood estimate (MLE) of the above quantities according to $\mathcal{D}$, is given with a hat ($\hat{}$) on top of the quantity. Further notations that are used in the proofs are explained when introduced.

Notation	Description	Dimension
$\mathcal{S}$	set of states	$\lvert\mathcal{S}\rvert$
$\widehat{\mathcal{S}}$	set of states that appear in batch $\mathcal{D}$	$\lvert\widehat{\mathcal{S}}\rvert$
$\mathcal{A}$	set of actions	$\lvert\mathcal{A}\rvert$
$\widehat{\mathcal{A}}_i$	set of actions that appear in batch $\mathcal{D}$ when agent is in state i	$\lvert\widehat{\mathcal{A}}_i\rvert$
$\mathcal{N} \subset \mathcal{S}$	non-terminal states	$\lvert\mathcal{N}\rvert$
$\widehat{\mathcal{N}} \subset \widehat{\mathcal{S}}$	non-terminal states that appear in batch $\mathcal{D}$	$\lvert\widehat{\mathcal{N}}\rvert$
$\mathcal{T} \subset \mathcal{S}$	terminal states	$\lvert\mathcal{T}\rvert$
$\widehat{\mathcal{T}} \subset \widehat{\mathcal{S}}$	terminal states that appear in batch $\mathcal{D}$	$\lvert\widehat{\mathcal{T}}\rvert$
I	identity matrix	$\lvert\mathcal{N}\rvert \times \lvert\mathcal{N}\rvert$
p^{π}_{jk}	probability of transitioning from state j to state k under a policy, π	-
p^{a}_{jk}	probability of transitioning from state j to state k after taking action a	-
$\bar{r}_j$	mean reward when transitioning from state j	-
$\bar{r}^{a}_{ij}$	mean reward when transitioning from state i to state j after taking action a	-
Q	$Q_{jk} \coloneqq p^{\pi}_{jk}$	$\lvert\mathcal{N}\rvert \times \lvert\mathcal{N}\rvert$
$[\mathbf{m}]_i$	expected reward on transitioning from state i to non-terminal state j i.e. $\sum_{j\in\mathcal{N}} p^{\pi}_{ij} r_{ij}$ or $\sum_{j\in\mathcal{N}} \sum_{a\in\mathcal{A}} \pi^{a}_{i} p^{a}_{ij} r^{a}_{ij}$	$\lvert\mathcal{N}\rvert$
$[\mathbf{h}]_i$	expected reward on transitioning from state i to non-terminal state j to terminal state i.e. $\sum_{j\in\mathcal{T}} p^{\pi}_{ij} r_{ij}$ or $\sum_{j\in\mathcal{T}} \sum_{a\in\mathcal{A}} \pi^{a}_{i} p^{a}_{ij} r^{a}_{ij}$	$\lvert\mathcal{N}\rvert$
$d_{\pi_e}(i)$	$\forall i \in \mathcal{S}$ weighted proportion of time spent in state i under policy π_e	-

2.2 Batch Value Prediction

This thesis investigates the problem of approximating, v^{π_e}, given a batch of data, $\mathcal{D}$ and an evaluation policy π_e. Let a single episode, τ, be defined as $\tau := (S_0, A_0, R_0, S_1, ..., S_{L_{\tau-1}}, A_{L_{\tau-1}}, R_{L_{\tau-1}})$, where L_τ is the length of the episode τ. The batch of data, $\mathcal{D}$, consists of m episodes, i.e., $\mathcal{D} := \{\tau_i\}_{i=0}^{m-1}$. The policy that generated the batch of data is called the *behavior policy*, π_b. If π_b is the same as π_e for all episodes then learning is said to be done

on-policy; otherwise it is *off-policy*.

In batch value prediction, a value function learning algorithm uses a fixed batch of data to learn an estimate, $\hat{v}^{\pi_e}$, that approximates the true value function v^{π_e}. In this work, we focus on the linear approximation of v^{π_e}:

$$\hat{v}^{\pi_e}(s) := \boldsymbol{w}^T \boldsymbol{x}(s)$$

where we seek to find a weight vector, $\boldsymbol{w}$, such that $\boldsymbol{w}^T \boldsymbol{x}(s)$ approximates the true value, $v^{\pi_e}(s)$. The error of the predicted value function, $\hat{v}^{\pi_e}$, with respect to the true value function, v^{π_e}, is measured by calculating the mean squared value error between $v^{\pi_e}(s)$ and $\hat{v}^{\pi_e}(s)$ $\forall s \in \mathcal{S}$ weighted by the proportion of time spent in each state under policy π_e, $d_{\pi_e}(s)$. Thus, we seek to find a weight vector $\boldsymbol{w}$ that minimizes:

$$\text{MSVE}(\boldsymbol{w}) := \sum_{s \in \mathcal{S}} d_{\pi_e}(s) \Big(v^{\pi_e}(s) - \boldsymbol{w}^T \boldsymbol{x}(s) \Big)^2 \tag{2.2}$$

2.3 Batch Linear TD(0)

A fundamental algorithm for value prediction is the single-step temporal difference learning algorithm, TD(0) (Sutton, 1988). Algorithm 1 gives pseudo-code for the batch linear TD(0) algorithm described by Sutton (1988).

Algorithm 1 Batch Linear TD(0) to estimate v^{π_e}

Input: policy to evaluate π_e, behavior policy π_b, batch $\mathcal{D}$, linear value function, $\hat{v} : \mathcal{S} \times \mathbb{R}^d \to \mathbb{R}$, step-size $\alpha > 0$, convergence threshold $\epsilon > 0$
Initialize: weight vector $\boldsymbol{w}$ arbitrarily (e.g.: $\boldsymbol{w} := \mathbf{0}$)
Initialize: update aggregation vector $\boldsymbol{u} := \mathbf{0}$
while $\Delta \boldsymbol{w} \geq \epsilon$ **do**
  for each episode, $\tau \in \mathcal{D}$ **do**
    for each transition, $(S, A, R, S') \in \tau$ **do**
      $\hat{y} \leftarrow R + \gamma \boldsymbol{w}^T \boldsymbol{x}(S')$
      $\rho \leftarrow \frac{\pi_e(A|S)}{\pi_b(A|S)}$ {for on-policy, $\pi_b = \pi_e$}
      $\boldsymbol{u} \leftarrow \boldsymbol{u} + \left[\rho \hat{y} - \boldsymbol{w}^T \boldsymbol{x}(S)\right] \boldsymbol{x}(S)$
    end for
  end for
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha \boldsymbol{u}$ {batch update}
  $\boldsymbol{u} \leftarrow \mathbf{0}$ {clear aggregation}
end while

Sutton (1988) proved that batch linear TD(0) converges to a fixed point in the on-policy case i.e. when $\pi_e = \pi_b$. An off-policy batch TD(0) algorithm uses importance sampling ratios to ensure that the expected update is the same as it would be if actions were taken with π_e instead of π_b (Precup et al., 2000a). Unlike on-policy TD(0), off-policy TD(0) is *not* guaranteed to converge (Baird, 1995).

2.4 Related Work

In this section, we discuss the related literature to our work.

2.4.1 Estimating the Behavior Policy

The idea of estimating the behavior policy and applying importance sampling has been motivated by prior work. Li et al. (2015) estimates the behavior policy in their REG estimator for off-policy evaluation to achieve a lower mean squared error than when using the true

behavior policy. Narita et al. (2018) show that estimating the behavior policy achieves lower variance than that of other estimators when estimating expected reward. Xie et al. (2018) introduce an approach called Maximum Likelihood Inverse Propensity Scoring (MLIPS) that uses a maximum likelihood estimate of the policy from a batch of data in the inverse propensity weights instead of the true policy to achieve an unbiased and lower mean squared error estimator. Hirano et al. (2003) and Rosenbaum (1987) show that propensity scoring techniques work better when using the estimated behavior policy over the true behavior policy. Henmi et al. (2007) and Delyon and Portier (2016) show that using the maximum likelihood of the sampling distribution improved the asymptotic variance in numerical integration tasks than when using the true sampling distribution. Our work is distinct from all these works in that they focus on the multi-armed bandit and causal inference settings, and we focus on the full Markov decision process (MDP).

There has also been work that uses importance sampling with an estimated behavior policy in the MDP setting. Hanna et al. (2019) introduce a family of methods called regression importance sampling methods (RIS) and show that they have lower variance than importance sampling with the true behavior policy when doing policy evaluation. Hanna and Stone (2019) also show that a similar technique led to more sample-efficient policy gradient learning. On the other hand, there has also been work by Farajtabar et al. (2018) who show that estimating the behavior policy may result in higher variance in importance sampling in MDPs when the estimation of the behavior policy is done using data *different* from the data used for off-policy evaluation. In contrast, our work differs in the following ways: 1) it focuses on *value function learning*, where the focus is on learning the expected return at every state visited by the agent instead of across a set of actions (multi-armed bandit) or for some start states that are a subset of all the states the agent visits, 2) it estimates the behavior policy on the *same* data used to learn the value function, and 3) it is focused on the on-policy setting, but is also applicable in the off-policy setting.

2.4.2 Reducing Sampling Error

PSEC-TD(0) corrects policy sampling error through importance sampling with an estimated behavior policy. Other works avoid policy sampling error entirely by computing analytic

expectations. Expected SARSA (van Seijen et al., 2009), learns action-values by analytically computing the expected return of the next state during bootstrapping as opposed to using the value of the sampled next action. The Tree-backup algorithm (Precup et al., 2000b) extends Expected SARSA to a multi-step algorithm. $Q(\sigma)$ (Asis et al., 2017) unifies SARSA (Sutton, 1996; Rummery and Niranjan, 1994), Expected SARSA, and Tree-backups, to find a balance between sampling and analytic expectation computation ($\sigma = 0$ is the Tree-backup algorithm and $\sigma = 1$ is SARSA). Similarly, there have been more advanced methods that combine $Q(\sigma)$ with eligibility traces to reduce sampling error (Yang et al., 2018). All these approaches rely on computing the analytic expectation over the actions on the state that they are bootstrapping on. Our work is distinct from these in that we do not require an analytic expectation to be computed and we focus on learning state values which may be preferable for prediction as well as for a variety of actor-critic approaches (Konda and Tsitsiklis, 2000; Mnih et al., 2016). To the best of our knowledge, no other approach exists for correcting policy sampling error when learning state values.

2.5 Summary

In this chapter, we established the notation and definitions used throughout this document, described batch value function learning, detailed a fundamental batch value function learning algorithm, batch linear TD(0), and surveyed the relevant literature.

Chapter 3

Convergence of Batch Linear TD(0)

In this chapter, we discuss the convergence of batch linear TD(0) to a fixed-point, the *certainty equivalence estimate* for the underlying Markov Reward Process (MRP). We refine this concept to better reflect our objective of evaluating a policy in an MDP and then prove that batch TD(0) converges to an equivalent fixed point that ignores knowledge of the known evaluation policy, π_e, leading to inaccuracy in the value function estimate. This result motivates our proposed solution algorithm, which we discuss in Chapter 4.

3.1 Additional Notational Setup

Before presenting the definitions and theory, we introduce additional notation and assumptions. In this chapter, we assume that we are in the on-policy setting ($\pi_b = \pi_e$). Let $\widehat{\mathcal{S}}$ be the set of states and $\widehat{\mathcal{A}}$ be the set of actions that appear in $\mathcal{D}$ and let $\bar{R}(s_j)$ be the mean reward received when transitioning from state s_j in the batch $\mathcal{D}$. Finally, if the notation includes a hat ($\hat{}$), it is the maximum-likelihood estimate (MLE) according to $\mathcal{D}$. For example, $\hat{\pi}$ is the MLE of π_b.

3.2 Convergence to the MRP Certainty Equivalence Estimate

Sutton (1988) proved that batch linear TD(0) converges to the *certainty equivalent estimate*. That is, it converges to the exact value function of the maximum likelihood MRP according to the observed batch. This exact value function can be calculated using dynamic programming (Bellman, 2003; Bertsekas, 1987) with the MLE MRP. We call this value function estimate

the *Markov reward process certainty equivalence estimate* (MRP-CEE).

Definition 1. *Markov Reward Process Certainty Equivalence Estimate (MRP-CEE) Value Function*

The MRP-CEE is the value function $\hat{v}_{\texttt{MRP}}$ *that,* $\forall s_j, s_k \in \widehat{\mathcal{S}}$*, satisfies:*

$$\hat{v}_{\texttt{MRP}}(s_j) = \bar{R}(s_j) + \gamma \sum_{k \in \widehat{\mathcal{S}}} \widehat{P}(s_k|s_j)\hat{v}_{\texttt{MRP}}(s_k). \tag{3.1}$$

Having now defined the MRP-CEE value function, we prove that batch TD(0) converges to the MRP-CEE value function. This fact was first proven by Sutton (1988) (see Theorem 3 of Sutton (1988)), however the original proof only considers rewards upon termination and no discounting. The extension to rewards per-step and discounting is straightforward, but to the best of our knowledge has not appeared in the literature before. Following the proof by Sutton (1988), we prove the extension here as a first step before extending the proof to an MDP, where the inefficiency of TD(0) becomes clear.

Note that in Theorem 1, we show that batch linear TD(0) converges to an equivalent form of Equation (3.1) in matrix notation (see, Section 2.1.1 for details on this notation):

$$\hat{v}(i) = \left[(I - \gamma\hat{Q})^{-1}(\hat{\mathbf{m}} + \hat{\mathbf{h}})\right]_i \tag{3.2}$$

In Appendix A.1.2, we show that Equation (3.1) and Equation (3.2) are equivalent.

Theorem 1 (Batch Linear TD(0) Convergence)**.** *For any batch whose observation vectors* $\{\boldsymbol{x}(s)|s \in \widehat{\mathcal{S}}\}$ *are linearly independent, there exists an* $\epsilon > 0$ *such that, for all positive* $\alpha < \epsilon$ *and for any initial weight vector, the predictions for linear TD(0) converge under repeated presentations of the batch with weight updates after each complete presentation to the fixed-point (3.2).*

Proof. Batch linear TD(0) makes an update to the weight vector, $\boldsymbol{w}_n$ (of dimension, length of the feature vector), after each presentation of the batch:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{L_\tau} \alpha \left[(\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1}) - \boldsymbol{w}_n^T \boldsymbol{x}_t\right] \boldsymbol{x}_t$$

where $\mathcal{D}$ is the batch of episodes, L_τ is the length of each episode τ, and α is the learning rate.

We can re-write the whole presentation of the batch of data in terms of the number of times there was a transition from state i to state j in the batch i.e. $\hat{c}_{ij} = \hat{d}_i \hat{p}_{ij}$, where $\hat{d}_i$ is the number of times state $i \in \widehat{\mathcal{N}}$ appears in the batch.

$$
\begin{aligned}
\boldsymbol{w}_{n+1} &= \boldsymbol{w}_n + \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{L_\tau} \alpha \left[(\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1} - \boldsymbol{w}_n^T \boldsymbol{x}_t) \right] \boldsymbol{x}_t \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{c}_{ij} \left[(\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j - \boldsymbol{w}_n^T \boldsymbol{x}_i) \right] \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{d}_i \hat{p}_{ij} \left[(\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j - \boldsymbol{w}_n^T \boldsymbol{x}_i) \right] \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{d}_i \hat{p}_{ij} (\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) \boldsymbol{x}_i - \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{d}_i \hat{p}_{ij} (\boldsymbol{w}_n^T \boldsymbol{x}_i) \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\widehat{\mathcal{N}}}} \hat{d}_i \boldsymbol{x}_i \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{p}_{ij} (\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i (\boldsymbol{w}_n^T \boldsymbol{x}_i) \boldsymbol{x}_i \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{p}_{ij} \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{p}_{ij} (\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right] && \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \hat{p}_{ij} = 1 \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\left(\sum_{j \in \widehat{\mathcal{N}}} \hat{p}_{ij} (\bar{r}_{ij} + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) \right) + \left(\sum_{j \in \widehat{\mathcal{T}}} \hat{p}_{ij} \bar{r}_{ij} \right) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right] && \text{If } \boldsymbol{x}_j \in \widehat{\mathcal{T}}, \boldsymbol{w}_n^T \boldsymbol{x}_j = 0 \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\left(\sum_{j \in \widehat{\mathcal{N}}} \hat{p}_{ij} \bar{r}_{ij} \right) + \left(\gamma \sum_{j \in \widehat{\mathcal{N}}} \hat{p}_{ij} \boldsymbol{w}_n^T \boldsymbol{x}_j \right) + \left(\sum_{j \in \widehat{\mathcal{T}}} \hat{p}_{ij} \bar{r}_{ij} \right) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right]
\end{aligned}
$$

$$
\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha \widehat{X} \widehat{D} \left[\hat{\mathbf{m}} + \gamma \widehat{Q} \widehat{X}^T \boldsymbol{w}_n + \hat{\mathbf{h}} - \widehat{X}^T \boldsymbol{w}_n \right] \tag{3.3}
$$

where $\widehat{X}$ denotes the matrix (of dimensions, length of the feature vector by $|\widehat{\mathcal{S}}|$) with columns, $\boldsymbol{x}_i \in \widehat{\mathcal{S}}$ and $\widehat{D}$ is a diagonal matrix (of dimensions, $|\widehat{\mathcal{S}}|$ by $|\widehat{\mathcal{S}}|$) with $\widehat{D}_{ii} = \hat{d}_i$. Given the successive updates to the weight vector $\boldsymbol{w}_n$, we now consider the actual values predicted as the following by multiplying $\widehat{X}^T$ on both sides:

$$
\begin{aligned}
\widehat{X}^T \boldsymbol{w}_{n+1} &= \widehat{X}^T \boldsymbol{w}_n + \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} + \gamma \widehat{Q} \widehat{X}^T \boldsymbol{w}_n - \widehat{X}^T \boldsymbol{w}_n \right) \\
&= \widehat{X}^T \boldsymbol{w}_n + \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} \right) + \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\gamma \widehat{Q} \widehat{X}^T \boldsymbol{w}_n - \widehat{X}^T \boldsymbol{w}_n \right) \\
&= \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} \right) + \left(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(I - \gamma \widehat{Q} \right) \right) \widehat{X}^T \boldsymbol{w}_n
\end{aligned}
$$

We then unroll the above equation by recursively applying $\widehat{X}^T \boldsymbol{w}_n$ till $n = 0$.

$$
\begin{aligned}
\widehat{X}^T \boldsymbol{w}_{n+1} &= \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} \right) + \left(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(I - \gamma \widehat{Q} \right) \right) \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} \right) \\
&\quad + \left(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(I - \gamma \widehat{Q} \right) \right)^2 \widehat{X}^T \boldsymbol{w}_{n-1} \\
&\quad \vdots \\
&= \sum_{k=0}^{n-1} \left(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(I - \gamma \widehat{Q} \right) \right)^k \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(\hat{\mathbf{m}} + \hat{\mathbf{h}} \right) \\
&\quad + \left(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} \left(I - \gamma \widehat{Q} \right) \right)^n \widehat{X}^T \boldsymbol{w}_0
\end{aligned} \tag{3.4}
$$

Assuming that as $n \to \infty$, $(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} (I - \gamma \widehat{Q}))^n \to 0$, we can drop the second term and the sequence $\{\widehat{X}^T \boldsymbol{w}_n\}$ converges to:

$$
\begin{aligned}
\lim_{n \to \infty} \widehat{X}^T \boldsymbol{w}_n &= \left(I - (I - \alpha \widehat{X}^T \widehat{X} \widehat{D} (I - \gamma \widehat{Q})) \right)^{-1} (\alpha \widehat{X}^T \widehat{X} \widehat{D} (\hat{\mathbf{m}} + \hat{\mathbf{h}})) \\
&= (I - \gamma \widehat{Q})^{-1} \widehat{D}^{-1} (\widehat{X}^T \widehat{X})^{-1} \alpha^{-1} \alpha \widehat{X}^T \widehat{X} \widehat{D} (\hat{\mathbf{m}} + \hat{\mathbf{h}}) \\
&= (I - \gamma \widehat{Q})^{-1} (\hat{\mathbf{m}} + \hat{\mathbf{h}}) \\
\lim_{n \to \infty} \mathbb{E} \left[\boldsymbol{x}_i^T \boldsymbol{w}_n \right] &= \left[(I - \gamma \widehat{Q})^{-1} (\hat{\mathbf{m}} + \hat{\mathbf{h}}) \right]_i, \forall i \in \widehat{\mathcal{N}}
\end{aligned}
$$

What is left to show now is $n \to \infty$, $(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} (I - \gamma \widehat{Q}))^n \to 0$. Following Sutton (1988), we first show that $\widehat{D}(I - \gamma \widehat{Q})$ is positive definite, and then that $\widehat{X}^T \widehat{X} \widehat{D} (I - \gamma \widehat{Q})$ has a full set of eigenvalues all of whose real parts are positive. This enables us to show that α can be chosen so that eigenvalues of $(I - \alpha \widehat{X}^T \widehat{X} \widehat{D} (I - \gamma \widehat{Q}))$ are less than 1 in modulus, which assures us that its powers converge to 0.

To show that $\widehat{D}(I - \gamma \widehat{Q})$ is positive definite, we refer to the Gershgorin Circle theorem

(Gerschgorin, 1931), which states that if a matrix, A, is real, symmetric, and strictly diagonally dominant with positive diagonal entries, then A is positive definite. However, we cannot apply this theorem as is to $\widehat{D}(I - \gamma\widehat{Q})$ since the matrix is not necessarily symmetric. To use the theorem, we first apply another theorem (Theorem A.3 from Sutton (1988)) that states: a square matrix A is positive definite if and only if $A + A^T$ is positive definite. So it suffices to show that $\widehat{D}(I - \gamma\widehat{Q}) + (\widehat{D}(I - \gamma\widehat{Q}))^T$ is positive definite.

Consider the matrix $S = \widehat{D}(I - \gamma\widehat{Q}) + (\widehat{D}(I - \gamma\widehat{Q}))^T$. We know that S is real and symmetric. It remains to show that the diagonal entries are positive and that S is strictly diagonally dominant. First, we look at the diagonal entries, $S_{ii} = 2[\widehat{D}(I - \gamma\widehat{Q})]_{ii} = 2\hat{d}_i(1 - \gamma\hat{p}_{ii}) > 0, \forall i \in \widehat{\mathcal{N}}$, which are positive. Second, we have the non-diagonal entries for $i \neq j$ as $S_{ij} = [\widehat{D}(I - \gamma\widehat{Q})]_{ij} + [\widehat{D}(I - \gamma\widehat{Q})]_{ji} = -\gamma\hat{d}_i\hat{p}_{ij} - \gamma\hat{d}_j\hat{p}_{ji} \leq 0$, which are nonpositive. We want to show that $|S_{ii}| \geq \sum_{j\neq i} |S_{ij}|$, with strict inequality holding for at least one i; we know that the diagonal elements $S_{ii} > 0$ and non-diagonal elements $S_{ij} \leq 0$, $i \neq j$. Hence, to show that S is strictly diagonally dominant, it is enough to show that $S_{ii} > -\sum_{j\neq i} S_{ij}$, which means we can simply show that the sum of each entire row is greater than 0, i.e. $\sum_j S_{ij} > 0$.

Before we show that $\sum_j S_{ij} > 0$, we note that $\hat{d}^T = \hat{\mu}^T(I - \widehat{Q})^{-1}$ where $\hat{\mu}_i$ is the empirical state distribution of state i. Given the definitions of $\hat{d}$, $\hat{\mu}$, and $\widehat{Q}$, this fact follows from Kemeny et al. (1960) and is used by Sutton (1988). Using this fact, we show that $\sum_j S_{ij} \geq 0$:

$$
\begin{aligned}
\sum_j S_{ij} &= \sum_j \left([\widehat{D}(I - \gamma\widehat{Q})]_{ij} + [\widehat{D}(I - \gamma\widehat{Q})]_{ij}^T \right) \\
&= \hat{d}_i \sum_j ([I - \gamma\widehat{Q}]_{ij} + \sum_j \hat{d}_j [I - \gamma\widehat{Q}]_{ij}^T) \\
&= \hat{d}_i \sum_j (1 - \gamma\hat{p}_{ij}) + \left[\hat{d}^T(I - \widehat{Q}) \right]_i \\
&= \hat{d}_i \sum_j (1 - \gamma p_{ij}) + \left[\hat{\mu}^T(I - \widehat{Q})^{-1}(I - \widehat{Q}) \right]_i \qquad \hat{d}^T = \hat{\mu}^T(I - \widehat{Q})^{-1} \\
&= \hat{d}_i(1 - \gamma \sum_j \hat{p}_{ij}) + \hat{\mu}_i \\
&\geq 0,
\end{aligned}
$$

where the final inequality is strict since $\hat{\mu}$ is positive for at least one element. Given the above, we have shown that S is real, symmetric, and strictly diagonally dominant; hence, S is positive definite according to the Gershgorin Circle theorem (Gerschgorin, 1931). Since, $S = \widehat{D}(I - \gamma\widehat{Q}) + (\widehat{D}(I - \gamma\widehat{Q}))^T$ is positive definite, we have $\widehat{D}(I - \gamma\widehat{Q})$ to be positive definite.

Now we need to show that $\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})$ has a full set of eigenvalues, all of whose real parts are positive. We know that $\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})$ has a full set of eigenvalues for the same reason shown by Sutton (1988), i.e. $\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})$ is a product of three non-singular matrices, which means $\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})$ is nonsingular as well; hence, no eigenvalues are 0 i.e. its set of eigenvalues is full.

Consider λ and y to be an eigevalue and eigenvector pair of $\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})$. First lets consider that y may be a complex number and is of the form $y = a + bi$, and let $z = (\widehat{X}^T\widehat{X})^{-1}y, y \neq 0$. Second, we consider $\widehat{D}(I - \gamma\widehat{Q})$ from earlier i.e. where $*$ is the conjugate-transpose:

$$\begin{aligned}
y^*\widehat{D}(I - \gamma\widehat{Q})y &= z^*\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})y && \text{substituting } y^* \\
&= z^*\lambda y && \widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q})y = \lambda y \\
&= \lambda z^*\widehat{X}^T\widehat{X}z && \text{substituting } y \\
&= \lambda(\widehat{X}z)^*\widehat{X}z \\
(a^T - b^Ti)(\widehat{D}(I - \gamma\widehat{Q}))(a^T + b^Ti) &= \lambda(\widehat{X}z)^*\widehat{X}z && \text{substituting } y^* \text{ and } y
\end{aligned}$$

From the above equality, we know that the real parts (Re) of the LHS and RHS are equal as well i.e.

$$\begin{aligned}
\text{Re}\left(y^*\widehat{D}(I - \gamma\widehat{Q})y\right) &= \text{Re}\left(\lambda(\widehat{X}z)^*\widehat{X}z\right) \\
a^T\widehat{D}(I - \gamma\widehat{Q})a + b^T\widehat{D}(I - \gamma\widehat{Q})b &= (\widehat{X}z)^*\widehat{X}z\text{Re}\left(\lambda\right)
\end{aligned}$$

LHS must be strictly positive since we already proved that $\widehat{D}(I - \gamma\widehat{Q})$ is positive definite and by definition, RHS, $(\widehat{X}z)^*\widehat{X}z$, is strictly positive as well. Thus, the Re(λ) must be positive. Finally, using this result we want to show that the eigenvalues of $(I - \alpha\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q}))$ are

of modulus less than 1 for a suitable α.

First, we can see that y is also an eigenvector of $(I - \alpha\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q}))$, since $(I - \alpha\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q}))y = y - \alpha\lambda y = (1 - \lambda\alpha)y$, where $\lambda' = (1 - \alpha\lambda)$ is an eigenvalue of $(I - \alpha\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{Q}))$. Second, we want to find suitable α such that the modulus of λ' is less than 1. We have the modulus of λ':

$$
\begin{aligned}
\|\lambda'\| &= \|1 - \alpha\lambda\| \\
&= \sqrt{(1 - \alpha a)^2 + (-\alpha b^2)} && \text{substituting } \lambda = a + bi \text{ of general complex form} \\
&= \sqrt{1 - 2\alpha a + \alpha^2 a^2 + \alpha^2 b^2} \\
&= \sqrt{1 - 2\alpha a + \alpha^2 (a^2 + b^2)} \\
&< \sqrt{1 - 2\alpha a + \alpha \frac{2a}{(a^2 + b^2)}(a^2 + b^2)} && \text{using } \alpha = \frac{2a}{(a^2 + b^2)} \\
&= \sqrt{1 - 2\alpha a + 2\alpha a} = 1
\end{aligned}
$$

From above, we can see that if α is chosen such that $0 < \alpha < \frac{2a}{a^2+b^2}$, then λ' will have modulus less than 1. Then using the theorem that states: if a matrix A has n independent eigenvectors with eigenvalues λ_i, then $A^k \to 0$ as $k \to \infty$ if and only if all $\|\lambda_i\| < 1$, which implies that $\lim_{n\to\infty} \left(I - \alpha\widehat{X}\widehat{D}(I - \widehat{Q})\widehat{X}^T\right)^n = 0$, taking the trailing element in Equation (3.4) to 0 for a suitable α. We thus prove convergence to the fixed point in Equation (3.1) if a batch linear TD(0) update is used with an appropriate step size α. □

3.3 Convergence to the MDP Certainty Equivalence Estimate

In RL, the transitions of the induced MRP are a function of both the action probabilities under the behavior policy and the MDP transition dynamics. That is $\forall s, s' \in \widehat{\mathcal{S}}$:

$$\widehat{P}(s'|s) = \sum_{a \in \widehat{\mathcal{A}}} \hat{\pi}(a|s)\widehat{P}(s'|s, a) \tag{3.5}$$

$$\bar{R}(s) = \sum_{a \in \widehat{\mathcal{A}}} \hat{\pi}(a|s)\bar{R}(s, a), \tag{3.6}$$

where $\bar{R}(s,a)$ is the mean reward observed in state s on taking action a. We define a new certainty-equivalence estimate that separates these two factors. We call this new value function estimate the *Markov decision process certainty equivalent estimate* (MDP-CEE).

Definition 2. *Markov Decision Process Certainty Equivalence Estimate (MDP-CEE) Value Function*

The MDP-CEE is the value function, $\hat{v}^{\hat{\pi}}_{\texttt{MDP}}$*, that,* $\forall s_j, s_k \in \widehat{\mathcal{S}}$*, satisfies:*

$$\hat{v}^{\hat{\pi}}_{\texttt{MDP}}(s_j) = \sum_{a \in \widehat{\mathcal{A}}} \hat{\pi}(a|s_j) \left(\bar{R}(s_j, a) + \gamma \sum_{k \in \widehat{\mathcal{S}}} \widehat{P}(s_k|s_j, a) \hat{v}^{\hat{\pi}}_{\texttt{MDP}}(s_k) \right). \tag{3.7}$$

From Definition 2 and Equations (3.5) and (3.6), it is straightforward to verify that MDP-CEE and MRP-CEE are equivalent.

Note that in Theorem 2, similar to Theorem 1, we show that batch linear TD(0) converges to an equivalent form of Equation (3.7) in matrix notation (see, Section 2.1.1 for details on matrix notation):

$$v^{\hat{\pi}}(i) = \left[(I - \gamma \widehat{Q})^{-1} (\hat{\mathbf{m}} + \hat{\mathbf{h}}) \right]_i \tag{3.8}$$

In Appendix A.1.4, we show that Equation (3.7) and Equation (3.8) are equivalent. Theorem 2 gives the convergence of batch TD(0) to the MDP-CEE value function:

Theorem 2 (Batch Linear TD(0) Convergence)**.** *For any batch whose observation vectors* $\{\boldsymbol{x}(s) | s \in \widehat{\mathcal{S}}\}$ *are linearly independent, there exists an* $\epsilon > 0$ *such that, for all positive* $\alpha < \epsilon$ *and for any initial weight vector, the predictions for linear TD(0) converge under repeated presentations of the batch with weight updates after each complete presentation to the fixed-point (3.8).*

Proof. Batch linear TD(0) makes an update to weight vector, $\boldsymbol{w}_n$ (of dimension, length of the feature vector), after each presentation of the batch:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{L_\tau} \alpha \left[(\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1}) - \boldsymbol{w}_n^T \boldsymbol{x}_t \right] \boldsymbol{x}_t$$

where $\mathcal{D}$ is the batch of episodes, L_τ is the length of each episode τ, and α is the learning rate.

We can re-write the whole presentation of the batch of data in terms of the number of times there was a transition from state i to state j when taking action a in the batch i.e. $\hat{c}_{ij}^a = \hat{d}_i \hat{\pi}_i^a \hat{p}_{ij}^a$, where $\hat{d}_i$ is the number of times state $i \in \widehat{\mathcal{N}}$ appears in the batch.

$$
\begin{aligned}
\boldsymbol{w}_{n+1} &= \boldsymbol{w}_n + \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{L_\tau} \alpha \left[(\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1} - \boldsymbol{w}_n^T \boldsymbol{x}_t) \right] \boldsymbol{x}_t \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{c}_{ij}^a \left[(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j - \boldsymbol{w}_n^T \boldsymbol{x}_i) \right] \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{d}_i \hat{\pi}_i^a \hat{p}_{ij}^a \left[(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j - \boldsymbol{w}_n^T \boldsymbol{x}_i) \right] \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{d}_i \hat{p}_{ij}^a \hat{\pi}_i^a (\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) \boldsymbol{x}_i - \alpha \sum_{i \in \widehat{\mathcal{N}}} \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{d}_i \hat{\pi}_i^a \hat{p}_{ij}^a (\boldsymbol{w}_n^T \boldsymbol{x}_i) \boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\widehat{\mathcal{N}}}} \hat{d}_i \boldsymbol{x}_i \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a (\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i (\boldsymbol{w}_n^T \boldsymbol{x}_i) \boldsymbol{x}_i \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{\pi}_i^a \hat{p}_{ij}^a
\end{aligned}
$$

$$
\begin{aligned}
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a (\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right] \qquad \sum_{j \in \widehat{\mathcal{N}} \cup \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{\pi}_i^a \hat{p}_{ij}^a = 1 \\
&= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\left(\sum_{j \in \widehat{\mathcal{N}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a (\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) \right) + \left(\sum_{j \in \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \pi_i^a \bar{r}_{ij}^a \right) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right] \quad \text{If } \boldsymbol{x}_j \in \widehat{\mathcal{T}}, \boldsymbol{w}_n^T \boldsymbol{x}_j = 0
\end{aligned}
$$

$$
= \boldsymbol{w}_n + \alpha \sum_{i \in \widehat{\mathcal{N}}} \hat{d}_i \boldsymbol{x}_i \left[\left(\sum_{j \in \widehat{\mathcal{N}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a \bar{r}_{ij}^a \right) + \left(\gamma \sum_{j \in \widehat{\mathcal{N}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a \boldsymbol{w}_n^T \boldsymbol{x}_j \right) + \left(\sum_{j \in \widehat{\mathcal{T}}} \sum_{a \in \hat{\mathcal{A}}_i} \hat{p}_{ij}^a \hat{\pi}_i^a \bar{r}_{ij}^a \right) - \boldsymbol{w}_n^T \boldsymbol{x}_i \right]
$$

$$
\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha \widehat{X} \widehat{D} \left[\hat{\mathbf{m}} + \gamma \widehat{Q} \widehat{X}^T \boldsymbol{w}_n + \hat{\mathbf{h}} - \widehat{X}^T \boldsymbol{w}_n \right] \tag{3.9}
$$

where $\widehat{X}$ denotes the matrix (of dimensions, length of the feature vector by $|\widehat{\mathcal{S}}|$) with columns, $\boldsymbol{x}_i \in \widehat{\mathcal{S}}$ and $\widehat{D}$ is a diagonal matrix (of dimensions, $|\widehat{\mathcal{S}}|$ by $|\widehat{\mathcal{S}}|$) with $\widehat{D}_{ii} = \hat{d}_i$.

Notice that Equation (3.9) is the same as Equation (3.3) since the considered MRP and MDP settings are equivalent. Due to this similarity, we omit the proof from here below

as it is identical to the Theorem 1 proof.

□

The MDP-CEE value function highlights two sources of estimation error in the value function estimate. Either the observed state transitions occur at a different frequency than the probability given by P i.e. $P \neq \widehat{P}$ or the observed actions occur at a different frequency than their probability under π_e i.e. $\pi_e \neq \hat{\pi}$. We describe the former as *transition sampling error* and the latter as *policy sampling error*. Transition sampling error may be unavoidable in a model-free setting – we do not know P and so we have no choice but to approximate it with sampling. However, we do know π_e and can use this knowledge to potentially correct policy sampling error. In the next Chapter, we present an algorithm that uses the knowledge of π_e to correct for policy sampling error and obtain a more accurate value function estimate.

3.4 Summary

In this chapter, we proved that batch linear TD(0) converges to two equivalent fixed-points in the per-step reward and discounted MRP and MDP setting. We then argued that these fixed-points are inaccurate since they ignore information about the evaluation policy. Finally, we used this limitation to motivate the focus of Chapter 4, our proposed algorithm that aims to correct for some of the inaccuracy of the value function learned by batch linear TD(0).

Chapter 4

Batch Linear Policy Sampling Error Corrected-TD(0)

In this chapter, we introduce the batch policy sampling error corrected-TD(0) (PSEC-TD(0)) algorithm that corrects for the policy sampling error in batch TD learning. Inspired by Theorem 2, we take the view that, for a finite batch generated by policy π_e, batch TD(0) evaluates the wrong policy; it evaluates the maximum likelihood policy, $\hat{\pi}$ instead of π_e. Under this view, PSEC-TD(0) treats policy sampling error as an off-policy learning problem and uses importance sampling (Precup et al., 2000a) to correct the weighting of TD(0) updates from $\hat{\pi}$ to π_e.

In addition to $\mathcal{D}$ and π_e, we assume we are given a set of policies, Π. Batch PSEC-TD(0) first computes the maximum likelihood estimate of the behavior policy:

$$\hat{\pi} \coloneqq \operatorname*{argmax}_{\pi' \in \Pi} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{L_\tau - 1} \log \pi'(a_t^\tau | s_t^\tau).$$

This estimation can be done in a number of ways. For example, in the tabular setting we could use the empirical count of actions in each state. This count-based approach is often intractable, and hence, in many problems of interest we must rely on function approximation. When using function approximation, the policy estimate can be obtained by minimizing a negative log-likelihood loss function.

Once $\hat{\pi}$ is computed, the batch PSEC-TD algorithm is the same as Algorithm 1 with

$\hat{\pi}$ replacing π_b in the importance sampling ratio. That is, for transition (s, a, r, s') in $\mathcal{D}$, the contribution to the weight update is:

$$\boldsymbol{w}_{i+1} \leftarrow \boldsymbol{w}_i + \alpha \left[\hat{\rho}\hat{y} - \boldsymbol{w}_i^T \boldsymbol{x}(s')\right] \boldsymbol{x}(s)$$

where $\hat{\rho} \coloneqq \frac{\pi_e(a|s)}{\hat{\pi}(a|s)}$, and $\hat{y} = r + \gamma \boldsymbol{w}_i^T \boldsymbol{x}(s')$. We will refer to $\hat{\rho}$ as the PSEC weight. Intuitively, when action a is observed more frequently than expected in state s, the PSEC-TD(0) new estimate $\hat{y}$ is decreased and when it is observed less frequently than expected, $\hat{y}$ is increased.

4.1 Convergence of Batch Linear PSEC-TD(0)

Chapter 3 defined two equivalent certainty-equivalence estimate definitions and showed that batch TD(0) converges to them. We now define a new certainty-equivalent estimate to which our new batch PSEC-TD(0) algorithm converges. Intuitively, the MDP-CEE estimate (Definition 2) is the exact value function for the *maximum likelihood estimate of the behavior policy,* $\hat{\pi}$, in the maximum likelihood estimate of the MDP environment; our new algorithm converges to the exact value function for π_e in the maximum likelihood estimate of the MDP environment. We define this new certainty-equivalent estimate as the *PSEC Markov Decision Process Certainty Equivalence Estimate* (PSEC-MDP-CEE) Value Function.

Definition 3. *PSEC Markov Decision Process Certainty Equivalence Estimate (PSEC-MDP-CEE) Value Function*

The PSEC-MDP-CEE is the value function, $\hat{v}^{\pi_e}_{\texttt{PSEC-MDP}}$*, that,* $\forall s_j \in \widehat{\mathcal{S}}$*, satisfies:*

$$\begin{aligned}\hat{v}^{\pi_e}_{\texttt{PSEC-MDP}}(s_j) = \sum_{a \in \widehat{\mathcal{A}}} \pi_e(a|s_j)[\bar{R}(s_j, a) + \\ \gamma \sum_{k \in \widehat{\mathcal{S}}} \widehat{P}(s_k|s_j, a)\hat{v}^{\pi_e}_{\texttt{PSEC-MDP}}(s_k))]\end{aligned} \tag{4.1}$$

Note that in Theorem 3, we show that batch linear PSEC-TD(0) converges to an equivalent form of Equation (4.1) in matrix notation (see, Section 2.1.1):

$$v^{\pi}(i) = \left[(I - \gamma \widehat{U})^{-1}(\hat{\mathbf{o}} + \hat{\mathbf{l}})\right]_i \tag{4.2}$$

In Appendix A.1.5, we show that Equation (4.1) and Equation (4.2) are equivalent.

Theorem 3 states that batch PSEC-TD(0) converges to the PSEC-MDP-CEE value function (Equation 4.1).

Theorem 3 (Batch Linear PSEC-TD(0) Convergence)**.** *For any batch whose observation vectors* $\{\boldsymbol{x}(s)|s \in \widehat{\mathcal{S}}\}$ *are linearly independent, there exists an* $\epsilon > 0$ *such that, for all positive* $\alpha < \epsilon$ *and for any initial weight vector, the predictions for linear PSEC-TD(0) converge under repeated presentations of the batch with weight updates after each complete presentation to the fixed-point (4.1).*

Proof. The proof for PSEC-TD(0) follows in large part the structure of the proof for TD(0). Below we highlight the salient points in the proof.

Batch linear PSEC-TD(0) makes an update to the weight vector, $\boldsymbol{w}_n$ (of dimension, length of the feature vector), after each presentation of the batch:

$$\boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \sum_{\tau \in \mathcal{D}} \sum_{t=1}^{L_\tau} \alpha \left[\hat{\rho}_t (\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1}) - \boldsymbol{w}_n^T \boldsymbol{x}_t \right] \boldsymbol{x}_t$$

where $\mathcal{D}$ is the batch of episodes, L_τ is the length of each episode τ, $\hat{\rho}_t$ is the PSEC correction weight at time t for a given episode τ, and α is the learning rate.

We can re-write the whole presentation of the batch of data in terms of the number of times there was a transition from state i to state j when taking action a in the batch i.e. $\hat{c}_{ij}^a = \hat{d}_i \hat{\pi}_i^a \hat{p}_{ij}^a$, where $\hat{d}_i$ is the number of times state $i \in \widehat{\mathcal{N}}$ appears in the batch.

$$
\begin{aligned}
\boldsymbol{w}_{n+1} &= \boldsymbol{w}_n + \sum_{\tau\in\mathcal{D}}\sum_{t=1}^{L_\tau} \alpha\left[\hat{\rho}_t(\bar{r}_t + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_{t+1}) - \boldsymbol{w}_n^T \boldsymbol{x}_t\right]\boldsymbol{x}_t \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}}\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{c}_{ij}^a\left[\hat{\rho}_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right]\boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}}\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{d}_i\hat{\pi}_i^a\hat{p}_{ij}^a\left[\hat{\rho}_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right]\boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}}\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{d}_i\hat{\pi}_i^a\hat{p}_{ij}^a\left[\frac{\pi_i^a}{\hat{\pi}_i^a}(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right]\boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}}\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{d}_i\hat{p}_{ij}^a\pi_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j)\boldsymbol{x}_i - \alpha \sum_{i\in\widehat{\mathcal{N}}}\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{d}_i\hat{\pi}_i^a\hat{p}_{ij}^a(\boldsymbol{w}_n^T \boldsymbol{x}_i)\boldsymbol{x}_i \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\widehat{\mathcal{N}}}} \hat{d}_i\boldsymbol{x}_i \sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \alpha \sum_{i\in\widehat{\mathcal{N}}} \hat{d}_i(\boldsymbol{w}_n^T \boldsymbol{x}_i)\boldsymbol{x}_i \sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{\pi}_i^a\hat{p}_{ij}^a
\end{aligned}
$$

$$
\begin{aligned}
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}} \hat{d}_i\boldsymbol{x}_i \left[\sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right] \qquad \sum_{j\in\widehat{\mathcal{N}}\cup\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{\pi}_i^a\hat{p}_{ij}^a = 1 \\
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}} \hat{d}_i\boldsymbol{x}_i \left[\left(\sum_{j\in\widehat{\mathcal{N}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a(\bar{r}_{ij}^a + \gamma \boldsymbol{w}_n^T \boldsymbol{x}_j)\right) + \left(\sum_{j\in\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a\bar{r}_{ij}^a\right) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right] \quad \text{If } \boldsymbol{x}_j \in \widehat{\mathcal{T}}, \boldsymbol{w}_n^T\boldsymbol{x}_j = 0
\end{aligned}
$$

$$
\begin{aligned}
&= \boldsymbol{w}_n + \alpha \sum_{i\in\widehat{\mathcal{N}}} \hat{d}_i\boldsymbol{x}_i \left[\left(\sum_{j\in\widehat{\mathcal{N}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a\bar{r}_{ij}^a\right) + \left(\gamma\sum_{j\in\widehat{\mathcal{N}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a\boldsymbol{w}_n^T\boldsymbol{x}_j\right) + \left(\sum_{j\in\widehat{\mathcal{T}}}\sum_{a\in\hat{\mathcal{A}}_i} \hat{p}_{ij}^a\pi_i^a\bar{r}_{ij}^a\right) - \boldsymbol{w}_n^T \boldsymbol{x}_i\right] \\
&= \boldsymbol{w}_n + \alpha\widehat{X}\widehat{D}\left[\hat{\mathbf{o}} + \gamma\widehat{U}\widehat{X}^T\boldsymbol{w}_n + \hat{\mathbf{l}} - \widehat{X}^T\boldsymbol{w}_n\right]
\end{aligned}
$$

where $\widehat{X}$ denotes the matrix (of dimensions, length of the feature vector by $|\widehat{\mathcal{S}}|$) with columns, $\boldsymbol{x}_i \in \widehat{\mathcal{S}}$ and $\widehat{D}$ is a diagonal matrix (of dimensions, $|\widehat{\mathcal{S}}|$ by $|\widehat{\mathcal{S}}|$) with $\widehat{D}_{ii} = \hat{d}_i$.

Assuming that as $n \to \infty$, $(I - \alpha\widehat{X}^T\widehat{X}\widehat{D}(I - \gamma\widehat{U}))^n \to 0$, we can drop the second term

and the sequence $\{\widehat{X}^T \boldsymbol{w}_n\}$ converges to:

$$\begin{aligned}
\lim_{n\to\infty} \widehat{X}^T \boldsymbol{w}_n &= \left(I - (I - \alpha \widehat{X}^T \widehat{X} \widehat{D}(I - \gamma \widehat{U}))\right)^{-1} (\alpha \widehat{X}^T \widehat{X} \widehat{D}(\hat{\mathbf{o}} + \hat{\mathbf{l}})) \\
&= (I - \gamma \widehat{U})^{-1} \widehat{D}^{-1} (\widehat{X}^T \widehat{X})^{-1} \alpha^{-1} \alpha \widehat{X}^T \widehat{X} \widehat{D}(\hat{\mathbf{o}} + \hat{\mathbf{l}}) \\
&= (I - \gamma \widehat{U})^{-1} (\hat{\mathbf{o}} + \hat{\mathbf{l}}) \\
\lim_{n\to\infty} \mathbb{E}\left[\boldsymbol{x}_i^T \boldsymbol{w}_n\right] &= \left[(I - \gamma \widehat{U})^{-1} (\hat{\mathbf{o}} + \hat{\mathbf{l}})\right]_i, \forall i \in \hat{\mathcal{N}}
\end{aligned}$$

What is left to show now is that as $n \to \infty$, $(I - \alpha \widehat{X}^T \widehat{X} \widehat{D}(I - \gamma \widehat{U}))^n \to 0$, which we can show by following the steps shown for Equation (3.4) in Chapter 3. Thus we prove convergence to the fixed-point (4.1).

□

4.1.1 Convergence to the MDP True Fixed-Point with Infinite Data

With batch linear PSEC-TD(0) we have corrected for the policy sampling error in batch linear TD(0). The remaining inaccuracy of the policy sampling corrected certainty-equivalence fixed-point is due to the transition dynamics sampling error. In a model-free setting, however, we cannot correct for this error in the same way we corrected the policy sampling error.

We argue that as the batch size approaches infinite, the maximum-likelihood estimate of the transition dynamics will approach the true transition dynamics i.e. $\hat{p} \to p$. It then follows that in expectation, the true value function will be reached. Thus, the batch linear PSEC-TD(0) with an infinite batch size will correctly converge to the true value function fixed-point given by Equation (A.7).

4.1.2 Remarks

We remark that convergence has only been shown for the on-policy setting. While PSEC-TD(0) can be applied in the off-policy setting, it may, like other semi-gradient TD methods, diverge when off-policy updates are made with function approximation (Baird, 1995). It is possible that a combination of PSEC-TD(0) and a more recent algorithm such as Emphatic TD (Mahmood et al., 2015) or Gradient-TD (Sutton et al., 2009) may result in provably convergent behavior with off-policy updates, however, that study is outside the scope of this

work. In our empirical analysis, we focus our attention on settings where on- and off-policy batch linear TD(0) are found to be convergent.

4.2 Extending PSEC to other TD Variants

In general, PSEC can improve any value function learning algorithm that computes the TD-error, δ, or equivalent errors. As an example, we consider the off-policy least-squares TD (LSTD) algorithm (Bradtke and Barto, 1996). The off-policy LSTD algorithm (Ghiassian et al., 2018) analytically computes the exact parameters that minimize the TD-error in a batch of data using the following steps:

$$
\begin{aligned}
A &= \sum_{(s,a,s')\in\mathcal{D}} \left[\hat{\rho}\boldsymbol{x}(s)(\boldsymbol{x}(s) - \gamma\boldsymbol{x}(s'))^T\right] \\
\mathbf{b} &= \sum_{(s,a,s')\in\mathcal{D}} R(s,a,s')\boldsymbol{x}(s) \\
\boldsymbol{w} &= A^{-1}\mathbf{b},
\end{aligned}
$$

where $\hat{\rho}$ is the PSEC weight. Even though we primarily consider TD(0) in this thesis, the extension to LSTD demonstrates that PSEC-TD can be extended to other value function learning algorithms.

4.3 Experiments

In this section, we empirically study PSEC-TD. Our experiments are designed to answer the following questions:

1. Does batch PSEC-TD(0) lower MSVE compared to batch TD(0)?
2. Does batch linear PSEC-TD(0) empirically converge to its certainty-equivalence solution?
3. Does PSEC improve any other TD-based algorithm?
4. What factors does PSEC depend on in the function approximation setting?

4.3.1 Empirical Setup

We first briefly describe the RL domains used in our experiments. Appendix A.2 includes additional details.

- **Gridworld:** In this domain, an agent navigates a 4×4 grid to reach a corner. The state and action spaces are discrete and we use a tabular representation for $\hat{v}^{\pi_e}$. PSEC-TD(0) uses count-based estimation for $\hat{\pi}$. The ground truth value function is computed with dynamic programming and the MSVE computation uniformly weights the error in each state.
- **CartPole:** In this domain, a cart agent attempts to balance a pole upright. The state space is continuous and action space is discrete. In our experiments, a neural network policy is trained using REINFORCE (Williams, 1992). The network has 2 hidden layers with 16 neurons. We evaluate PSEC with varying linear and non-linear representations for the value function. The $\hat{\pi}$ estimate maps the raw state features to a softmax distribution over the actions with varying linear and non-linear architectures. Since the true value function is unknown, we follow Pan et al. (2016) and use Monte Carlo rollouts from a fixed number of states sampled from episodes following the policy detailed above to approximate the ground-truth state-values of those states. We then compute the MSVE between the learned values and the average Monte Carlo return from these sampled states.
- **InvertedPendulum:** This domain is similar as CartPole in terms of the objective: to balance a pole upright. However, the state and action spaces are both continuous. In our experiments, a neural network policy is trained using Proximal Policy Optimization (PPO) (Schulman et al., 2017). The network has 2 hidden layers with 64 neurons each. We evaluate PSEC with varying linear and non-linear representations for the value function. The $\hat{\pi}$ estimate consists of two components: 1) a linear or non-linear mapping from raw state features to the mean vector of a Gaussian distribution, and 2) parameters representing the log standard deviation of each element of the output vector. Similar to above, since the true value function is unknown, we compute Monte Carlo rollouts for sampled states.

In all experiments, the value function learning algorithm iterates over the batch of data until convergence, after which the MSVE of the final value function is computed. All the MSVE values shown are averaged over a set number of trials with the errors bars showing the 95% confidence interval. Some experiments include a parameter sweep over the hyperparameters, which can be found in Appendix A.2.

4.3.2 Tabular Setting: Discrete States and Actions

In these set of experiments, we consider two variants of PSEC-TD that differ in the placement of the PSEC weight. In the off-policy TD(0) algorithm, the importance weight is sometimes applied to the TD-error instead of the new value estimate (Ghiassian et al., 2018). For off-policy TD(0), these placements are equivalent in expectation although the method using the TD-error has been reported to perform better in practice (Ghiassian et al., 2018). Following these results, our experiments consider two variants of PSEC-TD:

- PSEC-TD-Estimate: Applies $\hat{\rho}$ to the new estimate:

$$\hat{y} = R + \gamma \boldsymbol{w}^T \boldsymbol{x}(s').$$

- PSEC-TD: Applies $\hat{\rho}$ to the full TD error:

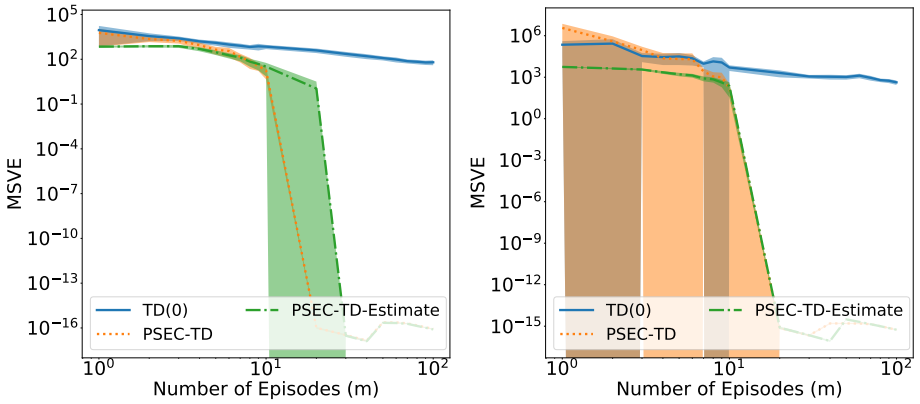
$$\delta = (R + \gamma \boldsymbol{w}^T \boldsymbol{x}(s')) - \boldsymbol{w}^T \boldsymbol{x}(s).$$

For off-policy TD(0), we always use the variant that applies the importance weight to the TD-error.

Performance of PSEC Variants for Varying Batch Size

Figures 4.1a and 4.1b compare the performance of TD(0) to the two variants of PSEC-TD(0) that arise from the location of the PSEC weight. For off-policy TD(0) we only consider the PSEC-TD variant as we found multiplying the new estimate by the weight was divergent.

Figure 4.1 shows that both variants of PSEC-TD(0) lower MSVE compared to batch TD(0) in both the on- and off-policy settings. All methods show higher variance for the

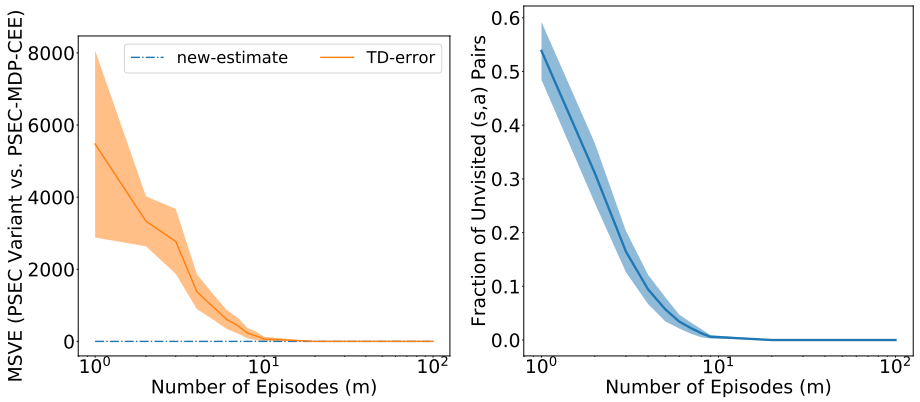


a On-Policy b Off-Policy

Figure 4.1: Deterministic Gridworld experiments. Both axes are log-scaled, leading to the asymmetric confidence intervals. Errors are computed over 50 trials with 95% confidence intervals. Figure 4.1a and Figure 4.1b compare the final errors achieved by variants of PSEC-TD(0) and TD(0) for varying batch sizes for on- and off-policy cases respectively.

off-policy setting, however, PSEC-TD(0) variants still provide more accurate value function estimates. The gap between PSEC-TD(0) and TD(0) increases dramatically with more data; we discuss this observation in the section below and connect to our earlier convergence results.

Convergence to the PSEC-Certainty-Equivalence



a b

Figure 4.2: Additional Gridworld experiments. Errors are computed over 50 trials with 95% confidence intervals. Figure 4.2a shows MSVE achieved by both variants of linear batch PSEC-TD(0), PSEC-TD and PSEC-TD-Estimate, with respect to the PSEC-MDP-CEE (4.1). Figure 4.2b shows the fraction of unvisited state-action pairs.

To address our second empirical question, we empirically verify that both variants of batch linear PSEC, PSEC-TD and PSEC-TD-Estimate converge to the dynamic programming computed PSEC-MDP-CEE value function (4.1) in Gridworld. According to Theorem 3, batch linear PSEC-TD-Estimate converges to the fixed-point (4.1) for all batch sizes. We also empirically confirm that the other variant of PSEC, PSEC-TD converges to the same fixed-point (4.1) when the following condition holds true: only when all non-zero probability actions for each state in the batch have been sampled at least once. We note that when this condition is false, PSEC-TD-Estimate treats the value of taking that action as 0. For example, if a state, s, appears in the batch and an action, a, that could take the agent to state s' does not appear in the batch, then PSEC-TD-Estimate treats the new estimate $R + \gamma \boldsymbol{w}^T \boldsymbol{x}(s')$ as 0, which is also done by the dynamic programming computation (4.1). We note that PSEC-TD converges to the fixed-point (4.1) only when this condition is true since the PSEC weight requires a fully supported probability distribution when applied to the current estimate (refer to Section 4.1). From Figure 4.2a and Figure 4.2b, we can see that this condition is true at batch size of 11 episodes. We also see from Figure 4.1a and 4.1b that this point is the point when the benefit of PSEC is fully realized. When this condition is met, PSEC is able to fully correct all the policy sampling error for state-actions that occurred in the batch.

Effect of Environment Stochasticity

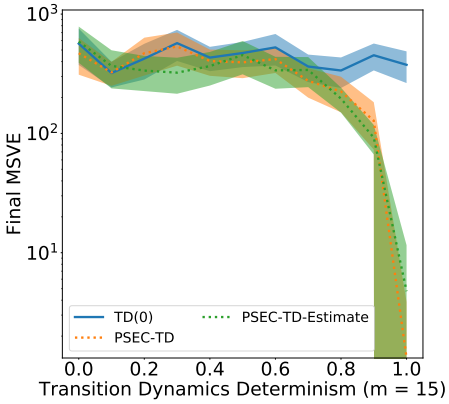
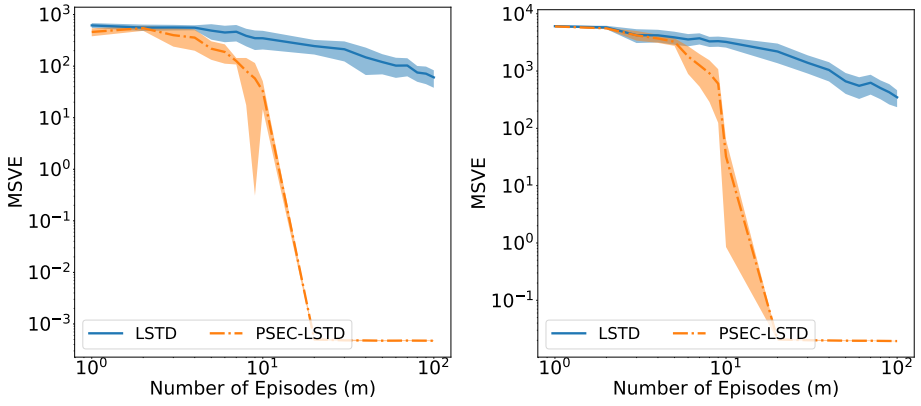


Figure 4.3: Additional Gridworld experiments. Errors are computed over 50 trials with 95% confidence intervals. Figure 4.3 is a y-axis log scaled graph that shows the final error (averaged over 100 trials) achieved by the two variants of PSEC-TD(0) and TD(0) for a given batch size (15 episodes) with varying levels of determinism of the transition dynamics.

According to Theorem 2, TD suffers from policy *and* transition dynamics sampling error. We study this observation through Figure 4.3, which illustrates how the performance of PSEC changes with different levels of transition dynamics determinism for a fixed batch size. In Gridworld, the determinism is varied according to a parameter, p, where the environment becomes purely deterministic or stochastic as $p \to 1$ or $p \to 0$ respectively. From Theorem 3 we expect PSEC to fully correct for the policy sampling error but not transition dynamics sampling error. Figure 4.3 confirms that PSEC is achieves a lower final MSVE than TD as $p \to 1$. As $p \to 0$, the transition dynamics become the dominant source of sampling error and PSEC-TD(0) and TD(0) perform similarly.

PSEC-Least Squares TD



a On-Policy b Off-Policy

Figure 4.4: LSTD Gridworld experiments. Both axes are log-scaled. Errors are computed over 50 trials with 95% confidence intervals. Figure 4.4a and Figure 4.4b compare the final errors achieved by PSEC-LSTD and LSTD for varying batch sizes for on- and off-policy cases respectively.

We address our third empirical question by applying PSEC to the LSTD algorithm in the Gridworld domain. Figure 4.4 compares the MSVE of LSTD and PSEC-LSTD and shows that PSEC-LSTD obtains lower MSVE estimates. The results are similar to those shown for regular TD(0) and PSEC-TD(0) (Figure 4.1) and suggest that the benefits of correcting for policy sampling error with PSEC weights can be easily extended to other value function learning algorithms.

4.3.3 Function Approximation: Continuous States and Discrete Actions

In this set of experiments, we answer our fourth empirical question on function approximation in PSEC. In particular, we study how various components of the training process of the PSEC policy impact the end performance. We conduct our experiments on the CartPole domain.

Our experiments focus on applying only the second variant of PSEC, PSEC-TD, since we found that PSEC-TD-Estimate diverges. The results shown below are for the on-policy case. To better understand the intricacies involved, we conduct our experiments on a fixed batch size and tune various components of the PSEC policy training process to better understand how they affect end performance. In these experiments, we have three function approximators: one for the value function; one to estimate the behavior policy, and the pre-learned behavior policy itself. When we refer to one of these approximators as "fixed", we mean that its architecture is unchanged.

In each experiment below, unless stated, the following components were fixed: a batch size of 10 episodes, the value function was represented with a neural network of single hidden layer of 512 neurons using tanh activation, the gradients were normalized to unit norm before the gradient descent step was performed, we used a learning rate of 1.0 and decayed the learning rate by 10% every 50 presentations of the batch to the algorithm. The true MSVE was computed by 200 Monte Carlo rollouts for 150 sampled states. In all PSEC training settings, PSEC performs gradient steps using the full batch of data and uses a separate batch of data as the validation data, and terminates training according to early stopping. Any remark on statistical significance is according to Welch's test (WELCH, 1947) with significance level of 0.05.

Effect of Value Function Model Architecture

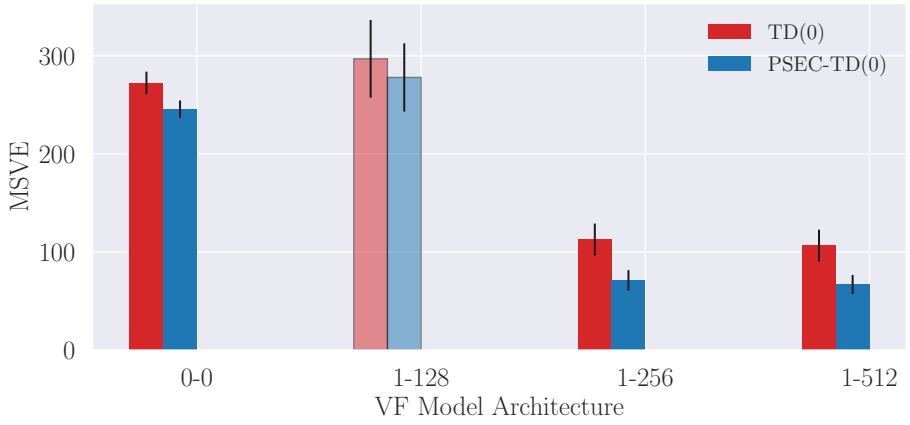


Figure 4.5: Comparing performance of PSEC with varying VF model architectures against TD for fixed batch size of 10 episodes on performance of TD and PSEC-TD. Results shown are averaged over 300 trials and error bar is 95% confidence. Darker shades represent statistically significant result. The label on the x axis shown is (# hidden layers - # neurons). The value function represented by 0-0 is a linear mapping with no activation function.

Figure 4.5 illustrates the impact of different value function classes on the performance of TD and PSEC, while holding the PSEC model and behavior policy architectures fixed. PSEC used a model architecture of 3 hidden layers with 16 neurons each and tanh activation, and learning rate of 0.025.

We generally found that a more expressive value function representation corresponded to better overall MSVE performance in both algorithms. Interestingly, however, the neural network with one hidden layer and 128 neurons performed worse than the linear representation, and PSEC's improvement over TD with this value function representation was not statistically significant. We also found that the gap between PSEC and TD improved as the value function representation became richer. We believe that even though PSEC finds a more accurate fixed point than TD in the space of all value functions, the shown difference between the two algorithms is dependent on the space of representable value functions according to the value function representation – a more representable function class can capture the difference between the two algorithms better.

Effect of PSEC Learning Rate

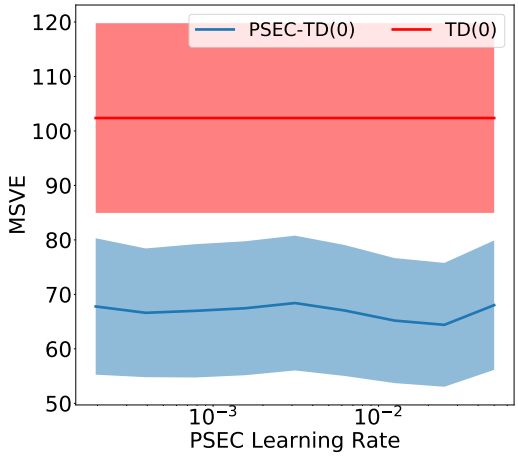


Figure 4.6: Comparing performance of PSEC with varying learning rates against TD for a fixed batch size of 10 episodes. Results shown are averaged over 300 trials and error bar is 95% confidence.

Figure 4.6 compares the performance of PSEC-TD vs TD for varying learning rates of the PSEC policy for a fixed batch size of 10 episodes, while holding the value function, PSEC model, and behavior policy architectures fixed. The PSEC policy used in this experiment was a neural network with: 3 hidden layers with 16 neurons each and tanh activation. Since TD does not use PSEC, its error for a given batch size is independent of the PSEC learning rate. From above, PSEC appears to be relatively stable in its improvement over TD regardless of the learning rate used.

Effect of PSEC Model Architecture

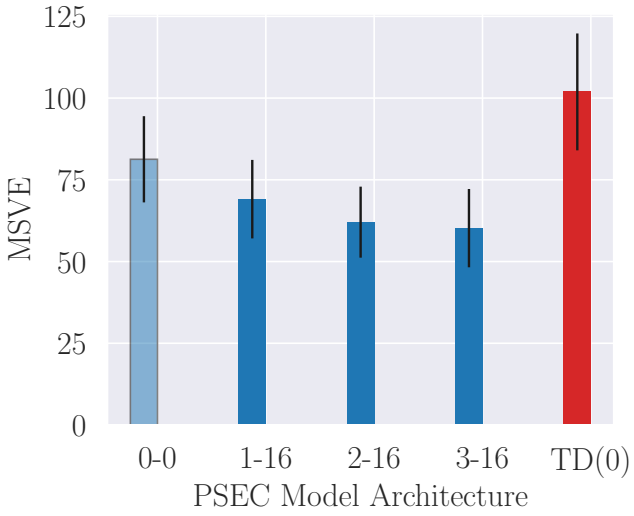


Figure 4.7: Comparing performance of PSEC with varying model architectures against TD for a fixed batch size of 10 episodes. Results shown are averaged over 300 trials and error bar is 95% confidence. Darker shades represent statistically significant result. The label on the x axis shown is (# hidden layers - # neurons). The value function represented by 0-0 is a linear mapping with no activation function.

Figure 4.7 compares the performance of PSEC against TD with varying PSEC neural network model architectures, while the value function and behavior policy architectures are fixed. PSEC used a learning rate of 0.025.

The chosen PSEC neural network models are with respect to the behavior policy described earlier, a 2 hidden layered with 16 neurons architecture. In general, we found that more expressive network models produced better PSEC corrections since they were able to better capture the maximum likelihood estimate of the policy from the data. Unlike the neural network PSEC policies, the linear function PSEC policy did not produce a statistically significant improvement over TD.

Varying PSEC Training Style

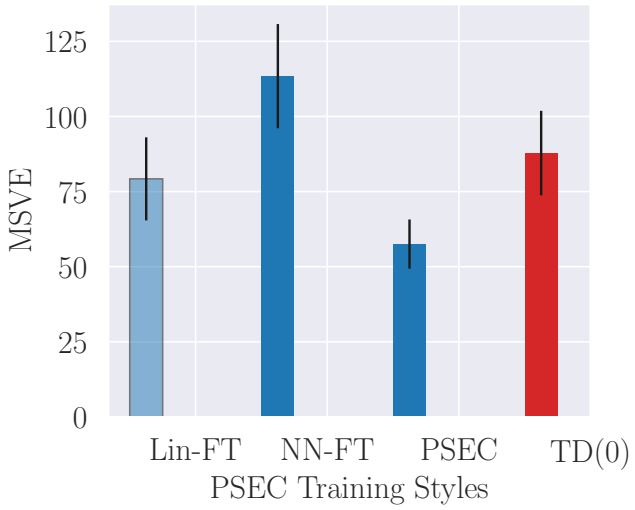


Figure 4.8: Comparing performance of varying training styles of PSEC against TD for a fixed batch size of 10 episodes. Results shown are averaged over 300 trials and shaded region is 95% confidence. Darker shades represent statistically significant result.

Figure 4.8 illustrates the performance of three variants of PSEC, while holding the value-function PSEC model, and behavior policy architectures fixed. All three variants use the same PSEC model architecture as that of the behavior policy, and each used a learning rate of 0.025 with tanh activation. The three variants are as follows: 1) Lin-FT is when PSEC initializes the PSEC model to the weights of the behavior policy and trains on the batch of data by finetuning only the last linear layer, 2) NN-FT is when PSEC initializes the PSEC model to the weights of the behavior policy but finetunes the all the weights of the network, and 3) PSEC uses the same training style in the previous experiments, where the model is initialized randomly and all the weights are tuned. We found that Lin-FT performed similarly to TD with a statistically insignificant improvement over TD; we believe this may be so since Lin-FT is initialized to the behavior policy and since there are only few weights to change in the linear layer, the newly learned Lin-FT is still similar to the behavior policy, which would produce PSEC corrections close to 1 (equivalent to TD). Interestingly, tuning all the weights of the neural network did better when the model was initialized randomly versus when it was

initialized to the behavior policy.

Effect of Underfitting and Overfitting during PSEC Policy Training

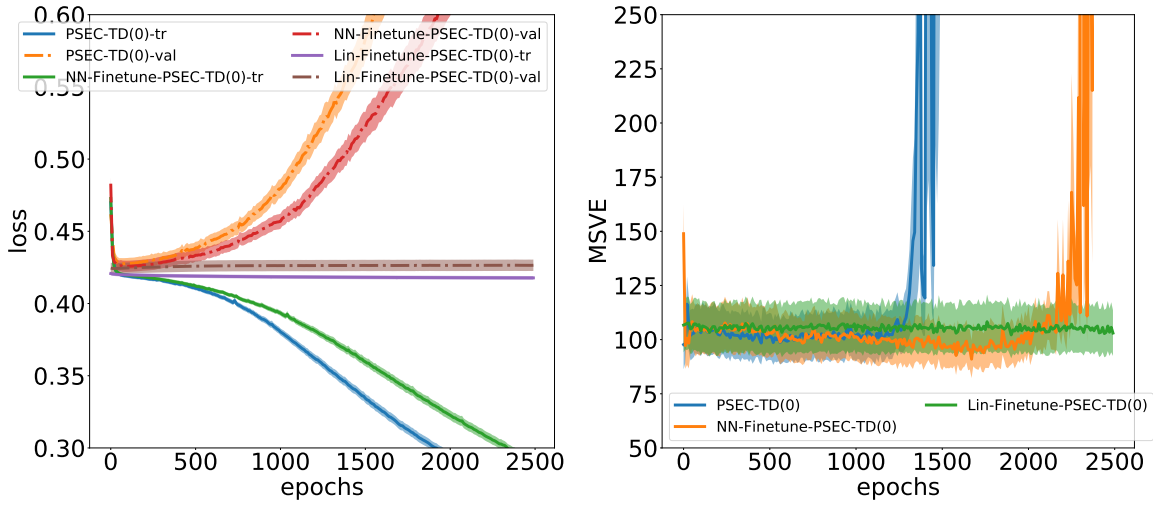


Figure 4.9: Comparing MSVE achieved and cross entropy loss by variants of PSEC after each epoch of training for a fixed batch size of 10 episodes. Results shown are averaged over 50 trials and shaded region is 95% confidence.

This experiment attempts to give an understanding of how the MSVE achieved by each PSEC variant is dependent on the number of epochs the PSEC model was trained for, while holding the value-function, PSEC and behavior policy architectures fixed. We conduct the experiment as follows: the PSEC algorithm performs 10 gradient descent steps (epochs) on the full batch of data, after which the resulting training and validation mean cross-entropy losses are plotted along with the MSVE achieved by that trained PSEC policy. For example, after 10 epochs, the training and validation loss of the PSEC model was nearly 0.5 and the model achieved an MSVE of nearly 150.

Since computing the MSVE can be computationally expensive, as it requires processing the batch until the value function converges, we change the learning rate decay schedule to starting with a learning rate of 1.0 but decaying learning rate by 50% every 50 presentations of the full batch to the algorithm (this change is also the reason why these results may be different from the ones shown earlier). All PSEC variants used a learning rate of 0.025 and PSEC model architecture of 2 hidden layers with 16 neurons each and tanh activation.

Figure 4.9 suggests that performance of PSEC, regardless of the variant, depends on the number of epochs it was trained for. Naturally, we do want to fit sufficiently well to the data, and the graph suggests that some overfitting is tolerable. However, if overfitting becomes extreme, PSEC's performance suffers, resulting in MSVE nearly 1000 times larger than the minimum error achieved (not shown for clarity). From the graph, we can see that the PSEC variant, which is initialized randomly, starts to extremely overfit before the NN-finetune variant does, causing its MSVE to degrade before that of NN-finetune variant. We also see that the Lin-finetune variant is not able to overfit since the last linear layer may not be expressible enough to overfit, causing it to have a relatively stable MSVE across all epochs.

Effect of Behavior Policy Distribution

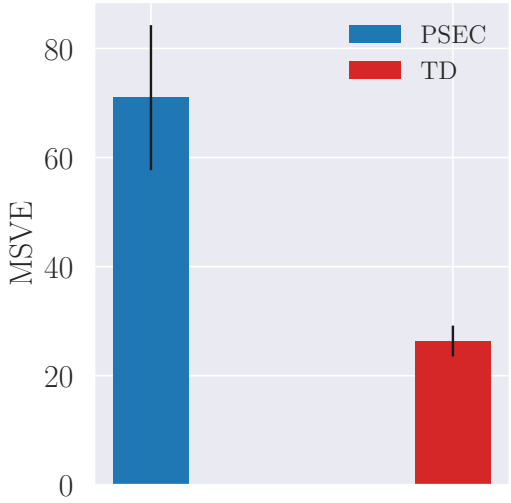


Figure 4.10: Comparing performance of PSEC against TD for a fixed batch size of 100 episodes when the behavior policy models a discontinuous function. Results shown are averaged over 30 trials and shaded region is 95% confidence.

So far, PSEC has used a function approximator of the similar function class as that of the behavior policy i.e. both the PSEC models and the behavior policy were neural networks of similar architectures. In this experiment, we evaluate the performance of PSEC with a neural network policy when the behavior policy that models a discontinuous function generates a larger batch size of 100 episodes, while the value function and PSEC model architectures are

fixed. In particular, we use a behavior policy in CartPole that does the following: if the sign of the pole angle is negative, move left with probability 0.75 and right with probability 0.25, and if the sign of the pole angle is positive, move right with probability 0.75 and left with probability 0.25. The PSEC policy is a neural network with 3 hidden layers with 16 neurons each with tanh activation.

Figure 4.10 shows that PSEC performs much worse than TD when the behavior policy is the discontinuous type function described above. We reason that the neural network finds it difficult to compute the MLE of the data since this discontinuous distribution is "hard" to model; therefore, producing incorrect PSEC weights, which degrade its performance. While we can largely ignore the distribution of the behavior policy, this experiment shows that PSEC may suffer in situations like the one described.

4.3.4 Function Approximation: Continuous States and Actions

This set of experiments also answers the fourth empirical question on function approximation in PSEC, and as done above, studies how various components of the training process of the PSEC policy impact the end performance. We conduct our experiments on the InvertedPendulum domain.

Our experiments focus on applying only the second variant of PSEC, PSEC-TD, since we found that PSEC-TD-Estimate diverges. Unlike in Section 4.3.3, we do not evaluate the performance of the finetuning variants of PSEC-TD since they performed worse on preliminary tests than regular PSEC-TD, which initializes the PSEC model randomly. The results shown below are for the on-policy case. To better understand the intricacies involved, we conduct our experiments on a fixed batch size and tune various components of the PSEC policy training process to better understand how they affect end performance. In these experiments, we have three function approximators: one for the value function; one to estimate the behavior policy, and the pre-learned behavior policy itself. When we refer to one of these approximators as "fixed", we mean that its architecture is unchanged.

In each experiment below, unless stated, the following components were fixed: a batch size of 20 episodes, the value function was represented with a neural network of 2 hidden layer with 64 neurons each using tanh activation, the gradients were normalized to unit norm

before the gradient descent step was performed, we used a learning rate of 1.0 and decayed the learning rate by 5% every 10 presentations of the batch to the algorithm. The true MSVE was computed by 100 Monte Carlo rollouts for 100 sampled states. In all PSEC training settings, PSEC performs gradient steps using the full batch of data and uses a separate batch of data as the validation data, and terminates training according to early stopping. Any remark on statistical significance is according to Welch's test (WELCH, 1947) with significance level of 0.05.

Effect of Value Function Model Architecture

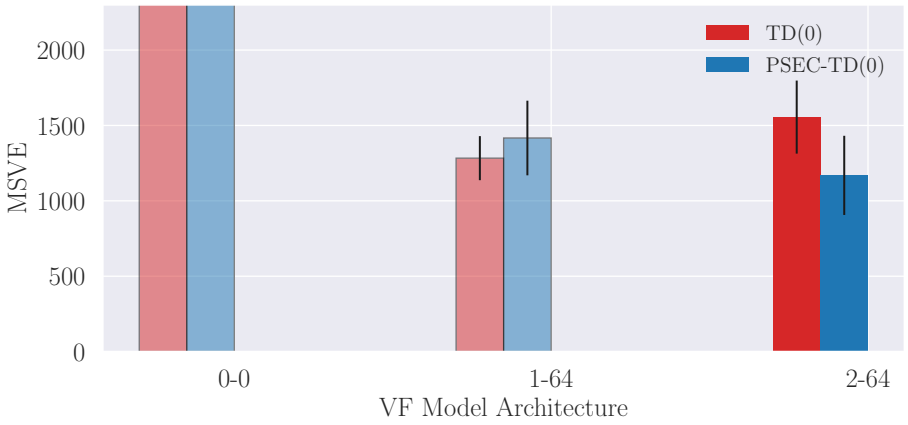


Figure 4.11: Comparing performance of PSEC with varying VF model architectures against TD for fixed batch size of 20 episodes on performance of TD and PSEC-TD. Results shown are averaged over 350 trials and error bar is 95% confidence. Darker shades represent statistically significant result. The label on the x axis shown is (# hidden layers - # neurons). The value function represented by 0-0 is a linear mapping with no activation function.

Figure 4.11 compares the performance of PSEC-TD and TD for different value function representations, while holding the PSEC estimation policy and behavior policy architectures fixed. The PSEC policy is 2 hidden layers with 64 neurons each and used a learning rate of 0.000781. Similar to the earlier findings, a more expressive value function captured the difference between PSEC and TD much better. For less expressive value functions, we found that both methods performed poorly, and any difference between the two was not statistically significant. Note that the linear architecture used produced an MSVE of ~ 5800 in both

algorithms (not shown for clarity) and the difference was statistically insignificant.

Effect of PSEC Learning Rate

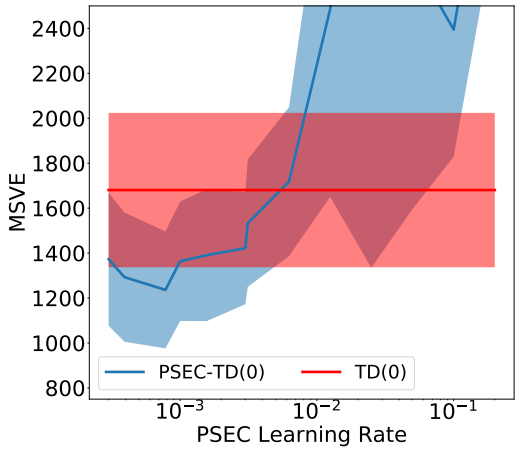


Figure 4.12: Comparing performance of PSEC with varying learning rates against TD for a fixed batch size of 20 episodes. Results shown are averaged over 200 trials and error bar is 95% confidence.

Figure 4.12 compares the performance of PSEC-TD to TD with varying learning rates for PSEC, while holding the PSEC and value function architecture fixed. Unlike earlier, the PSEC learning rate heavily influences the learned value function in the continuous state and action setting. In general, PSEC performance heavily degraded when the learning rate increased (y-axis limited for clarity). Among the tested learning rates, 0.000781 was the optimal, giving a statistically significant result.

Effect of PSEC Model Architecture

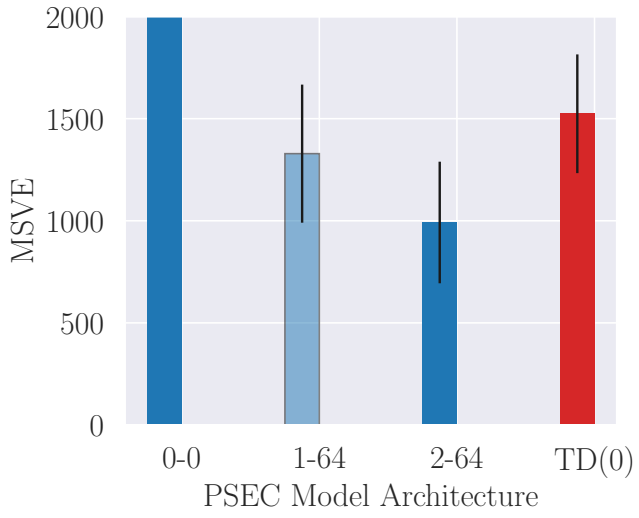


Figure 4.13: Comparing performance of PSEC with varying model architectures against TD for a fixed batch size of 20 episodes. Results shown are averaged over 200 trials and error bar is 95% confidence. Darker shades represent statistically significant result. The label on the x axis shown is (# hidden layers - # neurons). The value function represented by 0-0 is a linear mapping with no activation function.

Figure 4.13 compares the performance of PSEC-TD with TD with varying PSEC model architectures, while holding the value-function and behavior policy architectures fixed. All the shown PSEC architectures used a learning rate of learning rate 0.000781. Similar to our earlier findings, a more expressive network was able to better model the batch of data and produce a statistically significant improvement over TD. Less expressive PSEC models performed worse than TD, and any improvement was statistically insignificant. Note that the linear architecture used produced an MSVE of ~ 5800 (not shown for clarity) and its poor performance with respect to TD(0) was statistically significant.

Effect of Underfitting and Overfitting during PSEC Policy Training

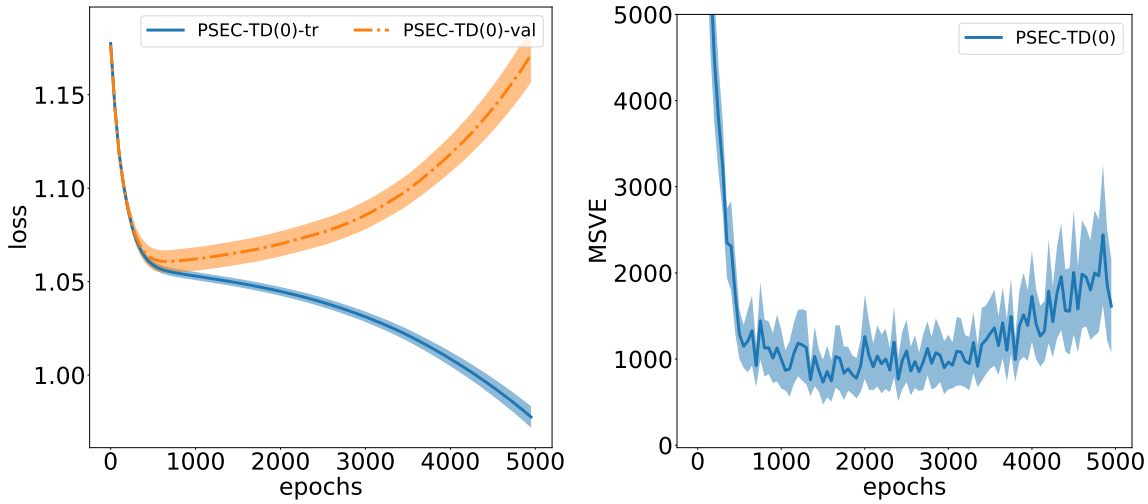


Figure 4.14: Comparing MSVE achieved and loss by PSEC after each epoch of training for a fixed batch size of 20 episodes. Results shown are averaged over 100 trials and shaded region is 95% confidence.

The motivation behind this experiment is the same as the one presented in the continuous state and discrete action case, i.e. understanding how the MSVE achieved by PSEC is dependent on the number of epochs the PSEC model was trained for, while holding the value-function, PSEC and behavior policy architectures fixed. The experiment is conducted in a similar manner as before except we perform 50 gradient descent steps (epochs) before plotting the training and validation loss, and MSVE achieved by PSEC after the gradient steps. The loss shown here is a regression loss detailed in Appendix A.2.3.

When computing MSVE, we used the learning rate schedule specified at the beginning of this section. PSEC used a learning rate of 0.000781 and model architecture of 2 hidden layers with 64 neurons each and tanh activation.

Figure 4.14 suggests the similar ideas that we discovered from Figure 4.9, i.e. some overfitting to the data is tolerable, perhaps even desirable, but extreme overfitting leads to poor performance. If some overfitting is desirable, then early stopping is not the preferred principled approach to terminate PSEC model training.

4.4 Summary

In this chapter, we gave a theoretical analysis of the more accurate fixed point that PSEC-TD converges to, PSEC-MDP-CEE. We then laid out and answered several empirical questions to better understand PSEC and its benefits. In general, we showed that PSEC brings immediate benefit to the tabular setting and that PSEC shows large potential in the function approximation setting, given favorable training settings. Automatically finding these settings in a principled manner in the function approximation case is an important future direction.

Chapter 5

Conclusion and Future Work

5.1 Summary

In batch value function approximation, we observed that TD(0) may converge to an inaccurate estimate of the value function due to policy sampling error. We proposed PSEC-TD(0) as a method to correct this error and show that it leads to a more desirable fixed point as compared to TD(0). In this thesis, we theoretically analyzed PSEC-TD and empirically evaluated it in the tabular and function approximation settings. Our empirical study validated that PSEC converged to a more accurate fixed point than TD, and studied how the numerous components in the PSEC training setup impact its performance with respect to TD.

5.2 Limitations of PSEC

Despite the performance benefits that batch PSEC-TD(0) introduced, there are limitations. First, it requires knowledge of the evaluation policy, which on-policy TD(0) does not. This comparative disadvantage is only for the on-policy setting as both TD(0) and PSEC-TD(0) require knowledge of the evaluation policy for the off-policy setting. Additionally, PSEC-TD(0), in the off-policy case, has the advantage of not requiring knowledge of the behavior policy π_b. Second, if either of the finetuning variants of PSEC were used, it would require knowing the paramaters of the policy that generated the data. Third, the policy estimation step required by PSEC-TD(0) could potentially be computationally expensive. For instance, requiring the computation and storage of $\mathcal{O}(|\mathcal{S}||\mathcal{A}|)$ parameters in the tabular setting.

5.3 Future Work

There are many directions for future work given the fundamental nature of the TD(0) algorithm. First, our theoretical and experimental work focused on improving the performance of batch TD(0). We expect that a variant of PSEC can improve value function learning with n-step TD and TD(λ) though we have not yet studied this line of work. Second, it remains to be seen how the PSEC weight could be applied to *online* value function learning. Third, with an improved value function learning algorithm, it would be interesting to see if an agent can learn better control policies. Fourth, it would be interesting to theoretically and empirically study PSEC when learning the state-action values. Finally, as mentioned in Chapter 4, automatically finding the optimal training setting for PSEC in the function approximation setting is another important direction for future work.

Appendix A

Supplemental Material

In this appendix, we include additional details to aid in the theoretical analysis in Chapters 3 and 4, and elaborate on the experimental details from Chapter 4. The notations used in this section are included in Chpater 2.

A.1 Fixed-point for an MDP in the Per-step Reward and Discounted Case

In this section, we establish several fixed-points that we expect the value function to converge to in the discounted per-step reward case for an MRP and MDP. Note that Sutton (1988) derived these fixed-points for MRPs when rewards are only received on termination and there is no discounting. We first specify the fixed-points for an MRP in the discounted per-step reward case, and then extend this result for MDPs.

For both an MRP and MDP, we establish two types of fixed-points in the per-step reward and discounted case. The first fixed-point is the *true* fixed-point in that it is the value function computed assuming that we have access to the true policy and transition dynamics distributions. Ideally, we would like our value function learning algorithms to converge to this fixed-point. The second fixed-point is the *certainty-equivalence estimate* fixed-point, which is the value function computed using the maximum-likelihood estimates of the policy and transition dynamics from a batch of fixed data. We note that due to sampling error in the policy and transition dynamics, the certainty-equivalence estimate is an *inaccurate* estimate

of the true value function. Finally, and only for MDPs, we specify the fixed-point that we expect the value function to learn after applying PSEC.

A.1.1 MRP True Fixed-Point

The true value function v for a state, $i \in \mathcal{N}$, induced by the policy-integrated transition dynamics p^π, reward function r, and policy, π, is given by:

$$
\begin{aligned}
v(i) &= \sum_{j \in \mathcal{N} \cup \mathcal{T}} p^\pi_{ij} \left[r_{ij} + \gamma v(j) \right] && \text{Bellman equation} \\
&= \sum_{j \in \mathcal{N}} p^\pi_{ij} \left[r_{ij} + \gamma v(j) \right] + \sum_{j \in \mathcal{T}} p^\pi_{ij} r_{ij} && \text{expected return from } \mathcal{T}, v(\mathcal{T}) = 0 \\
&= \sum_{j \in \mathcal{T}} p^\pi_{ij} r_{ij} + \sum_{j \in \mathcal{N}} p^\pi_{ij} \left[r_{ij} + \gamma \left[\sum_{k \in \mathcal{N} \cup \mathcal{T}} p^\pi_{jk} \left[r_{jk} + \gamma v(k) \right] \right] \right] && \text{recursively apply } v(i)
\end{aligned}
$$

$$
\begin{aligned}
v(i) = & \sum_{j \in \mathcal{T}} p^\pi_{ij} r_{ij} + \sum_{j \in \mathcal{N}} p^\pi_{ij} r_{ij} + \gamma \sum_{j \in \mathcal{N}} p^\pi_{ij} \sum_{k \in \mathcal{N} \cup \mathcal{T}} p^\pi_{jk} r_{jk} \\
& + \gamma^2 \sum_{j \in \mathcal{N}} p^\pi_{ij} \sum_{k \in \mathcal{N} \cup \mathcal{T}} p^\pi_{jk} v(k) \\
= & \sum_{j \in \mathcal{T}} p^\pi_{ij} r_{ij} + \sum_{j \in \mathcal{N}} p^\pi_{ij} r_{ij} \\
& + \gamma \sum_{j \in \mathcal{N}} p^\pi_{ij} \sum_{k \in \mathcal{N}} p^\pi_{jk} r_{jk} + \gamma \sum_{j \in \mathcal{N}} p^\pi_{ij} \sum_{k \in \mathcal{T}} p^\pi_{jk} r_{jk} \\
& + \gamma^2 \sum_{j \in \mathcal{N}} p^\pi_{ij} \sum_{k \in \mathcal{N}} p^\pi_{jk} v(k) && \text{splitting } \mathcal{N} \text{ and } \mathcal{T}
\end{aligned}
$$

We define vectors, $\mathbf{h}$ and $\mathbf{m}$ with, $[\mathbf{h}]_i = \sum_{j \in \mathcal{T}} p^\pi_{ij} r_{ij}$, $[\mathbf{m}]_i = \sum_{j \in \mathcal{N}} p^\pi_{ij} r_{ij}$, and Q is the true transition matrix of the Markov reward process induced by π and P, i.e., $[Q]_{ij} = p^\pi_{ij}$.

Then continuing from above, we have

$$v(i) = [\mathbf{h}]_i + [\mathbf{m}]_i + \gamma Q[\mathbf{h}]_i + \gamma Q[\mathbf{m}]_i + \gamma^2 Q^2[\mathbf{h}]_i + \gamma^2 Q^2[\mathbf{m}]_i + \dots \quad \text{unrolling } v(\mathcal{N} \cup \mathcal{T}) \tag{A.1}$$

$$= \left[\sum_{k=0}^{\infty} (\gamma Q)^k (\mathbf{m} + \mathbf{h})\right]_i \tag{A.2}$$

$$v(i) = \left[(I - \gamma Q)^{-1}(\mathbf{m} + \mathbf{h})\right]_i \tag{A.3}$$

The existence of the limit and inverse are assured by Theorem A.1 in Sutton (1988). The theorem is applicable here since $\lim_{k\to\infty}(\gamma Q)^k = 0$.

A.1.2 MRP Certainty-Equivalence Fixed -Point

For the certainty-equivalence fixed-point, we consider a batch of data, $\mathcal{D}$. We follow the same steps and similar notation from Equation (A.3), with the slight modification that the maximum-likelihood estimate (MLE) of the above quantities according to $\mathcal{D}$, is given with a hat ($\hat{}$) on top of the quantity. The observed sets of non-terminal and terminal states in the batch are given by $\widehat{\mathcal{N}}$ and $\widehat{\mathcal{T}}$ respectively.

Then similar to above, we can derive the certainty-equivalence estimate of the value function according to the MLE of the MRP transition dynamics from the batch for a particular state, $i \forall \widehat{N}$ is:

$$\hat{v}(i) = \left[(I - \gamma \widehat{Q})^{-1}(\hat{\mathbf{m}} + \hat{\mathbf{h}})\right]_i \tag{A.4}$$

A.1.3 MDP True Fixed-Point

The true value function, v^π, for a policy, π, for a state, $i \forall \mathcal{N}$, induced by the transition dynamics and reward function, p and r is given by:

$$
\begin{aligned}
v^\pi(i) &= \sum_{a\in\mathcal{A}} \pi_i^a \sum_{j\in\mathcal{N}\cup\mathcal{T}} p_{ij}^a \left[r_{ij}^a + \gamma v^\pi(j)\right] && \text{Bellman equation} \\
&= \sum_{a\in\mathcal{A}} \pi_i^a \sum_{j\in\mathcal{N}} p_{ij}^a \left[r_{ij}^a + \gamma v^\pi(j)\right] + \sum_{a\in\mathcal{A}} \pi_i^a \sum_{j\in\mathcal{T}} p_{ij}^a r_{ij}^a && \text{expected return from } \mathcal{T}, v^\pi(\mathcal{T}) = 0
\end{aligned}
$$

$$
\begin{aligned}
v^\pi(i) &= \sum_{a\in\mathcal{A}} \pi_i^a \sum_{j\in\mathcal{T}} p_{ij}^a r_{ij}^a + \sum_{a\in\mathcal{A}} \pi_i^a \sum_{j\in\mathcal{N}} p_{ij}^a \left[r_{ij}^a + \gamma \left[\sum_{a'\in\mathcal{A}} \pi_j^{a'} \sum_{k\in\mathcal{N}\cup\mathcal{T}} p_{jk}^{a'} \left[r_{jk}^{a'} + \gamma v^\pi(k)\right]\right]\right] \\
&= \sum_{j\in\mathcal{T}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a + \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a + \gamma \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a \sum_{k\in\mathcal{N}\cup\mathcal{T}}\sum_{a'\in\mathcal{A}} \pi_j^{a'} p_{jk}^{a'} r_{jk}^{a'} \\
&+ \gamma^2 \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a \sum_{k\in\mathcal{N}\cup\mathcal{T}}\sum_{a'\in\mathcal{A}} \pi_j^{a'} p_{jk}^{a'} v^\pi(k) \\
&= \sum_{j\in\mathcal{T}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a + \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a \\
&+ \gamma \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a \sum_{k\in\mathcal{N}}\sum_{a'\in\mathcal{A}} \pi_j^{a'} p_{jk}^{a'} r_{jk}^{a'} + \gamma \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a \sum_{k\in\mathcal{T}}\sum_{a'\in\mathcal{A}} \pi_j^{a'} p_{jk}^{a'} r_{jk}^{a'} \\
&+ \gamma^2 \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a \sum_{k\in\mathcal{N}}\sum_{a'\in\mathcal{A}} \pi_j^{a'} p_{jk}^{a'} v^\pi(k) && \text{splitting } \mathcal{N} \text{ and } \mathcal{T}
\end{aligned}
$$

Similar to earlier, we have vectors, $\mathbf{h}$ and $\mathbf{m}$ with, $[\mathbf{h}]_i = \sum_{j\in\mathcal{T}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a$, $[\mathbf{m}]_i = \sum_{j\in\mathcal{N}}\sum_{a\in\mathcal{A}} \pi_i^a p_{ij}^a r_{ij}^a$, and Q is the true transition matrix of the Markov reward process induced by π and P, i.e., $[Q]_{ij} = \sum_a \pi_i^a p_{ij}^a$. The terms are not overloaded since the expectation over the true policy yields the same values. Then continuing from above, we have

$$
v^\pi(i) = [\mathbf{h}]_i + [\mathbf{m}]_i + \gamma Q[\mathbf{h}]_i + \gamma Q[\mathbf{m}]_i + \gamma^2 Q^2[\mathbf{h}]_i + \gamma^2 Q^2[\mathbf{m}]_i + \ldots \quad \text{unrolling } v^\pi(\mathcal{N}\cup\mathcal{T}) \tag{A.5}
$$

$$
= \left[\sum_{k=0}^{\infty} (\gamma Q)^k (\mathbf{m} + \mathbf{h})\right]_i \tag{A.6}
$$

$$
v^\pi(i) = \left[(I - \gamma Q)^{-1}(\mathbf{m} + \mathbf{h})\right]_i \tag{A.7}
$$

The existence of the limit and inverse are assured by Theorem A.1 in Sutton (1988).

The theorem is applicable here since $\lim_{k\to\infty}(\gamma Q)^k = 0$.

A.1.4 MDP Certainty-Equivalence Fixed-Point

Similar to the above subsection, for certainty-equivalence fixed-point, we consider a batch of data, $\mathcal{D}$, with the maximum-likelihood estimate (MLE) of the above quantities according to $\mathcal{D}$ given with a hat ($\hat{}$) on top of the quantity. The observed sets of non-terminal and terminal states in the batch are given by $\widehat{\mathcal{N}}$ and $\widehat{\mathcal{T}}$ respectively.

Then similar to above, we can derive the certainty-equivalence estimate of the value function according to the MLE of the policy and transition dynamics from the batch for a particular state, $i \forall \hat{\widehat{N}}$ is:

$$v^{\hat{\pi}}(i) = \left[(I - \gamma\widehat{Q})^{-1}(\hat{\mathbf{m}} + \hat{\mathbf{h}})\right]_i \tag{A.8}$$

This fixed-point is called the certainty-equivalence estimate (CEE) (Sutton, 1988) for an MDP. We note that MLE of the policy and transition dynamics according to the batch may not be representative of the true policy and transition dynamics. In that case, MDP-CEE (Equation (A.8)) is inaccurate with respect to Equation (A.7) due to policy and transition dynamics *sampling error*.

A.1.5 Policy Sampling Error Corrected MDP Certainty-Equivalence Fixed-Point

We now derive a new fixed-point, the *policy sampling error corrected MDP certainty-equivalence fixed-point*. This fixed-point corrects the policy sampling error that occurs in the value function given by Equation (A.8), making the estimation more accurate with respect to the true value function given by Equation (A.7).

We introduce the PSEC weight, $\hat{\rho}_i^a = \frac{\pi_i^a}{\hat{\pi}_i^a}$, with π being the policy that we are interested in evaluating and $\hat{\pi}$ being the MLE of the policy according to batch $\mathcal{D}$. $\hat{\rho}$ is then applied to the above quantities to introduce a slightly modified notation. In particular, $\hat{\rho}$ applied to $\widehat{Q}$ results in $[\widehat{U}]_{ij} = \sum_{a\in\hat{\mathcal{A}}_i} \hat{\rho}_i^a \hat{\pi}_i^a \hat{p}_{ij}^a$, and applied to vectors $\hat{\mathbf{h}}$ and $\hat{\mathbf{m}}$ results in $[\hat{\mathbf{l}}]_i = \sum_{j\in\mathcal{T}} \sum_{a\in\hat{\mathcal{A}}_i} \hat{\rho}_i^a \hat{\pi}_i^a \hat{p}_{ij}^a \hat{r}_{ij}^a$ and $[\hat{\mathbf{o}}]_i = \sum_{j\in\mathcal{N}} \sum_{a\in\hat{\mathcal{A}}_i} \hat{\rho}_i^a \hat{\pi}_i^a \hat{p}_{ij}^a \hat{r}_{ij}^a$ respectively. After simplification, we have $[\widehat{U}]_{ij} = \sum_{a\in\hat{\mathcal{A}}_i} \pi_i^a \hat{p}_{ij}^a$, $[\hat{\mathbf{l}}]_i = \sum_{j\in\mathcal{T}} \sum_{a\in\hat{\mathcal{A}}_i} \pi_i^a \hat{p}_{ij}^a \hat{r}_{ij}^a$, and $[\hat{\mathbf{o}}]_i = \sum_{j\in\mathcal{N}} \sum_{a\in\hat{\mathcal{A}}_i} \pi_i^a \hat{p}_{ij}^a \hat{r}_{ij}^a$.

Using these policy sampling error corrected quantities, we can derive the fixed-point for true policy, π, in a similar manner as earlier:

$$v^{\pi}(i) = \left[(I - \gamma \widehat{U})^{-1}(\hat{\mathbf{o}} + \hat{\mathbf{l}})\right]_i \tag{A.9}$$

In computing this new fixed-point, we have corrected for the policy sampling error, resulting in a more accurate estimation of Equation (A.7) than Equation (A.8). Now, the value function is computed for the true policy that we are interested in evaluating, π.

A.2 Extended Empirical Description

In this appendix we provide additional details for our empirical evaluation.

A.2.1 Gridworld

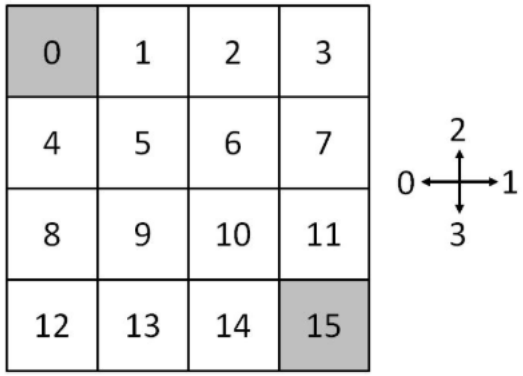


Figure A.1: The Gridworld environment. Start at top left, bottom right is terminal state, discrete action space consists of the cardinal directions, and discrete state space is the location in the grid. This specific image was taken from this link.

This domain is a 4×4 grid, where an agent starts at $(0, 0)$ and tries to navigate to $(3, 3)$. The states are the discrete positions in the grid and actions are the 4 cardinal directions. The reward function is 100 for reaching $(3, 3)$, -10 for reaching $(1, 1)$, 1 for reaching $(1, 3)$, and -1 for reaching all other states. If an agent takes an action that hits a wall, the agent stays in the same location. The transition dynamics are controlled by a parameter, p, where with probability p, an agent takes the intended action, else it takes an adjacent action with

probability $(1-p)/2$. All policies use a softmax action selection distribution with value θ_{sa}, for each state, s, and action a. The probability of taking action a in state s is given by:

$$\pi(a|s) = \frac{e^{\theta_{sa}}}{\sum_{a' \in \mathcal{A}} e^{\theta_{sa'}}}$$

In the on-policy experiments, the evaluation and behavior policies were equiprobable policies in each cardinal direction. In the off-policy experiments, the evaluation policy was such that each θ was generated from a standard normal distribution and behavior policy was the equiprobable policy.

For the comparisons of batch linear PSEC-TD(0) and TD(0), we conducted a parameter sweep of the learning rates for the varying batch sizes. The parameter sweep was over: $\{5e^{-3}, 1e^{-3}, 5e^{-2}, 1e^{-2}, 5e^{-1}\}$. We used a value function convergence threshold of $1e^{-10}$. For PSEC-LSTD and LSTD, we stabilized the matrix, A, before inverting it by adding ϵI to the computed A. We conducted a parameter sweep over the following: $\epsilon \in \{1e^{-6}, 1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}, 1e^{-1}\}$.

A.2.2 CartPole

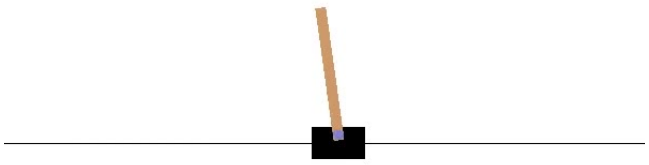


Figure A.2: CartPole-v0 from OpenAI Gym (Brockman et al., 2016)

In this domain, the goal of the agent is to balance a pole for as long as possible. We trained our behavior policy using REINFORCE (Williams, 1992) with the Adam optimizer (Kingma and Ba, 2014) with learning rate $3e^{-4}$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The behavior policy mapped raw state features to a softmax distribution over actions. The policy was a neural network

with 2 hidden layers with 16 neurons each, and used the tanh activation function and was initialized with Xavier initialization (Glorot and Bengio, 2010).

The value function used by all algorithms was initialized by Xavier initialization and used the tanh activation function, and was trained using semi-gradient TD (Sutton and Barto, 2018). We used a convergence threshold of 0.1.

The PSEC policy was initialized by Xavier initialization and used the tanh activation function. PSEC-TD sweeped over the following learning rates $\alpha \in \{0.1 \times 2.0^j | j = -7, -6, ...1, 2\}$. It used a validation set of 10% the size of the batch size. It used an L2 regularization of $2e^{-2}$. More details can be found in Chapter 4.

A.2.3 InvertedPendulum

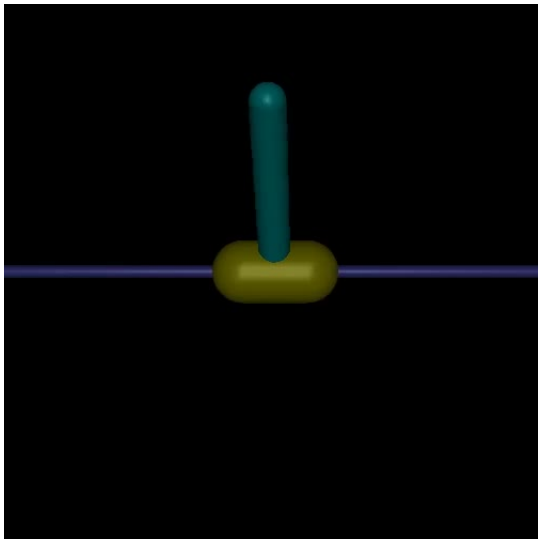


Figure A.3: InvertedPendulum-v2 from OpenAI Gym and MuJoCo (Brockman et al., 2016; Todorov et al., 2012)

In this domain, the goal of the agent is to balance a pole for as long as possible. We trained our behavior policy using PPO (Schulman et al., 2017) with the default settings found on Gym (Brockman et al., 2016). The policy was a neural network with 2 hidden layers with 64 neurons each, and used the tanh activation function and was initialized with Xavier initialization (Glorot and Bengio, 2010). It mapped state features to an output vector that represented the mean vector of a Gaussian distribution. This mapping along with a separate parameter set representing the log standard deviation of each element in the output vector,

make up the policy. The policy was trained by minimizing the following loss function:

$$\mathcal{L} = \sum_{i=1}^{m} 0.5((a_i - \mu(s_i))/e^{\sigma})^2 + \sigma$$

where m are the number of state-action training examples, a_i is the action vector of the i^{th} example, $\mu(s_i)$ is the mean vector outputted by the neural network of the Gaussian distribution for state s_i, and σ is the the seperate parameter representing the log standard deviation of each element in the output vector, $\mu(s_i)$.

The value function used by all algorithms was initialized by Xavier initialization and used the tanh activation function, and was trained using semi-gradient TD (Sutton and Barto, 2018). We used a convergence threshold of 0.1.

The PSEC policy was initialized by Xavier initialization and used the tanh activation function. PSEC-TD sweeped over the following learning rates $\alpha \in \{0.1 \times 2.0^j | j = -8, -6, ..., 1, 2\}$. It used a validation set of 20% the size of the batch size. More details can be found in Chapter 4.

Bibliography

Kristopher De Asis, J. Fernando Hernandez-Garcia, G. Zacharias Holland, and Richard S. Sutton. Multi-step reinforcement learning: A unifying algorithm, 2017.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*. 1995.

Richard Ernest Bellman. *Dynamic Programming*. Dover Publications, Inc., USA, 2003. ISBN 0486428095.

Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall, Inc., USA, 1987. ISBN 0132215810.

Steven Bradtke and Andrew Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 03 1996. doi: 10.1007/BF00114723.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *ArXiv*, abs/1606.01540, 2016.

Bernard Delyon and Franois Portier. Integral approximation by kernel smoothing. *Bernoulli*, 22(4):2177?2208, Nov 2016. ISSN 1350-7265. doi: 10.3150/15-bej725. URL http://dx.doi.org/10.3150/15-BEJ725.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. *CoRR*, abs/1802.03493, 2018. URL http://arxiv.org/abs/1802.03493.

S. Gerschgorin. Uber die abgrenzung der eigenwerte einer matrix. *Izvestija Akademii Nauk SSSR, Serija Matematika*, 7(3):749–754, 1931.

Sina Ghiassian, Andrew Patterson, Martha White, Richard S. Sutton, and Adam White. Online off-policy prediction, 2018.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016. URL http://arxiv.org/abs/1610.00633.

Josiah Hanna and Peter Stone. Reducing sampling error in the monte carlo policy gradient estimator. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, May 2019.

Josiah Hanna, Scott Niekum, and Peter Stone. Importance sampling policy evaluation with an estimated behavior policy. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, June 2019.

Masayuki Henmi, Ryo Yoshida, and Shinto Eguchi. Importance sampling via the estimated sampler. *Biometrika*, 94(4):985–991, 2007. URL https://EconPapers.repec.org/RePEc:oup:biomet:v:94:y:2007:i:4:p:985-991.

Keisuke Hirano, Guido W. Imbens, and Geert Ridder. Efficient estimation of average treatment effects using the estimated propensity score. *Econometrica*, 71(4):1161–1189, 2003. doi: 10.1111/1468-0262.00442. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0262.00442.

John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Jens Kober, J. Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32:1238–1274, 09 2013. doi: 10.1177/0278364913495721.

Vijay R. Konda and John N. Tsitsiklis. Actor-critic algorithms. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1008–1014. MIT Press, 2000. URL http://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf.

Lihong Li, Rémi Munos, and Csaba Szepesvári. Toward minimax off-policy value estimation. In *AISTATS*, 2015.

A Rupam Mahmood, Huizhen Yu, Martha White, and Richard S Sutton. Emphatic temporal-difference learning. *arXiv preprint arXiv:1507.01569*, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

Yusuke Narita, Shota Yasui, and Kohei Yata. Efficient counterfactual learning from bandit feedback. *CoRR*, abs/1809.03084, 2018. URL http://arxiv.org/abs/1809.03084.

Yangchen Pan, Adam M. White, and Martha White. Accelerated gradient temporal difference learning. *CoRR*, abs/1611.09328, 2016. URL http://arxiv.org/abs/1611.09328.

Doina Precup, Richard S. Sutton, and Satinder Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 759–766, 2000a.

Doina Precup, Richard S. Sutton, and Satinder P. Singh. Eligibility traces for off-policy policy evaluation. In *ICML*, 2000b.

Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978. ISSN 00251909, 15265501. URL http://www.jstor.org/stable/2630487.

Paul R. Rosenbaum. Model-based direct adjustment. *Journal of the American Statistical Association*, 82(398):387–394, 1987. ISSN 01621459. URL http://www.jstor.org/stable/2289440.

G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S Sutton, Hamid Reza Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 993–1000, 2009.

Philip S. Thomas. A notation for markov decision processes. *CoRR*, abs/1512.09075, 2015. URL http://arxiv.org/abs/1512.09075.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

H. van Seijen, H. van Hasselt, S. Whiteson, and M. Wiering. A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 177–184, March 2009. doi: 10.1109/ADPRL.2009.4927542.

B. L. WELCH. THE GENERALIZATION OF ?STUDENT'S? PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED. *Biometrika*, 34(1-2):28–35, 01 1947. ISSN 0006-3444. doi: 10.1093/biomet/34.1-2.28. URL https://doi.org/10.1093/biomet/34.1-2.28.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3?4):229?256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

Yuan Xie, Boyi Liu, Qiang Liu, Zhaoran Wang, Yuan Zhou, and Jian Peng. Off-policy evaluation and learning from logged bandit feedback: Error reduction via surrogate policy. *CoRR*, abs/1808.00232, 2018. URL http://arxiv.org/abs/1808.00232.

Long Yang, Minhao Shi, Qian Zheng, Wenjia Meng, and Gang Pan. A unified approach for multi-step temporal-difference learning with eligibility traces in reinforcement learning. *CoRR*, abs/1802.03171, 2018. URL http://arxiv.org/abs/1802.03171.