Temporal Regularization in Reinforcement Learning

Pierre Thodoroff

Computer Science
McGill University, Montreal

March 15, 2020

Abstract

Reinforcement Learning is a widely used framework for sequential decision making in many domains such as robotics and video games. However, its use in the real-world remains limited due, in part, to the high variance of value function estimates, leading to poor sample complexity. Regularization is a cornerstone tool of modern Machine Learning designed to reduce variance of the target estimates by adding inductive bias to the model. Temporal Regularization induces the following inductive bias: temporally close data points should have similar predictions.

In this thesis, we introduce and analyze the concept of Temporal Regularization in Reinforcement Learning. First, we demonstrate how many works in Reinforcement Learning can be interpreted as Temporal Regularization. Then, we design a prototypical version of Temporal Regularization in Model-Free Reinforcement Learning and analyze it, both theoretically and experimentally. Finally, we propose a method capable of learning which states can be smoothed temporally without introducing significant bias.

Résumé

L'apprentissage par renforcement est un modèle d'apprentissage fréquemment utilisé pour modéliser les problèmes de décisions séquentielles comme la robotique et les jeux vidéos. Cependant son utilisation dans le monde réel reste limité à cause, en partie, de la forte variance de la fonction de valeur des états. La régularisation est l'un des piliers de l'apprentissage automatique moderne, conçu pour réduire la variance des estimations cibles en ajoutant au modèle un biais inductif. La régularisation temporelle induit le biais inductif suivant: les points de données proches dans le temps doivent avoir des estimations similaires.

Dans cette thèse, nous introduisons et analysons le concept de régularisation temporelle dans l'apprentissage par renforcement. Tout d'abord, nous montrons combien d'œuvres en apprentissage par renforcement peuvent être interprétées comme une régularisation temporelle. Ensuite, nous concevons une version prototype de la régularisation temporelle dans l'apprentissage par renforcement et nous l'analysons théoriquement et expérimentalement. Enfin, nous proposons une méthode visant à savoir quels états peuvent être lissés temporellement sans introduire de biais significatif.

Acknowledgements

I am deeply grateful to my supervisor Joelle Pineau for her continuous support, encouragement, and insightful discussion throughout my Master. I thank all my co-authors with whom I worked on the ideas presented in this thesis: Doina Precup, Audrey Durand, Lucas Gaccia, and Nishanth Anand. I would also like to thank all the persons that I interacted, debated and exchanged ideas throughout my thesis, in particular, Pierre-Luc Bacon, Harsh Sajita, Joshua Romoff, Ahmed Touati, Prakash Panagden, Philip Amortilla, and Guillaume Rabusseau. Finally, all of this would not have been possible without the continuous support of my partner Maria Vedechkina.

Contribution of Authors

This thesis introduces the concept of Temporal Regularization in Reinforcement Learning.

- Chapter 1 and 2 are written for this thesis and introduce the basic concepts in Reinforcement Learning and Markov Chains.
- Chapter 3 presents the concept of Temporal Regularization in Reinforcement Learning. Audrey Durant provided invaluable help on the correctness, writing, and clarity of its content. Most of the material presented originates from the paper presented at NeurIPS (Thodoroff, Durand, et al., 2018).
- Chapter 4 introduces Recurrent Value Functions and is joint work with Nishanth Anand and Lucas Gaccia. Both Nishanth and Lucas participated in the design of the algorithm and the experiments. This material is going to be presented at RLDM (Thodoroff, Anand, et al., 2019).

Contents

List of Figures

List of Algorithms

1

Introduction

Since the advent of computers, designing machines capable of displaying human-like intelligence has been the ultimate goal as displayed by the design of the Turing test in 1953. Our intelligence in the real world is reflected by the *decisions* we make at every instant, which is why in the 1950s researcher's started designing mathematical frameworks to solve the problem of *sequential decision making*. To this date, the two most successful paradigms for sequential decision making are Optimal Control (Lee and Markus, 1967; Zhou, Doyle, Glover, et al., 1996) and Trial And Error (Klopf, 1982; R. S. Sutton and Barto, 1998).

Optimal Control attempts to find an *optimal policy* for a dynamical system according to some *optimality criterion*. Richard Bellman pioneered the field with the concepts of Dynamic Programming and the Bellman equation(Bellman et al., 1954). The Bellman equation remains at the center of most Reinforcement Learning(RL) algorithm nowadays and many other fields such as Economic theory. However, most of the Optimal Control methods relied on the knowledge of the dynamics of the system.

In contrast, Trial And Error took its essence from animals and relied on experiences. The main idea is to use experiences from the real world to drive the behavior towards *desirable states*. This thread focused more on sampled experience from the environment to learn the optimal behavior. When referring to Reinforcement Learning, the community often refers to the latter. In the Trial And Error thread, the core idea

is to estimate the *value* of each state of the world and direct the agent towards good state *based on a stream of experience*. The value of a state describes the expected future return from this state onwards, under the current *policy*. The goal of RL is two-fold, one learning a value function describing how *good* each state is, two learning to control an agent towards those *good* states.

In RL, one can also consider 2 paradigms: model-free (R. S. Sutton and Barto, 1998) or model-based (R. S. Sutton, 1990). Model-based RL attempts to model the environment and behave optimally according to that model. In contrast, model-free RL solely relies on samples obtained to derive its behavior. Although model-based RL is intuitively a more appealing solution, learning a model of the environment can be complicated. In most of the recent successes of RL, model-free RL has shown to be more sample efficient and practical than its counter-part model-based (Silver et al., 2016; Schulman, Wolski, et al., 2017; Mnih, Kavukcuoglu, Silver, Graves, et al., 2013). In this thesis, we focus on model-free RL.

In recent years, model-free RL has shown promises in many domains such as such as robotics (Kober, Bagnell, and Peters, 2013; Abbeel, Coates, and Ng, 2010) and video games (Vinyals et al., 2017; Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Mnih, Badia, et al., 2016). It is also used in some real-life applications such as hydro control (Grinberg, Precup, and Gendreau, 2014) and power grid management (François-Lavet et al., 2016). However, model-free RL use in the real-world remains limited due, in part, to the high variance of value function estimates (Greensmith, Bartlett, and Baxter, 2004), leading to poor sample complexity (Gläscher et al., 2010; Kakade et al., 2003). This variance can be due to various phenomena such as randomness in data collection, effects of initial conditions, the complexity of learner function class, hyper-parameter configuration, or sparsity of the reward signal (Henderson et al., 2017). Those phenomena are exacerbated by the noisy conditions of the real-world (Fox, Pakman, and Tishby, 2015; Pendrith, 1994). Real-world applications remain challenging as they often involve noisy data such as sensor noise and partially

observable environments.

Regularization is a cornerstone of modern Machine Learning (Engl, Hanke, and Neubauer, 1996; Poggio, Torre, and Koch, 1987; Groetsch, 1984). At its core, regularization is a tool designed to reduce the variance of estimates by introducing some inductive bias. The inductive bias induced is decided by the form of regularization applied to the model. For example, l_2 regularization on a linear regression model pushes parameters to small values (Hoerl and Kennard, 1970). The underlying assumption is *data points that are close to each other in feature space should have similar predictions*. In sequential domains, we have the opportunity to consider a new kind of regularization. Temporal regularization assumes that states that are close to each other *temporally* have similar value. The term Temporal Regularization has not been used in the RL literature; however, many works can be seen as such (Z. Xu et al., 2017; Baird, 1995). The concept has also been explored in several sequential fields of supervised learning such as Time-Series prediction (Yu, Rao, and Dhillon, 2016), Dynamic network visualization (K. S. Xu, Kliger, and Hero, 2013) or image reconstruction (Asif et al., 2013).

In this thesis, we consider temporally regularizing value estimates in Reinforcement Learning. Many of the concepts introduced can be used on actions as well. Effectively, value-based temporal regularization considers smoothing over the trajectory, whereby the estimate of the value function at one state is assumed to be related to the value function at the state(s) that typically occur before it in trajectories. This structure arises naturally out of the fact that the value at each state is estimated using the Bellman equation. The standard Bellman equation clearly defines the dependency between value estimates. In Temporal Regularization, we amplify this dependency by making each state depend more strongly on estimates of *previous* states as opposed to multi-step that considers future states. However, smoothing along the trajectory can result in bias when the value function changes dramatically through the trajectory (non-stationarity). This bias could be a problem if the environment encounters sharp

changes, such as falling off a cliff, and the estimates are smoothed. The motivation behind the second work of this thesis is to design a method capable of learning when to temporally smooth estimates such as to minimize the bias induced.

Instead of modifying the target, we estimate the value functions directly using exponential smoothing. We propose Recurrent Value Functions (RVFs): an exponential smoothing of the value function. The value function of the current state is defined as an exponential averaging of the values of states visited along the trajectory where the value function of past states is summarized by the previous RVF. To alleviate the "falling off a cliff" issue, we propose to use exponential smoothing on value functions using a trainable state-dependent emphasis function which controls the smoothing coefficients. The emphasis function identifies important states in the environment. An important state can be defined as one where *its value differs significantly from the previous values along the trajectory*. For example, when falling off a cliff, the value estimate changes dramatically, making states around the cliff more salient.

In chapter 2, we introduce useful Markov chain concepts. The third chapter introduces the basic concepts of Reinforcement Learning. In chapter 4, we propose a class of temporally regularized value function estimates. We discuss properties of these estimates, based on notions from Markov chains, under the policy evaluation setting and extend the notion to the control case. Our experiments show that temporal regularization effectively reduces variance and estimation error of value estimates in discrete and continuous MDPs. We then present some positive results in discrete control environments and some negative results in continuous control using temporal regularization. In particular, we hypothesize that lack of state-dependent smoothing coefficient can hinder learning. Finally, chapter 5 introduces the concept of Recurrent Value Functions(RVFs). RVFs estimate the value function of a state by exponentially smoothing the value estimates along the trajectory. RVF formulation leads to a natural way of learning an emphasis function which mitigates the bias induced by smoothing. We provide an asymptotic convergence proof in tabular settings by leveraging the

literature on asynchronous stochastic approximation (Tsitsiklis, 1994). Finally, we perform a set of experiments to demonstrate the robustness of RVFs with respect to noise in continuous control tasks and provide a qualitative analysis of the learned emphasis function which provides interpretable insights into the structure of the solution.

2

Markov Chains

2.1 Discrete-time Markov chains

We begin by introducing discrete Markov chain concepts that will be used to study the properties of temporally regularized MDPs. In this thesis, we focus on discrete Markov chains, however the concepts can be extended to the continuous case.

Discrete Markov chain are stochastic models representing sequences of random variable satisfying the Markov property. Formally, we define a discrete-time Markov chain (Norris, 1998; Levin and Peres, 2008; Brémaud, 2013) with finite state space $\mathcal{S}$ by a sequence of $|\mathcal{S}|-$valued random variable $X_0, X_1, ...$ and a transition function $\mathcal{P} : \mathcal{S} \times \mathcal{S} \mapsto [0, 1]$. The sequence of random variable needs to satisfy the Markov property:

Definition 1. *(Markov property (Markov, 1954)). A stochastic process satisfies the Markov property if:*

$$\mathcal{P}(X_{n+1} = j | X_n = i, X_{n-1} = k, ...) = \mathcal{P}(X_{n+1} = j | X_n = i). \tag{2.1}$$

Intuitively, this means that the probability of moving to the next states is based solely on its present state and not its history. One of the most studied Markov chains considers the property of randomly walking on a chain as described in figure (2.1).

Discrete time Markov chains can also be represented in matrix form. The transition

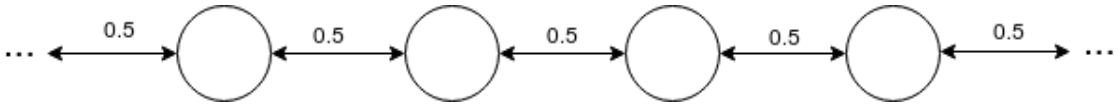


Figure 2.1: Random walk

function can be represented as a $|\mathcal{S}|\times|\mathcal{S}|$ matrix P such that $P_{ij} = \mathcal{P}(X_{n+1} = j|X_n = i)$. Studying Markov chains from an algebraic perspective can sometimes simplify the analysis.

We now define some useful fundamental properties of Markov chains. If every state is accessible from every other, the chain(or its transition matrix) is said to be irreducible.

Definition 2. *(Irreducible chain (Norris, 1998)). A chain is said to be irreducible if* $\forall i, j \in \mathcal{S}$ *there exists* $n \in \mathbb{N}$ *such that:*

$$P(X_n = j|X_0 = i) > 0. \tag{2.2}$$

Definition 3. *(Period of a state (Norris, 1998)). The period of a state is the greatest common divisor of the set* $n \in \mathbb{N} : \quad P(X_n = i|X_0 = i) > 0$. *The chain is defined as aperiodic if every state has period 1.*

Definition 4. *(Recurrent and Transient state (Norris, 1998)). Let* T_i *define the hitting time of state i such that:*

$$T_i = inf(n \geq 1 : X_n = i|X_0 = i). \tag{2.3}$$

We define a transient state if T_i *is not finite. A state is called recurrent if it is not transient.*

Definition 5. *(Ergodic chain (Norris, 1998)). A chain is defined as ergodic if it is positive recurrent and aperiodic.*

Throughout this thesis, we make the following mild assumption on the Markov chain:

Assumption 1. *P is ergodic*

Most of the Reinforcement learning theory relies on this assumption. However, some works considers the case when the chain is not ergodic (Leike, 2016).

2.2 Stationary distribution

It is often interesting to study the properties of Markov chains in the limit. We define the stationary distribution μ_i as the *proportion of time* spend in each state $i \in \mathcal{S}$.

Definition 6. *(Stationary distribution (Norris, 1998)). Assuming that P is ergodic, P has a unique stationary distribution μ that satisfies:*

$$\begin{aligned} \mu &= \mu P, \\ \sum_i \mu_i &= 1. \end{aligned} \tag{2.4}$$

There exists many different metrics used to define distance's from stationary distribution (Levin and Peres, 2008). One common metric in discrete Markov chains can be defined as follows:

$$d_t(P) = \left\| P^t \mathbb{1} - \mu \right\|_\infty, \tag{2.5}$$

where $\mathbb{1}$ is a vector of one's.

2.3 Detailed Balance

The concept of detailed balance originated from physics. It is used to study the behavior of systems in the limit at *equilibrium*.

Definition 7 (Detailed balance (Kemeny and Snell, 1976))**.** *Let P be an irreducible Markov chain with invariant stationary distribution μ. μ_i defines the ith element of μ. A chain is said to satisfy detailed balance if and only if*

$$\mu_i P_{ij} = \mu_j P_{ji} \qquad \forall i, j \in \mathcal{S}. \tag{2.6}$$

Intuitively, this means that if we start the chain in a stationary distribution, the amount of probability that flows from i to j is equal to the one from j to i. In other words, the system must be at equilibrium. An intuitive example of a physical system not satisfying detailed balance is a snow flake in a coffee.

Remark. *If a chain satisfies detailed balance, it is called reversible.*

2.4 Mixing Time

In Markov chains theory, one of the main challenges is to study the mixing time of the chain (Levin and Peres, 2008). The mixing time corresponds to the time needed for the chain to be *close* to its stationary distribution μ. More formally, it can be defined as:

$$t_{mix}(\epsilon) = \min\{t : d_t(P) < \epsilon\}, \tag{2.7}$$

where $d_t(P)$ can be defined as in (2.5).

When the chain is reversible, it is possible to estimate and bound the mixing time relatively efficiently (Diaconis and Stroock, 1991). Indeed, many chains do not satisfy this detailed balance property. In this one case it is possible to use a different, but related, chain called the reversal Markov chain to infer mixing time bounds (Fill et al., 1991; K.-M. Chung et al., 2012).

2.5 Reversal Markov Chains

The reversal Markov chain $\widetilde{P}$ can be interpreted as the Markov chain P with time running backward. It is a key concept used to define convergence and bias induced by temporal regularization later in this thesis.

Definition 8 (Reversal Markov chain (Kemeny and Snell, 1976))**.** *Let $\widetilde{P}$ the reversal*

Markov chain of P be defined as:

$$\widetilde{P_{ij}} = \frac{\mu_j P_{ji}}{\mu_i} \qquad \forall i, j \in \mathcal{S}. \tag{2.8}$$

As an example, assuming a Markov chain P has a uniform stationary distribution, if a transition is highly irreversible, like falling off a cliff ($P_{ij} \# P_{ji}$), the difference between the forward and the backward chain in that state will be high. We now introduce some properties of reversal Markov chains that will be used later in the thesis.

Remark. *If P is irreducible with invariant distribution μ, then $\widetilde{P}$ is also irreducible with invariant distribution μ.*

Remark. *If P is reversible than $P = \widetilde{P}$.*

Furthermore, both P and $\widetilde{P}$ have the same stationary distribution and so does any convex combination of them. We now prove a lemma used later in this thesis.

Lemma 1. *P and $(1-\beta)P + \beta\widetilde{P}$ have the same stationary distribution μ* $\quad \forall \beta \in [0, 1]$.

Proof.

$$\begin{aligned} \mu((1-\beta)P + \beta\widetilde{P}) &= (1-\beta)\mu P + \beta\mu\widetilde{P} \\ &= (1-\beta)\mu + \beta\mu \\ &= \mu. \end{aligned} \tag{2.9}$$

□

3

Reinforcement Learning

In science, mathematical frameworks are used to study the behavior of objects. In physics, for example, Newton's laws laid the foundation of classical mechanics used to describe the motion of macroscopic objects. In this thesis, we are interested in the problem of sequential decision making. The most popular mathematical framework used to study sequential decision making is called the Markov Decision Process (Bellman, 1957)(MDP). The underlying core assumption is the Markovian assumption on the state space. MDP assumes the state space is fully observable and the future is independent of the past conditioned on the current state. More formally, it is defined as:

$$p(s_{t+1}|s_t, s_{t-1}, ..., s_0) = p(s_{t+1}|s_t) \tag{3.1}$$

Sequential decision making differs from supervised learning in several ways. Supervised learning is a set of models designed to predict an output based on IID data. However, in reinforcement learning our prediction/decision often impact the distribution of the data. For example, choosing to do turn right at an intersection will siginificantly change the distribution of future states.

3.1 Markov Decision Process

We now formally introduce the concept used in Markov Decision Process.

Definition 9 (Markov Decision Process ((Puterman, 1994))). *A Markov Decision Process (MDP) is defined as a tuple $(\mathcal{S}, \mathcal{P}, r)$ where:*

- $\mathcal{S}$ *is a discrete set of states.*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *is a transition function.*
- $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ *is a reward function.*

On each round t, the learner observes current state $s_t \in \mathcal{S}$ and selects action $a_t \in \mathcal{A}$, after which it receives reward $r_t = r(s_t, a_t)$ and moves to new state $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$. We define a stationary policy π as a probability distribution over actions conditioned on states $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, such that $a_t \sim \pi(\cdot|s_t)$.

3.2 Policy Evaluation

For policy evaluation, given a policy π, the goal is to find the associated value function of that policy V^π. When performing policy evaluation in the discounted case, the goal is to estimate the discounted expected return of policy π at a state $s \in \mathcal{S}$, $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0 = s]$, with discount factor $\gamma \in [0, 1)$. This can be written in matrix form as:

$$v^\pi = \sum_{i=0}^{\infty} \gamma^i (P^\pi)^i r, \tag{3.2}$$

where P^π denotes the $|\mathcal{S}| \times |\mathcal{S}|$ transition matrix under policy π, v^π is the state values column-vector, and r is the reward column-vector. The actions do not appear in the equations explicitly as the policy has been coupled inside P^π. The matrix P^π also defines a Markov chain. In practice, we often do not have access to transitions and rewards directly. Instead, we sample tuples (s, s', r) from the environment and use those to estimate in expectation the discounted expected cumulative return for a state. The most straightforward way to estimate v would be to collect tuples (s, s', r) and average the discounted reward. However, this often suffers from high variance (Kearns

and Singh, 2000). By unrolling the discounted sum of reward, it is possible to obtain a recursive form based on V:

$$\begin{aligned} V^{\pi}(s_0) &= \mathop{\mathbb{E}}_{\pi}[r(s_0, \pi(s_0)) + \sum_{t=1}^{\infty} \gamma^t r_{t+1} | s_0 = s] \\ &= \mathop{\mathbb{E}}_{\pi}[r(s_0, \pi(s_0)) + \gamma \mathop{\mathbb{E}}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_1] | s_0 = s] \\ &= \mathop{\mathbb{E}}_{\pi}[r(s_0, \pi(s_0)) + \gamma v^{\pi}(s_1) | s_0 = s]. \end{aligned} \tag{3.3}$$

This can also be rewritten as a linear system of equation and solved using standard algebra methods:

$$\begin{aligned} v^{\pi} &= r + \gamma P^{\pi} v^{\pi} \\ &\equiv (I - P^{\pi}) v^{\pi} = r. \end{aligned} \tag{3.4}$$

However, inverting a matrix can be costly, unstable and as mentioned earlier in practice we often do not have access to P^{π}, r directly. This suggests that using iterative method may be more suitable for reinforcement learning.

3.2.1 Bellman Operator

We consider the operator-theoretic point of view by defining the following operator:

Definition 10. *The Bellman operator $\mathcal{T}^{\pi}$ has a unique fixed point v^{π} where:*

$$\mathcal{T}^{\pi} v = r + \gamma P^{\pi} v. \tag{3.5}$$

In order to show this, we use Banach Fixed point theorem stating that:

Theorem 1 (Banach Fixed Point Theorem (Banach, 1922))**.** *Let U be a Banach space: if $\mathcal{T}U \to U$ is a contraction mapping then:*

- *There exists a unique fixed point v^* such that $\mathcal{T} v^* = v^*$.*
- *$lim_{t \to \infty} \mathcal{T}^t v = v^*$.*

As we saw in the previous section in equation 3.4 v^π is a fixed point. We can prove its unicity and the convergence of the operator by proving that $\mathcal{T}^\pi$ is a contraction. In this thesis, unless stated otherwise, the norm considered is the infinity norm.

Lemma 2. *$\mathcal{T}^\pi$ is a contraction with a contraction factor of γ*

Proof.

$$\begin{aligned} \|\mathcal{T}^\pi u - \mathcal{T}^\pi v\| &= \|r + \gamma P^\pi u - (r + \gamma P^\pi v)\| \\ &= \|\gamma P^\pi (u - v)\| \\ &= \gamma \|u - v\| . \end{aligned} \tag{3.6}$$

□

Algorithm 1 shows an example of a stochastic version of equation 3.5.

Algorithm 1 Policy evaluation

1: Input: π, γ
2: **for all** steps **do**
3: Choose $a \sim \pi(\mathcal{S})$
4: Take action a, observe $r(s), s'$
5: $V(s) = r(s) + \gamma V(s')$
6: **end for**

It is also possible to study the stochastic version of this operator using techniques from dynamical system and ordinary differential equations (Borkar and Meyn, 2000).

3.2.2 Temporal Difference

At each step of the bellman operator, the previous estimate v_t^π at time step t is forgotten.

$$v_{t+1} = r + \gamma P v_t. \tag{3.7}$$

Temporal difference attempts to exploit previous estimates by averaging them such that $v_{t+1} = \alpha \sum_{i=0}^{t+1} (1-\alpha)^{t-i} v_i$.

$$\begin{aligned} v_{t+1} &= \alpha(r + \gamma P v_t) + (1-\alpha) v_t \\ &= \alpha(r + \gamma P v_t) + (1-\alpha)[\alpha(r + \gamma P v_{t-1}) + (1-\alpha) v_{t-1}]. \end{aligned} \tag{3.8}$$

where α is the learning rate(averaging coefficient). This gives rise to the temporal difference algorithm:

$$\begin{aligned} v_{t+1} &= \alpha(r + \gamma P v_t) + (1-\alpha) v_t \\ &= \alpha(r + \gamma P v_t - v_t) + v_t. \end{aligned} \tag{3.9}$$

Definition 11 (Temporal Difference (R. S. Sutton, 1988))**.** *The temporal difference operator $\mathcal{T}_\alpha$ parametrized by α can be written as:*

$$\mathcal{T}_\alpha v = (1-\alpha) v + \alpha(r + \gamma P v). \tag{3.10}$$

A stochastic version of temporal difference can be found in algorithm 2.

Algorithm 2 Temporal Difference (R. S. Sutton, 1984)

1: Input: π, α, γ
2: **for all** steps **do**
3: Choose $a \sim \pi(\mathcal{S})$
4: Take action a, observe $r(s), s'$
5: $V(s) = V(s) + \alpha(r(s) + \gamma V(s') - V(s))$
6: **end for**

Bootstrapping on previous estimates may introduce bias depending on how well v is estimated. Several papers attempt to characterize this bias in various ways (Kearns and Singh, 2000; R. S. Sutton and Singh, n.d.). Methods bootstrapping on previous estimates are also called semi-iterative methods (Varga, 2009) in the field of iterative methods. It would be interesting to examine algorithms like Chebyshev semi-iterative method (Golub and Varga, 1961) that attempts to find optimal α's using Chebyshev polynomials. There might exist interesting connections with meta-learning algorithms.

3.2.3 Lambda Return

In the previous sections, we defined algorithms that bootstrap on the next value, to reduce variance, instead of looking at the rewards. Lambda return (R. S. Sutton, 1984) generalizes this intuition by bootstrapping on all future values, not just the direct next one. This is done by first unrolling the bellman updates, yielding N-step return of the form:

Definition 12 (N-step return)**.**

$$\mathcal{T}_n = \sum_{i=0}^{n} \gamma^i r + \gamma^{n+1} P v. \tag{3.11}$$

Instead of choosing a specific N, lambda-return exponentially averages through all the N-step return:

Definition 13 (Lambda-return)**.**

$$\mathcal{T}^{(\lambda)} = (1-\lambda) \sum_{i=0}^{\infty} \lambda^i \mathcal{T}_i, \tag{3.12}$$

where λ is the averaging coefficient. Varying λ yields Monte Carlo on one side ($\lambda \to 1$) and TD(0) on the other ($\lambda \to 0$). Lambda-return can be implemented efficiently in an online fashion using eligibility traces (R. S. Sutton, 1984; Singh and R. S. Sutton, 1996; Precup, 2000).

However, this algorithm is biased if the trace is used online. This is due to the

Algorithm 3 Temporal Difference with eligibility traces (R. S. Sutton, 1984)

1: Input: $\pi, \alpha, \gamma, \lambda$
2: **for all** steps **do**
3: Choose $a \sim \pi(\mathcal{S})$
4: Take action a, observe $r(s), s'$
5: $e(t) = \mathbb{1}_s + \gamma\lambda e(t-1)$
6: $V = V + \alpha(r(s) + \gamma V(s') - V(s))e(t)$
7: **end for**

fact that when an update is done, the trace becomes biased as the distribution with respect to the new parameters would have been different. True online TD (Seijen and

R. Sutton, 2014) solves this issue by accounting for those changes. This problem is similar to the one encountered by real time recurrent learning (Williams and Zipser, 1995). It would be interesting to study if True Online TD can be extended to the non-linear settings.

3.2.4 Convergence using Stochastic Approximation

Convergence in the stochastic setting is usually proven by casting the learning algorithm as a stochastic approximation (Tsitsiklis, 1994; Borkar, 2009; Borkar and Meyn, 2000) of the form:

$$\theta_{t+1} = \theta_t + \alpha(\mathcal{T}\theta_t - \theta_t + w(t)), \tag{3.13}$$

where $\mathcal{T} : \mathbb{R}^{|\mathbb{S}|} \to \mathbb{R}^{|\mathbb{S}|}$ is a contraction operator, $w(t)$ is a noise term, α a learning rate and θ the parameters. In the tabular setting θ is just a vector representing the values at each state. As an example, TD(0) is known to converge to the fixed point of the bellman operator (R. S. Sutton, 1988):

$$\mathcal{T}V_\theta(s_t) = \mathop{\mathbb{E}}_{s_{t+1}\sim\pi}[r(s_t) + \gamma V_\theta(s_{t+1})]. \tag{3.14}$$

However, in practice we have access to a noisy version of the operator $\widetilde{\mathcal{T}}$ due to sampling process hence the noise term $w(t)$:

$$w(t) = r_t + \gamma V_\theta(s_{t+1}) - \mathop{\mathbb{E}}_{s_{t+1}\sim\pi}[r + \gamma V_\theta(s_{t+1})]. \tag{3.15}$$

In practice V^π is approximated using Monte Carlo rollouts (R. S. Sutton and Barto, n.d.) or TD methods (R. S. Sutton, 1988).

3.2.5 Linear Function Approximation

The methods developed previously scale linearly with the number of states. This becomes quickly intractable for large discrete state space and continuous settings. One way to remedy this is to use function approximation. It is possible to develop similar

operators using function approximator. The aim is to find a function $V_\theta : \mathbb{S} \to \mathbb{R}$ parametrized by θ that approximates V^π. We can fall back to the tabular setting by representing the states in a one hot vector form with $\theta \in \mathbb{R}^{|\mathbb{S}|}$. The goal is to find a set of parameters θ that minimizes the squared loss:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\pi}[(V^\pi - V_\theta)^2], \tag{3.16}$$

which yields the following update by taking the derivative with respect to θ:

$$\theta_{t+1} = \theta_t + \alpha(V^\pi(s_t) - V_{\theta_t}(s_t))\nabla_{\theta_t} V_{\theta_t}(s_t), \tag{3.17}$$

where α is a learning rate.

However, using function approximation can introduce the issue of *partial observability* (Åström, 1965). Indeed as states are approximated using a function, we do not have access to the full state but rather only a *observation*. One fundamental assumption of the MDP framework is that the world is fully observable; however, both in the real world and with function approximation this assumption is not satisfied.

3.2.6 Partially Observable MDP

A partially observable Markov Decision Process(POMDP) is a generalization of the MDP framework that allows the agents to only have access to observations of the world rather than the full state.

Definition 14 (Partially Observable Markov Decision Process (Sondik, 1978))**.** *A Partially Observable Markov Decision Process (POMDP) is defined as a tuple $(\mathcal{S}, \mathcal{P}, r, \omega, O)$ where:*

- $\mathcal{S}$ *is a discrete set of states.*
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ *is a transition function.*
- $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ *is a reward function.*

- ω *is a discrete set of observation.*
- $O : \mathcal{S} \times \mathcal{A} \times \omega \mapsto [0, 1]$ *is a function defining the probability of observing a certain observation* $o \in \omega$ *based on a state and action.*

This framework is much more general and applicable than the fully observable one. However, from both a theoretical and practical point of view, it remains a difficult framework to analyze, hence why the fully observable framework is more popular.

3.3 Control

In the previous section, we discussed estimating the value of a policy. However, in many cases, the actual goal (control) is to use this estimate to *improve* on the policy.

3.3.1 Bellman Optimality Equations

In the control case, the goal is to find the optimal policy π^* that maximizes the discounted expected return. As in the previous section, we define the optimal value function V^* as the fixed point of the nonlinear optimal Bellman operator:

$$\mathcal{T}^* v^* = \max_{a \in \mathcal{A}} [r(a) + \gamma P(a) v^*]. \tag{3.18}$$

3.3.2 Policy Iteration

Using the operator defined in eq 3.18, we now define a control algorithm. The main framework used for control in Reinforcement Learning is called Generalized Policy Improvements(GPI) (R. S. Sutton and Barto, 1998). GPI alternates between policy evaluation and policy improvements.

However, this algorithm requires to have a model of the environment to select the next optimal action. To remedy the concept of Q-function was introduced (Watkins and Dayan, 1992).

Algorithm 4 General Policy Improvements (R. S. Sutton and Barto, 1998)

Input: π, α, γ
Policy Evaluation:
  for all steps **do**
    Choose $a \sim \pi(\mathcal{S})$
    Take action a, observe $r(s), s'$
    $V(s) = r(s) + \gamma V(s')$
  end for
Policy Improvement:
  for all $s \in \mathcal{S}$ **do**
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s'} p(s', r|s, a) + [r + \gamma V(s')]$
  end for
If policy stable stop, else go to Policy Evaluation

3.3.3 Q function

Q-function's are state action pair (Watkins and Dayan, 1992) representing the expected discounted return from s_0 if action a_0 is to be taken. An optimal policy can be obtained by simply selecting the action at each state yielding the highest future discounted return. Mathematically this can be observed using Bellman's equation:

$$\begin{aligned} Q^*(s,a) &= \mathop{\mathbb{E}}_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s = s_0, a = a_0] \\ &= \mathop{\mathbb{E}}_{s'}[r_t + \gamma V(s')] \\ &= \mathop{\mathbb{E}}_{s'}[r_t + \gamma \text{max}_{a'} Q^*(s', a')]. \end{aligned} \tag{3.19}$$

Selecting the optimal action can easily be done by taking the max Q-values over all actions. This algorithm is called Sarsa (State–action–reward–state–action).

Algorithm 5 SARSA (R. S. Sutton and Barto, 1998)

Input: π, α, γ
for all steps **do**
  Choose $a \sim \pi(\mathcal{S})$
  Take action a, observe $r(s), s'$
  $Q(s,a) = Q(s,a) + \alpha(r(s) + \gamma Q(s', a') - Q(s,a))$
end for

Estimating the optimal action requires calculating the value of each action and

taking the argmax. This can be problematic in continuous action space.

3.3.4 Policy Gradient

If the action space is continuous or large, using look-up tables for Q-values can become quickly intractable. One way to circumvent this problem is to use function approximation on the policy. We define a policy π_θ parametrized by θ. The goal is to find a set of parameters θ such as to maximize: $J(\theta) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_{t+1}]$.

Definition 15 (Policy Gradient (R. S. Sutton, McAllester, et al., 2000))**.** *The gradient* $\nabla J(\theta)$ *can be expressed as:*

$$\nabla J(\theta) = \int_s d^\pi(s) \int_a \nabla_\theta \pi(a, s) Q^\pi(s, a). \tag{3.20}$$

In practice, those integrals can be estimated using samples. Furthermore Q is often unknown, but can be approximated using rollouts $G_t = r_t + \gamma r_{t+1} + ... + \gamma^n r_{t+n}$. This algorithm is called REINFORCE (Williams and Zipser, 1995).

3.3.5 Actor-Critic

Policy gradient methods can be enhanced by bootstrapping on a learned value function to approximate G_t instead of using Monte Carlo rollouts. This is the central idea behind the actor-critic (Konda and Tsitsiklis, 2000) method illustrated in figure 3.1. Actor-critic can be thought of as combining policy gradient and value based method. Many of the recent algorithms developed are based on this framework (Mnih, Badia, et al., 2016; Schulman, Wolski, et al., 2017; Wu et al., 2017).

3.4 Deep Reinforcement Learning

Before the advent of Deep Learning, Reinforcement learning algorithm showed great success on low dimensional tasks but failed on high-dimensional data. In recent years

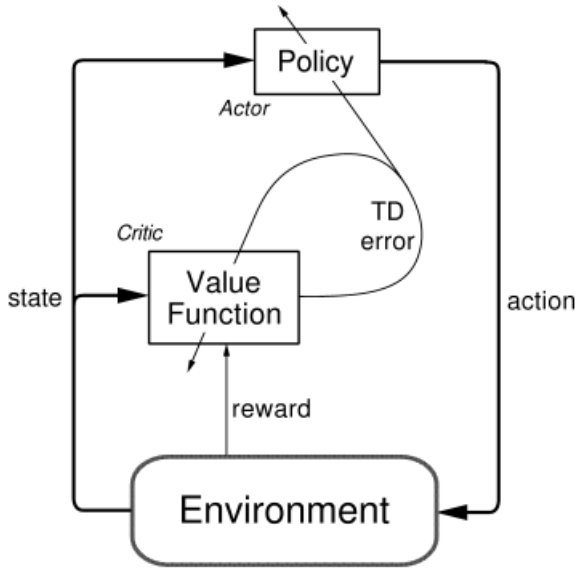


Figure 3.1: Actor-critic

there has been significant advances in Supervised Learning to tackle high-dimensional data using Deep Neural Networks. Their use in the field of Reinforcement Learning has driven many of the recent successes (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Silver et al., 2016).

3.4.1 Deep Neural Network

A Deep Neural Network (DNN) is a an artificial neural network with several non linear layers as described in Fig 3.2.

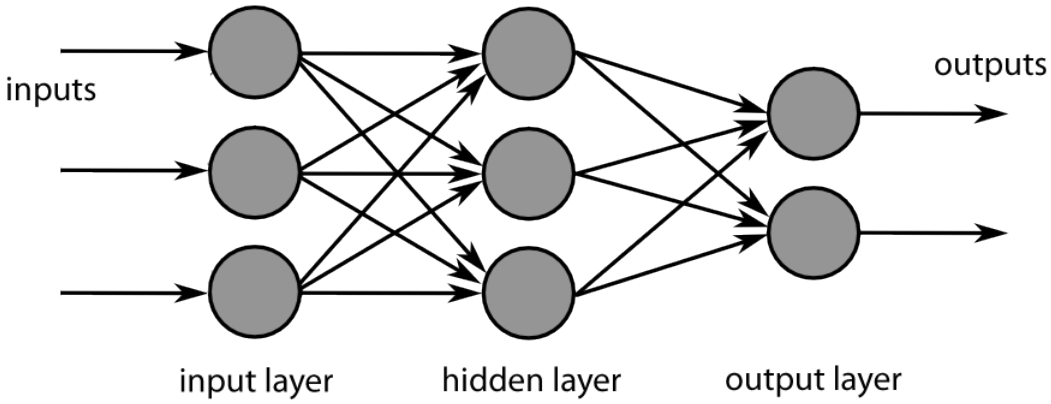


Figure 3.2: Deep Neural Network

DDN are biologically inspired model that posses the ability to learn compact low dimensional representation of high dimensional data (Goodfellow, Bengio, and Courville, 2016; Krizhevsky, Sutskever, and Hinton, 2012). There exists many variations/parametrization of a neuron but the prototypical one can be described as follows:

$$\widehat{y} = f(w^T x + b), \tag{3.21}$$

where x is the input vector, w the weight vector, b the bias term, and f a nonlinear activation function. There exist many activation functions used in the deep learning literature. One of the most popular ones is called ReLU (Nair and Hinton, 2010; B. Xu et al., 2015)

$$f(x) = \max(0, x). \tag{3.22}$$

In classification the goal is to find a set of weights w such as to minimize the loss:

$$\text{argmin}_w \left\| \widehat{y} - y \right\|_2, \tag{3.23}$$

where y are the labels. The most widely used algorithm used to find those weights is called back-propagation (Werbos, 1982). Back-propagation is a family of gradient descent approach exploiting the chain rules to learn optimal weights. In Supervised Learning, this has shown to be an effective mechanism to learn meaningful representations of high dimensional data, as demonstrated in (Krizhevsky, Sutskever, and Hinton, 2012). To improve on the general formalization of neural network to sequential data, the concept of Recurrent Neural Network was introduced (Rumelhart, Hinton, Williams, et al., 1987).

3.4.2 Recurrent Neural Network

Recurrent neural network(RNN) are a special kind of neural network designed specifically for sequential data (Rumelhart, Hinton, Williams, et al., 1987; Hochreiter and Schmidhuber, 1997). The central concept behind RNN is to share the weights learned

across time steps. In this section let's consider the task of regression. The goal is to predict a sequence of labels $y_1, y_2, ..., y_t$ from a sequence of input $x_1, x_2, ...x_t$. At each time step the prediction $\widetilde{y}_t$ is calculated as follows:

$$\begin{aligned} \widetilde{y}_t &= w_y^T h_t + b \\ h_t &= f(w_h^T h_{t-1} + w_x^T x_t) \end{aligned} \tag{3.24}$$

where w_h, w_y, w_x are weight matrices, f a non linear function, h the hidden state and b the bias term. A visual illustration of an RNN can be found in Fig 3.3. The weight

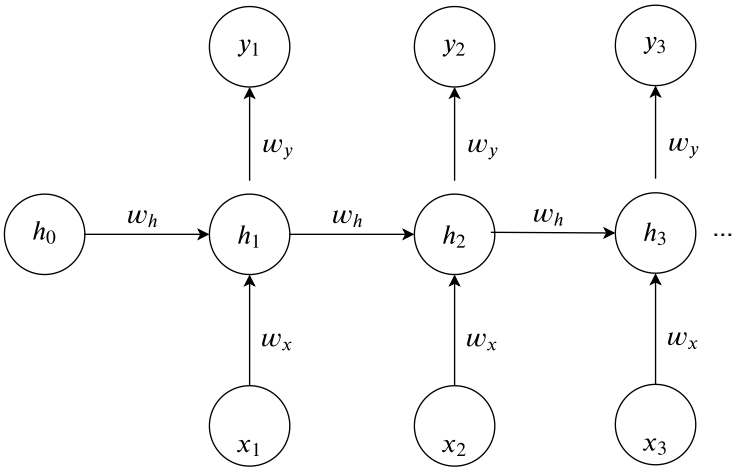


Figure 3.3: Graphical representation of a Recurrent Neural Network.

matrices are trained by back propagating the error through time using the chain rule (Williams and Zipser, 1995). For many years RNN's were complicated model to train due to the vanishing and exploding gradient issue (Bengio, Simard, Frasconi, et al., 1994; Hochreiter, 1998). When the RNN is unrolled for many time steps, the successive matrix multiplication causes the gradient to explode or vanish.

Long Short Term Memory cells are a robust solution to this problem introduced by (Hochreiter and Schmidhuber, 1997). The specific architecture is complicated; however, the key component of LSTM is a gating mechanism z_t defined as follows:

$$\begin{aligned} z_t &= \sigma(w_{z,h}^T h_{t-1} + w_{z,x}^T x_t + b) \\ h_t &= z_t h_t + (1 - z_t) h_{t-1} \end{aligned} \tag{3.25}$$

This gating z_t enables the network to learn to ignore some states and propagate the gradient directly.

3.4.3 Deep Neural Network meets Reinforcement Learning

Reinforcement learning algorithms combined with deep neural network as function approximation yields promising results on high dimensional problem (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013; Mnih, Badia, et al., 2016; Schulman, Wolski, et al., 2017) such as Atari (Bellemare et al., 2013) and Mujoco (Todorov, Erez, and Tassa, 2012). However, they appear to be unstable to train and requires many *tricks* to converge to a good solution. One severe problem when using deep neural networks as function approximation arises from the fact that the data is not IID (Independently and Identically Distributed). Indeed the samples received during the trajectory are highly correlated. This problem does not arise in supervised learning, as it is possible to shuffle the data randomly. It has been shown to cause severe learning problem in Reinforcement Learning. The first paper attempting to combine Deep Learning and Reinforcement Learning (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013) used a replay buffer with Q-learning to combat this issue.

We now describe 2 different deep RL algorithm: Asynchronous Actor Critic (A3C (Mnih, Badia, et al., 2016)) and Proximal Policy Optimzation (PPO (Schulman, Wolski, et al., 2017)), and explain the tools used to make them stable. Both methods are based on an actor critic formulation.

3.4.4 A3C

In Asynchronous Advantage Actor-Critic (Mnih, Badia, et al., 2016), the main idea is to propose a framework using asynchronous gradient descent for the optimization of deep neural network controllers. This effectively decorrelates samples by having several agents running different trajectories at the same time. The data is then aggregated,

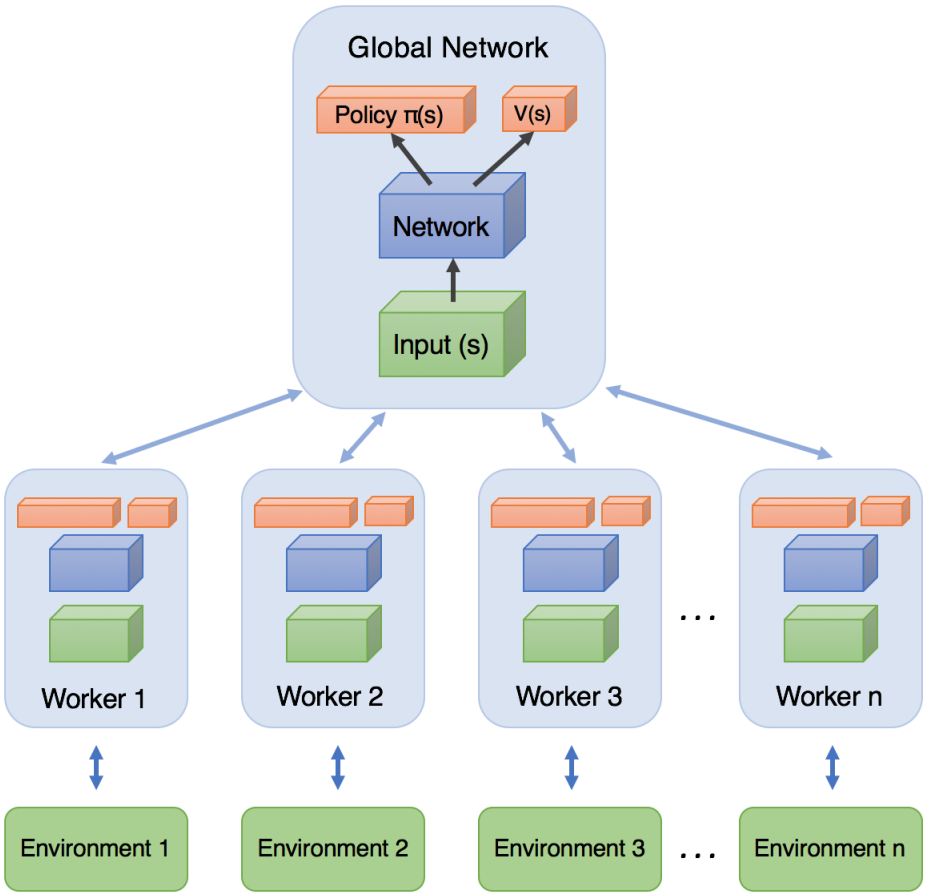


Figure 3.4: A3C diagram(reproduced from this blog with permission)

and one update is applied to all agents. This is effectively done by running several agents in parallel and using a batch of data to update the parameters. Figure 3.4 is a diagram explaining the algorithm.

In the paper, they demonstrate how their proposed method achieves performant policy on Atari domain. However, A3C achieved poor performance on continuous control tasks.

3.4.5 PPO

The motivation behind Proximal Policy Optimization (Schulman, Wolski, et al., 2017) is to enable multiple epochs of mini-batch updating on deep neural controllers without

suffering from *catastrophic forgetting*. It is built on top of the concept of Trust-Region Policy methods (Schulman, Levine, et al., 2015). Trust Region Policy Optimization (TRPO) is a mathematical framework designed to guarantee a monotonic improvement of the policy at every time step. Concretely TRPO minimizes the following constrained loss:

$$\begin{aligned} &\text{argmax}_\theta \mathop{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t \right] \\ &\text{subject to} \quad \mathop{\mathbb{E}}_t \left[\text{KL}\left[\pi_\theta(a_t|s_t), \pi_{\theta_{\text{old}}}(a_t|s_t) \right] \right] \leq \delta, \end{aligned} \tag{3.26}$$

where A_t is an advantage function at time step t, δ a regularization hyper-parameter. Effectively the constraint prevents the policy from changing too much when the *advantage* is not large enough. In Proximal Policy Optimization(PPO) they implement this regularization with Deep Neural Networks and demonstrate state of the art results on continuous control tasks. Furthermore they demonstrate how using this regularization it is possible to effectively train a deep neural network for several epochs on a single batch of data.

PPO is a good example of a successful application of regularization in Reinforcement Learning.

3.5 Regularization in Reinforcement Learning

Regularization is a central concept in Machine Learning and has shown to be highly effective in Reinforcement Learning as well (Neu, Jonsson, and Gómez, 2017; Farahmand, 2011; Schulman, Wolski, et al., 2017). Regularization in RL has been considered in several different perspectives. In this section, we discuss the most popular one.

3.5.1 Spatial Regularization

As described in the previous section when the state space becomes large, the solution is to use a function approximator to represent the state. One line of investigation focuses on regularizing the features learned on the state space (Farahmand et al., 2009; Petrik et al., 2010; Pazis and Parr, 2011; Farahmand, 2011; B. Liu, Mahadevan, and J. Liu, 2012; Harrigan, 2016). These approaches assume that nearby states in the state space have similar value. It is done by controlling the complexity of the approximator used. One simple example of this is using an l1 regularization on the parameters of the model (Farahmand et al., 2009):

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{\pi}[(V^{\pi} - V_{\theta})^2] + \|\theta\|_1 . \tag{3.27}$$

More complex regularization scheme can be studied, for example (Farahmand, 2011) introduce Approximate Value iteration, designed to apply regularization technique to select value function estimators from rich function space. As in supervised learning, it is crucial to favor *simple* solution while approximating optimal policies. Regularizing too heavily can impact learning.

3.5.2 Entropy Policy Regularization

There exist several ways to regularize policy gradient methods. The most commonly used in recent research is called entropy regularization (Neu, Jonsson, and Gómez, 2017; Schulman, Wolski, et al., 2017; Bartlett and Tewari, 2009). In policy gradient methods, one successful approach consists of regularizing the entropy of your policy distribution (Neu, Jonsson, and Gómez, 2017). Policy gradient methods tend to converge to distributions with low entropy. By adding an entropy bonus, it encourages the policy to explore and often yields much better performance:

$$\text{argmax}_{\pi} = \mathop{\mathbb{E}}_{s,a\sim\pi}[r(s,a) + R(s)], \tag{3.28}$$

where R is a function calculating the entropy of a distribution.

3.5.3 Value based temporal regularization

In this section, we describe various work that attempts to regularize either the value or the action temporally. The notion of *temporal regularization* has not been used in the Reinforcement Learning vocabulary, but we argue that many works can be depicted as such.

N-step methods (R. S. Sutton, 1985): In Reinforcement Learning the target used (expected discounted return) has a temporal aspect. N-step and λ methods are a way to enforce temporal consistency in the future. In contrast, in this thesis, we propose to enforce temporal consistency *backward* in time. Bootstrapping on values that are close to each other along the trajectory enforces temporal consistency and reduces variance at the cost of introducing bias.

Backward bootstrapping (Baird, 1995): Residual algorithms are a class of algorithm aimed at stabilizing reinforcement learning with function approximation. Residual methods can be interpreted as regularizing in feature space based on temporal proximity. To see this, let's first consider the original semi-gradient TD update:

$$\nabla_{\text{TD}} = -\alpha(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t))\nabla_\theta V_\theta(s_t). \tag{3.29}$$

The gradient with respect to $V(s_{t+1}$ is ignored. The idea behind residual is to also consider this gradient yielding the following update:

$$\nabla_{\text{RG}} = -\alpha(r_t + \gamma V_\theta(s_{t+1}) - V_\theta(s_t))(\nabla_\theta V_\theta(s_{t+1}) - \nabla_\theta V_\theta(s_t)). \tag{3.30}$$

Intuitively the gradient for residual methods is scaled by how the two states s_t, s_{t-1} differs. This will encourage state that are close to each other both in feature space and temporal space to have similar value.

Natural Value Approximator (Z. Xu et al., 2017): Natural Value Approximator is probably the most relevant work for this thesis. They define the value of a

state by projecting the previous state's estimates by adjusting for the reward and γ:

$$G_\theta^\beta(s_t) = (1 - \beta(s_t))V_\theta(s_t) + \beta(s_t)\frac{G_\theta^\beta(s_{t-1}) - r(s_t)}{\gamma}, \tag{3.31}$$

and then modify the value estimate using the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}[(1 - \beta(s_t))(V^\pi(s_t) - V_\theta(s_t))^2] + \mathrm{c}_\beta\, \mathbb{E}[\beta(s_t)(V^\pi(s_t) - G_\theta^\beta(s_t))^2], \tag{3.32}$$

where c_β is a hyper-parameter. They do not differentiate through the sequence and cut the gradient after 1 step. While the motivations are similar, the work by (Z. Xu et al., 2017) focused exclusively on experimental performance, whereas in this thesis, we approach it from a more theoretical perspective. The precise differences between our work and this one are discussed more in depth later in the thesis.

3.5.4 Action based temporal regularization

The concept of Temporal Regularization can also be applied to action. It is particularly relevant in physical domains. Indeed actions in the physical world often display strong temporal coherence, in particular for exploration.

Advantage Amplification in Slowly Evolving Latent-State Environments (Mladenov et al., 2019): The motivation of this work is to use RL in latent state environments. They argue that a central issue in long-horizon environments is one of *small action gap*. The main proposed method is to introduce a cost for switching action. Theoretically, they demonstrate how it leads to an amplification of the action gap. It is a natural result as penalizing the policy only change action when the *benefit* is larger than the switching cost.

Autoregressive Policies for Continuous Control Deep Reinforcement Learning (Korenkevych et al., 2019): In this work the author consider the problem of *structured exploration*. To encourage coherent exploration in time, the noise introduced in policy gradient for exploration is modelled using an autoregressive model

(Lütkepohl, 2005). Effectively this regularize the actions to explore in a temporally coherent manner through the trajectory.

4

Value-Based Temporal Regularization

Regularization in the feature/state space or *spatial regularization* as we call it, exploit the regularities that exist in the observation (or state). In contrast, *temporal regularization* considers the temporal structure of the value estimates through a trajectory. Practically this is done by smoothing the value estimate of a state using estimates of states that occurred earlier in the trajectory. In this section, we first introduce the concept of temporal regularization and discuss its properties in the policy evaluation setting. We then show how this concept can be extended to exploit information from the entire trajectory by casting temporal regularization as a time series prediction problem.

4.1 Algorithm

Let us focus on the simplest case where the value estimate at the current state is regularized using only the value estimate at the previous state in the trajectory, yielding updates of the form:

$$\begin{aligned} V^{\beta}(s_t) &= \mathop{\mathbb{E}}_{s_{t+1}, s_{t-1} \sim \pi}[r(s_t) + \gamma((1-\beta)V^{\beta}(s_{t+1}) + \beta V^{\beta}(s_{t-1}))] \\ &= r(s_t) + \gamma(1-\beta) \sum_{s_{t+1} \in \mathcal{S}} p(s_{t+1}|s_t) V^{\beta}(s_{t+1}) + \gamma\beta \sum_{s_{t-1} \in \mathcal{S}} \frac{p(s_t|s_{t-1})p(s_{t-1})}{p(s_t)} V^{\beta}(s_{t-1}), \end{aligned} \tag{4.1}$$

for a parameter $\beta \in [0, 1]$ and $p(s_{t+1}|s_t)$ the transition probability induced by the policy π. It can be rewritten in matrix form as $v^\beta r = r + \gamma(((1-\beta)P^\pi + \beta\widetilde{P^\pi})v^\beta)$, where $\widetilde{P^\pi}$ corresponds to the reversal Markov chain of the MDP. We define a temporally regularized Bellman operator as:

$$\mathcal{T}^\beta v^\beta = r + \gamma((1-\beta)P^\pi v_\beta + \beta\widetilde{P^\pi}v^\beta). \tag{4.2}$$

To alleviate the notation, we denote P^π as P and $\widetilde{P^\pi}$ as $\widetilde{P}$.

Remark. *For $\beta = 0$, Eq. 4.2 corresponds to the original Bellman operator.*

We can prove that this operator has the following property.

Theorem 2. *The operator $\mathcal{T}^\beta$ has a unique fixed point $V^{\pi,\beta}$ and $\mathcal{T}^\beta$ is a contraction mapping.*

Proof. We first prove that $\mathcal{T}^\beta$ is a contraction mapping in L_∞ norm. We have that

$$\begin{aligned}\left\|\mathcal{T}^\beta u - \mathcal{T}^\beta v\right\|_\infty &= \left\|r + \gamma((1-\beta)Pu + \beta\widetilde{P}u) - (r + \gamma((1-\beta)Pv + \beta\widetilde{P}v))\right\|_\infty \\ &= \gamma\left\|((1-\beta)P + \beta\widetilde{P})(u-v)\right\|_\infty \\ &\leq \gamma\left\|u - v\right\|_\infty,\end{aligned} \tag{4.3}$$

where the last inequality uses the fact that the convex combination of two row stochastic matrices is also row stochastic (the proof can be found in lemma 3). Then using Banach fixed point theorem, we obtain that $V^{\pi,\beta}$ is a unique fixed point. □

Lemma 3. *The convex combination of two row stochastic matrices is also row stochastic.*

Proof. Let e be vector a columns vectors of 1.

$$\begin{aligned}(\beta P^\pi + (1-\beta)\widetilde{P^\pi})e &= \beta P^\pi e + (1-\beta)\widetilde{P^\pi}e \\ &= \beta e + (1-\beta)e \\ &= e.\end{aligned} \tag{4.4}$$

□

This theorem does not prove that it converges to the optimal solution of the original MDP but rather that it converges to some fixed point. In the policy evaluation setting, the bias between the original value function V^π and the regularized one V^π_β can be characterized as a function of the difference between P and its Markov reversal $\widetilde{P}$, weighted by β and the reward distribution.

Proposition 1. *Let* $v^\pi = \sum_{i=0}^{\infty} \gamma^i P^i r$ *and* $v^{\pi,\beta} = \sum_{i=0}^{\infty} \gamma^i ((1-\beta)P + \beta\widetilde{P})^i r$. *We have that*

$$\left\| v - v^{\pi,\beta} \right\|_\infty = \left\| \sum_{i=0}^{\infty} \gamma^i (P^i - ((1-\beta)P + \beta\widetilde{P})^i) r \right\|_\infty \leq \sum_{i=0}^{\infty} \gamma^i \left\| (P^i - ((1-\beta)P + \beta\widetilde{P})^i) r \right\|_\infty . \tag{4.5}$$

This quantity is naturally bounded for $\gamma < 1$.

This bound is intuitive. If the chain is highly irreversible at a state where the reward changes a lot like falling off a cliff then smoothing induces bias. However if the reward does not change when falling off a cliff then the bias induced is negligible even though the chain is irreversible.

Remark. *Let* P^∞ *denote a matrix where columns consist of the stationary distribution* μ. *By the property of reversal Markov chains and lemma 4, we have that* $\lim_{i\to\infty} \|P^i r - P^\infty r\| \to 0$ *and* $\lim_{i\to\infty} \|((1-\beta)P + \beta\widetilde{P})^i r - P^\infty r\| \to 0$, *such that the Marvov chain* P *and its reversal* $(1-\beta)P + \beta\widetilde{P}$ *converge to the same value. Therefore, the norm* $\|(P^i - ((1-\beta)P + \beta\widetilde{P})^i) r\|_p$ *also converges to 0 in the limit.*

Remark. *It can be interesting to note that if the chain is reversible, meaning that* $P = \widetilde{P}$, *then the fixed point of both operators is the same, that is* $v = v^\beta$.

4.1.1 Discounted average reward

The temporally regularized MDP has the same discounted average reward as the original one as it is possible to define the discounted average reward (Tsitsiklis and

Van Roy, 2002) as a function of the stationary distribution π, the reward vector and γ. To establish this result we first need the following lemma:

Lemma 4. *P and $(1-\beta)P+\beta\widetilde{P}$ have the same stationary distribution μ* $\quad \forall\beta \in [0,1]$.

Proof. It is known that P^π and $\widetilde{P^\pi}$ have the same stationary distribution. Using this fact we have that

$$\begin{aligned}\mu((1-\beta)P^\pi + \beta\widetilde{P^\pi}) &= (1-\beta)\mu P^\pi + \beta\mu\widetilde{P^\pi} \\ &= (1-\beta)\mu + \beta\mu \\ &= \mu.\end{aligned} \tag{4.6}$$

□

This leads to the following property.

Proposition 2. *For a reward vector r, the MDPs defined by the transition matrices P and $(1-\beta)P+\beta\widetilde{P}$ have the same average reward ρ.*

$$\frac{\rho}{1-\gamma} = \sum_i^\infty \gamma^i \pi^T r. \tag{4.7}$$

Proof. Using lemma 4, both P and $(1-\beta)P+\beta\widetilde{P}$ have the same stationary distribution and so discounted average reward. □

Intuitively, this means that temporal regularization only reweighs the reward on each state based on the Markov reversal, while preserving the average reward. State that are irreversible with rewards far from the average reward will be smoothed. This reflect the inductive bias underlying temporal regularization: *states that are close to each other temporally should have similar value.*

4.1.2 Temporal Regularization as a time series prediction problem:

It is possible to cast this problem of temporal regularization as a time series prediction problem, and use richer models of temporal dependencies, such as exponential

smoothing (Gardner, 2006), ARMA model (Box, Jenkins, and Reinsel, 1994), etc. We can write the update in a general form using n different regularizers $(\widetilde{v_0}, \widetilde{v_1}...\widetilde{v_{n-1}})$:

$$V(s_t) = r(s) + \gamma \sum_{i=0}^{n-1} [\beta(i) \widetilde{V}_i(s_{t+1})], \tag{4.8}$$

where $\widetilde{V}_0(s_{t+1}) = V(s_{t+1})$ and $\sum_{i=0}^{n-1} \beta(i) = 1$. For example, using exponential smoothing where $\widetilde{V}(s_{t+1}) = (1-\beta_\lambda)V(s_{t-1}) + (1-\beta_\lambda)\beta_\lambda V(s_{t-2})...$, the update can be written in operator form as:

$$\mathcal{T}^\beta v^\beta = r + \gamma \Bigg((1-\beta)\, Pv + \beta\, (1-\beta_\lambda) \sum_{i=1}^{\infty} \beta_\lambda^{i-1} \widetilde{P}^i v^\beta \Bigg), \tag{4.9}$$

and a similar argument as Theorem 1 can be used to show the contraction property. The bias of exponential smoothing in policy evaluation can be characterized as:

$$\left\| v^\pi - v^{\pi,\beta} \right\|_\infty \leq \sum_{i=0}^{\infty} \gamma^i \left\| (P^i - ((1-\beta)P + \beta(1-\beta_\lambda) \sum_{j=1}^{\infty} \beta_\lambda^{j-1} \widetilde{P}^j)^i) r \right\|_\infty . \tag{4.10}$$

Using more powerful regularizers could be beneficial, for example, to reduce variance by smoothing over more values (exponential smoothing) or to model the trend of the value function through the trajectory using trend adjusted model (Gardner Jr, 1985). An example of policy evaluation with temporal regularization using exponential smoothing is provided in Algorithm 6.

Algorithm 6 Policy evaluation with exponential smoothing

1: Input: $\pi, \alpha, \gamma, \beta, \lambda$
2: $p = V(s)$
3: **for all** steps **do**
4: Choose $a \sim \pi(s)$
5: Take action a, observe reward $r(s)$ and next state s'
6: $V(s) = V(s) + \alpha(r(s) + \gamma((1-\beta)V(s') + \beta p) - V(s))$
7: $p = (1-\beta_\lambda)V(s) + \beta_\lambda p$
8: **end for**

4.1.3 Control

In this section, we propose two sets of algorithms to regularize actions-values temporally. The first one considers regularizing the values directly, which indirectly should

encourage temporally coherent actions. The other one is to regularize actions directly to be temporally coherent.

4.1.3.1 Value regularization

Temporal regularization can be extended to MDPs with actions by modifying the target of the value function (or the Q values) using temporal regularization. Experiments (Sec. 4.3.2) present an example of how temporal regularization can be applied within an actor-critic framework. The theoretical analysis of the control case is not studied in this thesis.

4.1.3.2 Action regularization

Another example could be to regularize action to be *temporally coherent* through the trajectory. One way to enforce such coherence in Policy Gradient methods would be to regularize the objective function:

$$R(\pi) = \beta(\pi(a_{t-1}|s_{t-1}) - \pi(a_t|s_t)). \tag{4.11}$$

It is possible to see this formulation as a kind of entropy regularization (Neu, Jonsson, and Gómez, 2017) where the regularization is done in trajectory space compared as opposed to PPO, where it is done in policy space. Intuitively one wants to lower the entropy between actions that are temporally close to each other and increase it for states that are far from each other. This kind of regularization can lead to more consistent behavior in time, better exploration in physical systems, and overall less variance. Another perspective is to view the action through the trajectory as a time series and apply some entropy regularization to the prediction of the time series. It would be interesting to study the concepts of entropy in dynamical systems and demonstrate the convexity of the loss proposed using the framework developed by (Neu, Jonsson, and Gómez, 2017).

4.1.4 Temporal difference with function approximation:

It is also possible to extend temporal regularization using function approximation such as semi-gradient TD (R. S. Sutton and Barto, 2017). Assuming a function V_θ^β parameterized by θ, we can consider $r(s)+\gamma((1-\beta)V_\theta^\beta(s_{t+1})+\beta V_\theta^\beta(s_{t-1}))-V_\theta^\beta(s_t)$ as the target and differentiate with respect to $V_\theta^\beta(s_t)$. A simple instance of a temporally regularized semi-gradient TD algorithm can be found in algorithm 7.

Algorithm 7 Temporally regularized semi-gradient TD

1: Input: policy π,β,γ
2: **for all** steps **do**
3: Choose $a \sim \pi(s_t)$
4: Take action a, observe $r(s), s_{t+1}$
5: $\theta = \theta + \alpha(r + \gamma((1-\beta)V_\theta(s_{t+1}) + \beta V_\theta(s_{t-1})) - V_\theta(s_t))\nabla V_\theta(s_t)$
6: **end for**

4.1.5 Related work

As mentioned earlier, regularization in RL has been considered via several different perspectives. One line of investigation focuses on regularizing the features learned on the state space (Farahmand et al., 2009; Petrik et al., 2010; Pazis and Parr, 2011; Farahmand, 2011; B. Liu, Mahadevan, and J. Liu, 2012; Harrigan, 2016). Explicit regularization in the temporal space has received much less attention. Temporal regularization in some sense may be seen as a "backward" multi-step method (R. S. Sutton and Barto, 1998). The closest work to this one is possibly (Z. Xu et al., 2017), where they define natural value approximator by projecting the previous state's estimates by adjusting for the reward and γ. As a reminder, the natural value estimate is defined as:

$$G_t^\beta = (1-\beta_t)V_t + \beta_t \frac{G_{t-1}^\beta - r_t}{\gamma}, \tag{4.12}$$

and then modify the value estimate using the following loss:

$$\mathbb{E}[(1-\beta_t)(V^\pi - V_t)^2] + \mathrm{c}_\beta\, \mathbb{E}[\beta_t (V^\pi - G_t^\beta)^2]. \tag{4.13}$$

Their formulation is driven by intuition for experimental performance. This is in contrast to ours that attempts to lay the foundations to formally study the concept of temporal regularization. The first difference is that they divide their estimate by γ. While this may make sense intuitively, in theory, this can easily lead to divergence with small γ. We do not advocate for one solution or the other in particular as it is problem-dependent. In the methods presented in this thesis, γ is a complicated quantity to deal with as it limits the horizon considered and leads to arbitrary decisions on the form of the algorithm. Those issues could be addressed by considering the average reward case. In the Reinforcement Learning community, there is an ongoing debate on the role of γ. In this thesis, we hypothesize that transitioning to average reward would solve and strengthen those methods (Mahadevan, 1994). This is left for future work. The other main difference is the loss considered. Rather than having β decide between the forward and the backward loss, the authors add a scalar coefficient c_β and argue that this leads to better performance. From a theoretical perspective, this scalar impacts both the amount of regularization and the learning rate of the algorithm. Higher c_β leads to a higher learning rate. Mixing both quantities into one hyper-parameter can lead to complications when tuning it. Their formulation, while sharing similarities in motivation, leads to different theory and algorithm. Convergence properties and bias induced by this class of methods were also not analyzed in Z. Xu et al. (2017).

4.2 Policy evaluation

We now present empirical results illustrating potential advantages of temporal regularization, and characterizing its bias and variance effects on value estimation.

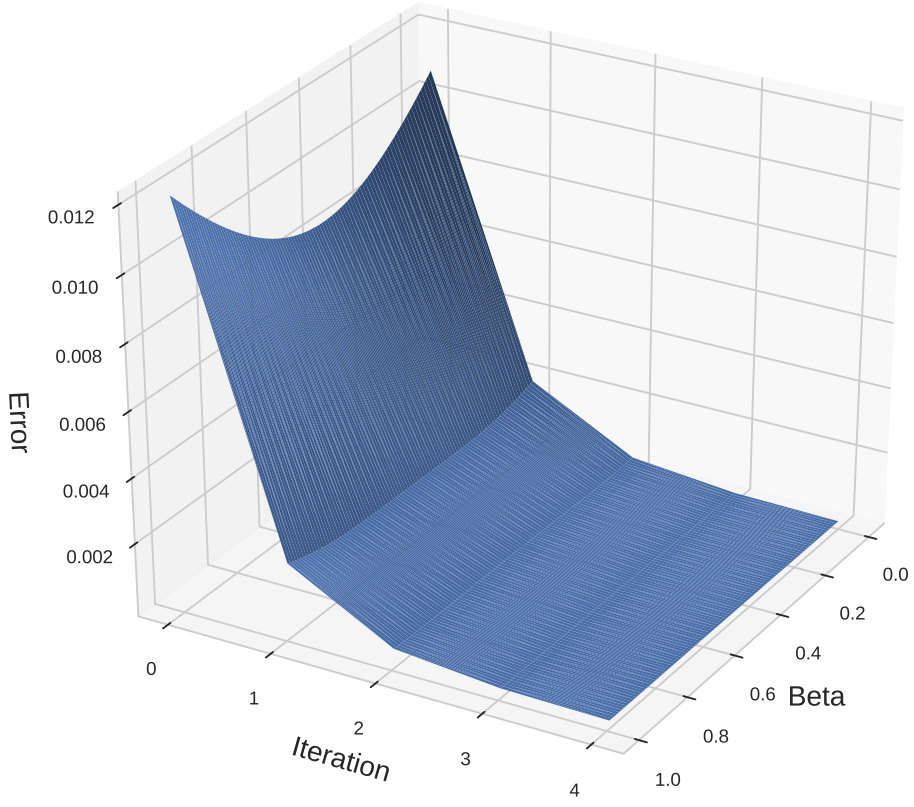


Figure 4.1: Distance between the stationary transition probabilities and the estimated transition probability for different values of regularization parameter β.

4.2.1 Mixing time

This first experiment showcases that the underlying Markov chain of an MDP can have a smaller mixing time when temporally regularized. The mixing time can be seen as the number of time steps required for the Markov chain to get *close enough* to its stationary distribution. Therefore, the mixing time also determines the rate at which policy evaluation converges to the optimal value function (Baxter and Bartlett, 2001).

We consider a synthetic MDP with 10 states where transition probabilities are sampled from the uniform distribution. Let P^{∞} denote a matrix where columns consist of the stationary distribution μ. To compare the mixing time, we evaluate the error corresponding to the distance of P^i and $\left((1-\beta)P + \beta\widetilde{P}\right)^i$ to the convergence point

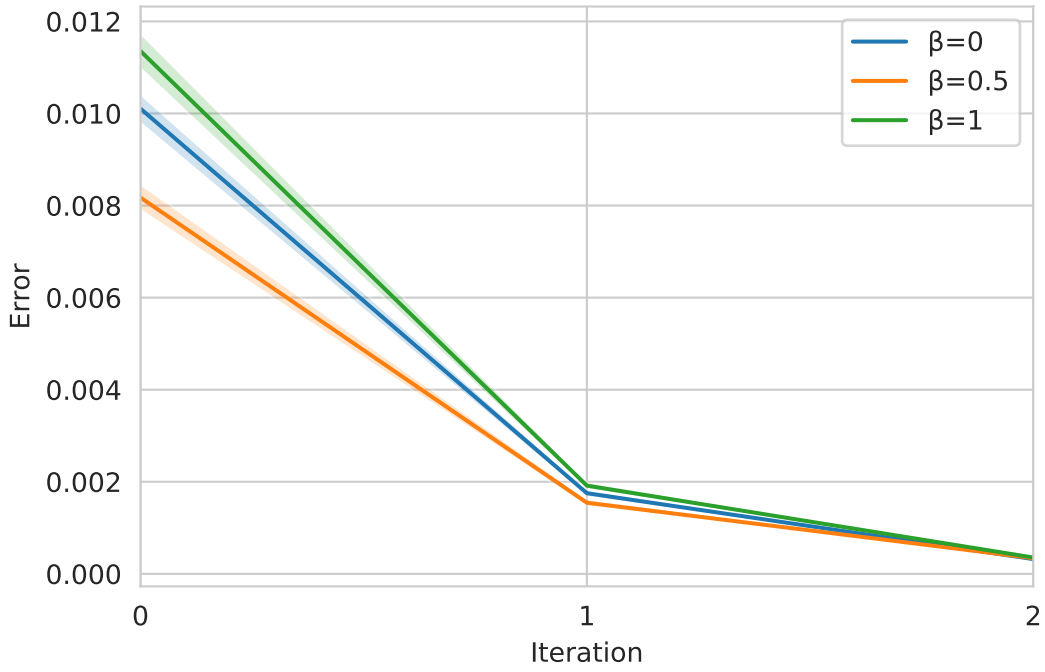


Figure 4.2: Distance between the stationary transition probabilities and the estimated transition probability for $\beta = \{0, 0.5, 1\}$

.

P^{∞} after i iterations. Figure 4.1 and 4.2 display the error curve when varying the regularization parameter β. We observe a U-shaped error curve, that intermediate values of β in this example yields faster mixing time. One explanation is that transition matrices with extreme probabilities (low or high) yield poorly conditioned transition matrices. Regularizing with the reversal Markov chain often leads to a better-conditioned matrix at the cost of injecting bias.

4.2.2 Bias

It is well known that reducing variance comes at the expense of inducing (smaller) bias. This has been characterized previously (Sec. 4.1) in terms of the difference between the original Markov chain and the reversal weighted by the reward. In this experiment, we attempt to give an intuitive idea of what this means. More specifically, we would expect the bias to be small if values along the trajectories have

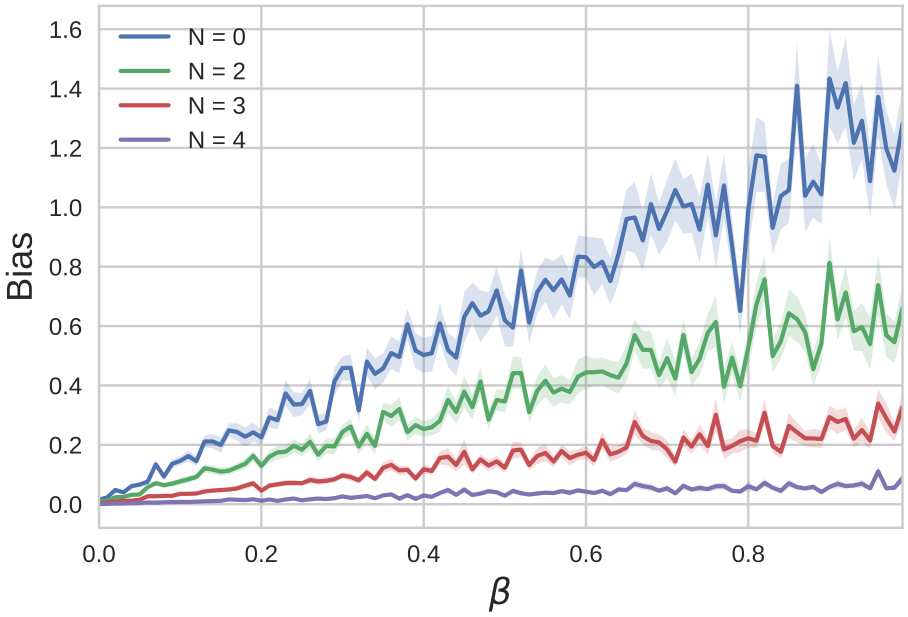


Figure 4.3: Mean difference between V^{π} and $V^{\pi,\beta}$ given the regularization parameter β, for different amount of smoothed states N.

similar values. To this end, we consider a synthetic MDP with 10 states where both transition functions and rewards are sampled randomly from a uniform distribution. In order to create temporal dependencies in the trajectory, we smooth the rewards of N states that are temporally close (in terms of trajectory) using the following formula: $r(s_t) = \frac{r(s_t)+r(s_{t+1})}{2}$. Figure 4.3 shows the difference between the regularized and un-regularized MDPs as N changes, for different values of regularization parameter β. We observe that increasing N, meaning more states get rewards close to one another, results in less bias. It is due to rewards emphasizing states where the original and reversal Markov chain are similar.

4.2.3 Variance

The primary motivation of this work is to reduce variance; therefore, we now consider an experiment targeting this aspect. Figure 4.4 shows an example of a synthetic, 4-state MDP, where the variance of the reward for S_1 is (relatively) high.

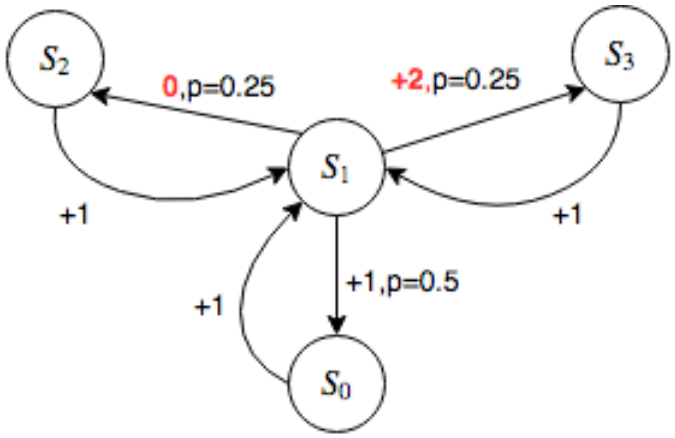


Figure 4.4: Synthetic MDP where state S_1 has high variance.

We consider an agent that is evolving in this world, changing states following the stochastic policy indicated. We are interested in the error when estimating the optimal state value of S_1, $V^{\pi}(S_1)$, with and without temporal regularization, denoted $V^{\pi,\beta}(S_1)$, $V^{\pi}(S_1)$, respectively. The discount factor used is $\gamma = 0.95$. Figure 4.5 shows

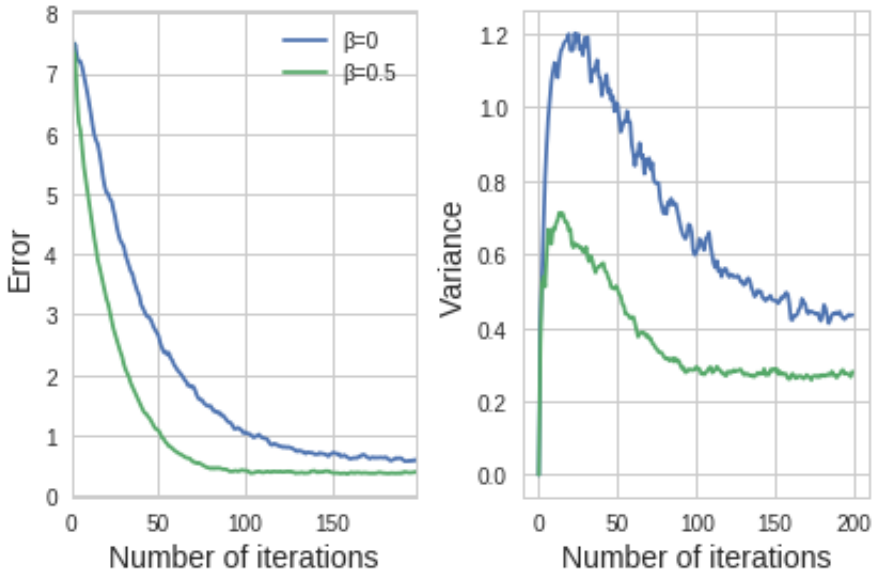


Figure 4.5: Left plot shows absolute difference between original ($V(S_1)$) and regularized ($V^{\beta}(S_1)$) state value estimates to the optimal value $V^{\pi}(S_1)$. Right plot shows the variance of the estimates V.

these errors at each iteration, averaged over 100 runs. We observe that temporal regularization indeed reduces the variance and thus helps the learning process by making the value function easier to learn.

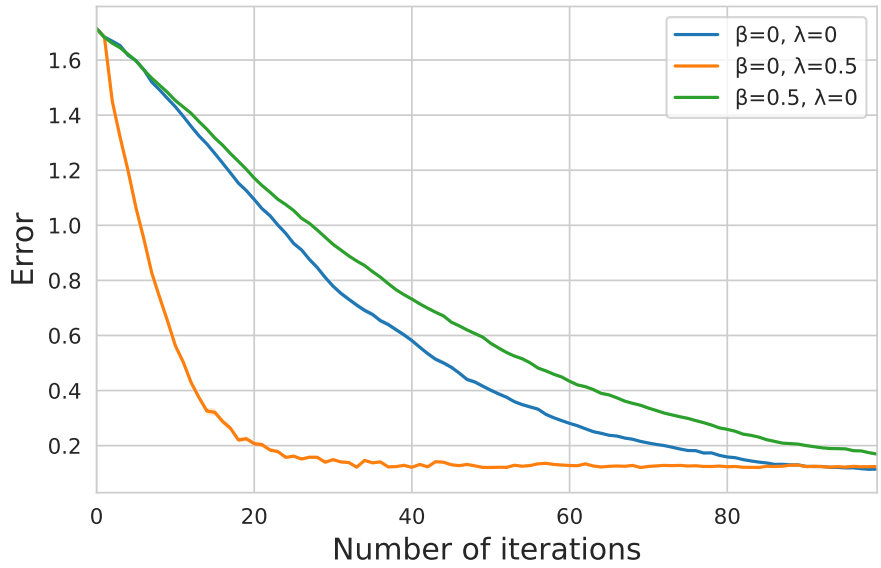


Figure 4.6: Performance comparison between TD(0), TD(λ) and temporally regularized TD(0) when S_2 and S_3 are terminal states.when S_2 and S_3 are terminal states and $r(S_2) = r(S_3) = 2$.

We now modify the environment to demonstrate several interesting properties of temporal regularization. We consider the episodic framework where S_2 and S_3 are terminal states. The first aspect we demonstrate is that if the underlying assumption of smoothness of the value function through the trajectory is not satisfied, using temporal regularization can slow down learning. To illustrate this we define the rewards as follows: $r(S_0) = 0, r(S_1) = 0, r(S_2) = r(S_3) = 2$. The estimate of $V(S_1)$ does not suffer anymore from the variance issue as the value of the next states is $V(S_2) = V(S_3)$. Furthermore, in contrary to last example, $V(S_0)$ is not a better estimator of the value of $V(S_1)$ than $V(S_2)$ and $V(S_3)$. In practice, this means that temporal regularization will slow down learning. This phenomena is illustrated in Fig 4.6. The fastest method is $TD(\lambda)$ followed by $TD(0)$. This experiment underlines the fact that if the underlying assumption is not satisfied, temporal regularization can slow down learning. The second modification is used to underline the difference between the forward smoothing that TD(λ) does and the backward smoothing done by temporal regularization. It is used to demonstrate when temporal regularization can

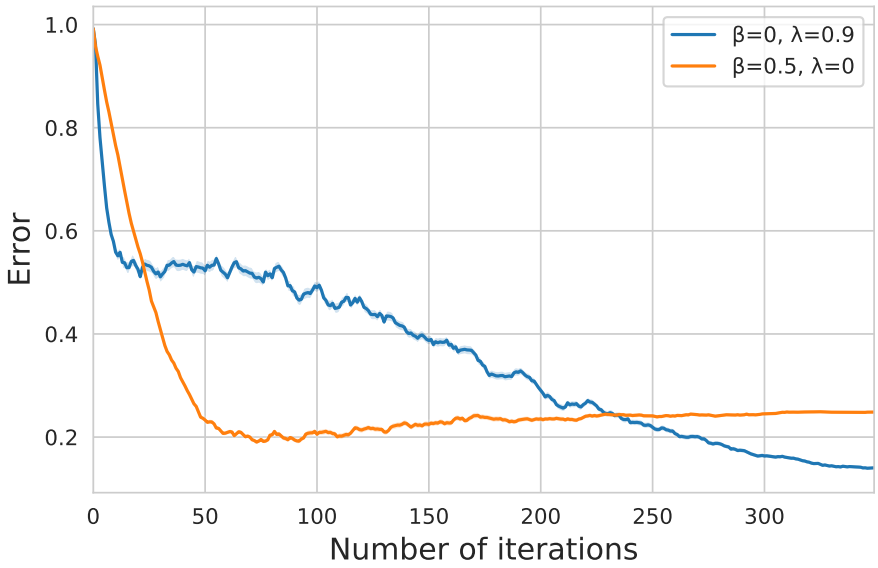


Figure 4.7: Performance comparison between TD(λ) and temporally regularized TD(0) when S_2 and S_3 are terminal states.

be harmful or helpful. We modify the environments such that S_2 and S_3 are terminal states as described in Figure 4.4. The results are reported in Fig 4.7. During the first phase, TD(λ) learns faster than temporal regularization due to the fast propagation of the information. In the second phase, the error in TD(λ) remains high compared to temporal regularization due to the variance of the returns between S_2 and S_3. As the learning rate slowly decays, the performance of TD(λ) converges to 0. In contrast, while temporal regularization is able to learn faster in the middle of the training using backward bootstrapping, some residual bias remains at the end of the training. In particular, when evaluating $V(S_3)$, its value is bootstrapped on $V(S_1)$ creating a bias. This bias will not go to 0 unless the regularization coefficient goes to 0. In this experiment there exists a clear difference between forward and backward smoothing due to the fact that S_2 and S_3 are terminal states. In settings where the environment would quickly loop back, it is expected that forward and backward smoothing could have similar effects.

4.2.4 Propagation of the information

We now illustrate with a simple experiment how temporal regularization allows the information to spread faster among the different states of the MDP. For this purpose, we consider a simple MDP, where an agent walks randomly in two rooms (18 states) using four actions (up, down, left, right), and a discount factor $\gamma = 0.9$. The reward is $r_t = 1$ everywhere and passing the door between rooms (shown in red on Figure 4.8) only happens 50% of the time (on attempt). The episode starts at the top left and terminates when the agent reaches the bottom right corner. The goal is to learn the true value function associated with the policy by walking along this MDP (this is not a race toward the end).

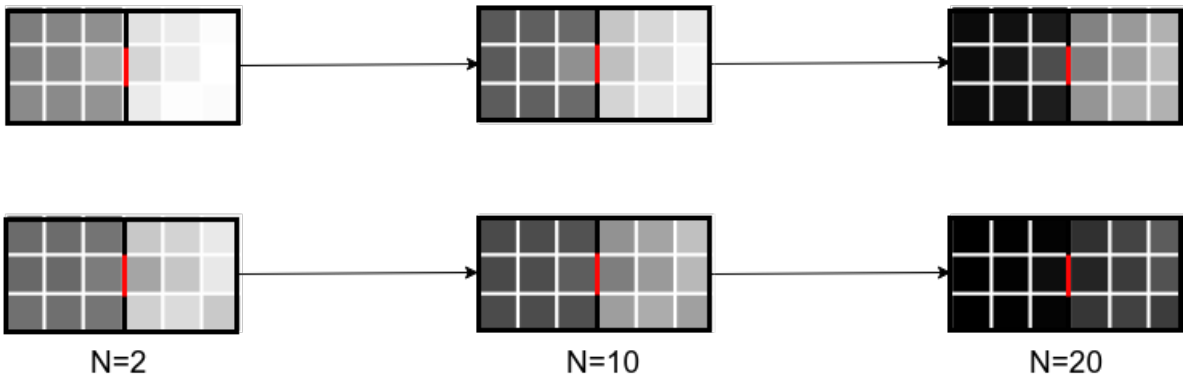


Figure 4.8: Proximity of the estimated state value to the optimal value after N trajectories. Top row is the original room environment and bottom row is the regularized one ($\beta = 0.5$). Darker is better.

Figure 4.8 shows the proximity of the estimated state value to the true value with and without temporal regularization. The darker the state, the closer it is to its true value. The heatmap scale has been adjusted at each trajectory to observe the difference between both methods. We first notice that the overall propagation of the information in the regularized MDP is faster than in the original one. In this context, information refers to value functions. We also observe that when first entering the second room, bootstrapping on values coming from the first room allows the agent to learn the true value faster.

This suggests that temporal regularization could help agents explore faster by using their prior from the previously visited state for learning the corresponding true value faster. It is also possible to consider more complex and powerful regularizers. Let us study a different time series prediction model, namely exponential averaging, as defined in (4.9). The complexity of such models is usually articulated by hyper-parameters, allowing complex models to improve performance by better adapting to problems. We illustrate this by comparing the performance of regularization using the previous state and an exponential averaging of all previous states.

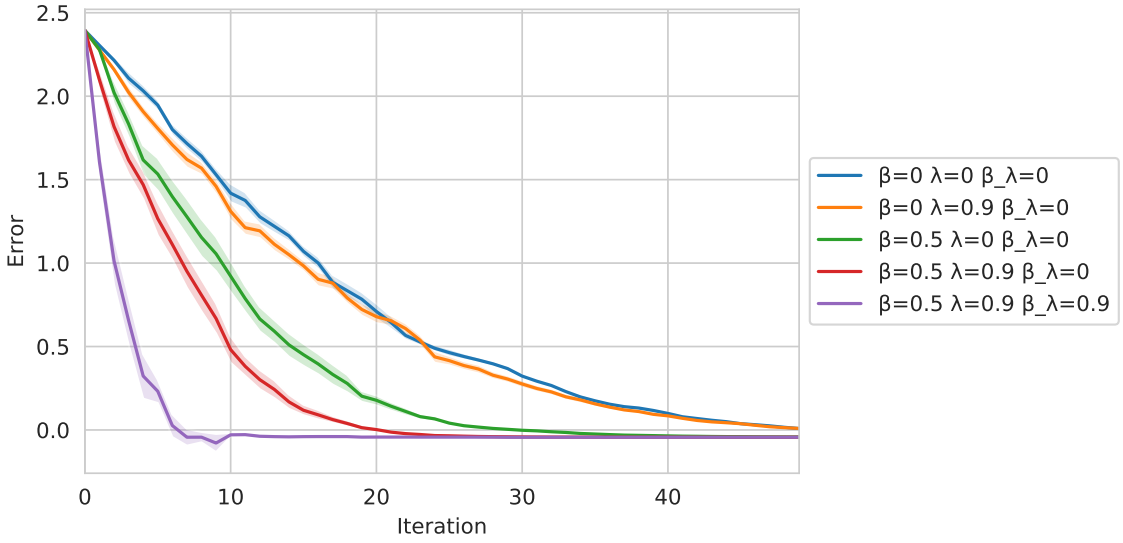


Figure 4.9: Benefits of complex regularizers on the room domain. λ refers to the λ-return coefficient and β_λ to the smoothing coefficient for exponential averaging.

Fig. 4.9 shows the average error on the value estimate using past state smoothing, exponential smoothing, without smoothing and λ-return. In this setting, exponential smoothing transfers information faster, thus enabling faster convergence to the true value. Another interesting observation from Fig. 4.9 is that TD(λ) and temporal regularization can be used together to yield better performance.

One way to interpret this results is to think in terms of prior over the value functions. By walking in the first room, the agent learns a prior value for all states that it will consider when entering the new room and bootstrap partially on previous states.

If this prior is accurate(i.e. the value in the first and second room are the same), then this will result in faster learning. However, if the underlying assumption is violated, temporal regularization will introduce some irrecoverable bias. In other words, if the value changes abruptly during the trajectory and the values are smoothed, it can introduce bias. In figure 4.10, we illustrate how smoothing temporally can introduce bias. Practically we modify the room experiment such that all the rewards in the second room now are $r = 3$. As illustrated in figure 4.10, aggressive regularization can yield higher bias.

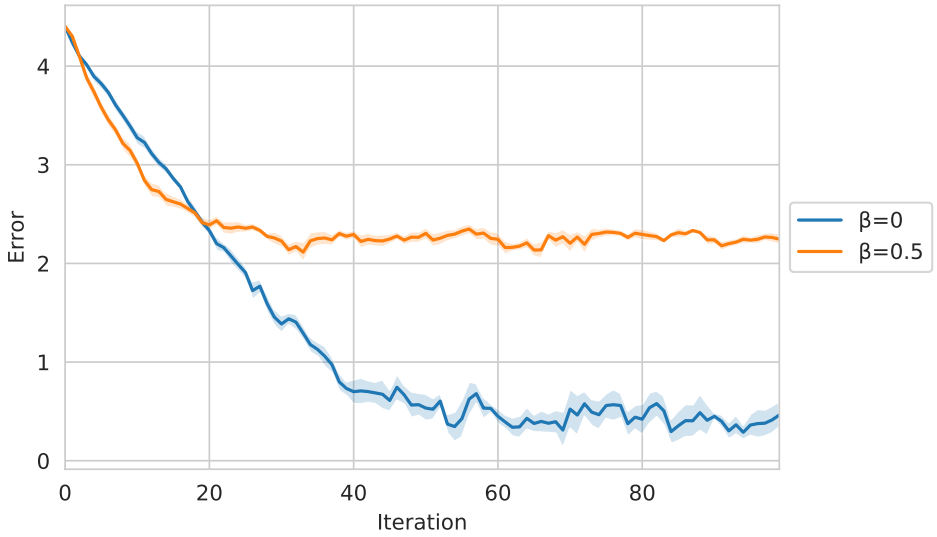


Figure 4.10: Disadvantages of temporal regularization when the value changes between room. The Y-axis represents the distance between the learned value function and the optimal one.

Those two experiments demonstrate the bias-variance trade-off induced by temporally regularizing values.

Impact of initialization: In some cases, there exists a relationship between the initialization of the value function and the performance of temporal regularization. In the room experiment, assuming the agent would quickly go in the next room, it would then rely on values of the previous room that are close to initialization. However, if changing room is hard(as is the case in the experiment), then the agent's value will not

be dependent anymore on the initialization as the values will be learned. Depending on the environment, three phenomena can interact, initialization, the validity of the smoothness assumption and temporal regularization.

4.2.5 Noisy state representation

The next experiment illustrates a strength of temporal regularization, that is its robustness to noise in the state representation. This situation can naturally arise when the state sensors are noisy or insufficient to avoid aliasing. For this task, we consider the synthetic, one dimensional, continuous setting. A learner evolving in this environment walks randomly along this line with a discount factor $\gamma = 0.95$, as shown in figure 4.11.

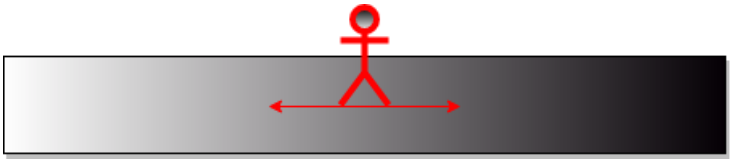


Figure 4.11: Noisy continuous random walk.

Let $x_t \in [0, 1]$ denote the position of the agent along the line at time t. The next position $x_{t+1} = x_t + a_t$, where action $a_t \sim \mathcal{N}(0, 0.05)$. The state of the agent corresponds to the position perturbed by a zero-centered Gaussian noise ϵ_t, such that $s_t = x_t + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ are i.i.d. When the agent moves to a new position x_{t+1}, it receives a reward $r_t = x_{t+1}$. The episode ends after 1000 steps. In this experiment we model the value function using a linear model with a single parameter θ. We are interested in the error when estimating the optimal parameter function θ^* with and without temporal regularization, that is θ^π_β and θ^π, respectively. In this case we use the TD version of temporal regularization presented at the end of Sec. 4.1.

Figure 4.12 shows these errors, averaged over 1000 repetitions, for different values of noise variance σ^2. We observe that as the noise variance increases, the un-regularized estimate becomes less accurate, while temporal regularization is more robust. Using

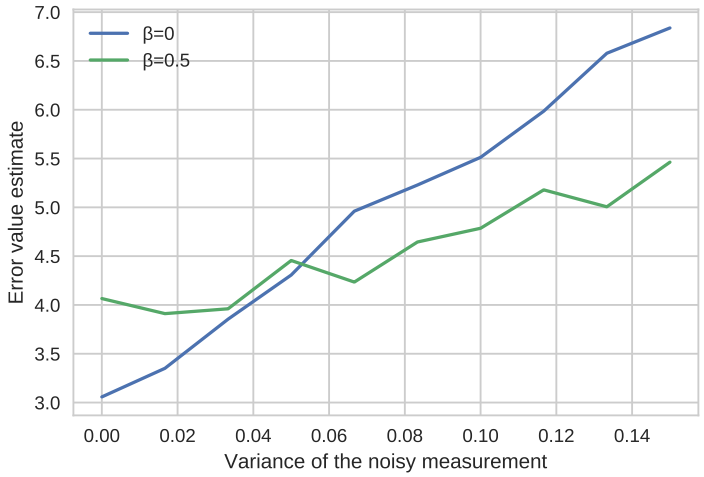


Figure 4.12: Absolute distance from the original (θ^{π}) and the regularized (θ^{π}_{β}) state value estimates to the optimal parameter θ^* given the noise variance σ^2 in state sensors.

more complex regularizer can improve performance as shown in the previous section, but this potential gain comes at the price of a potential loss in case of a model misfit.

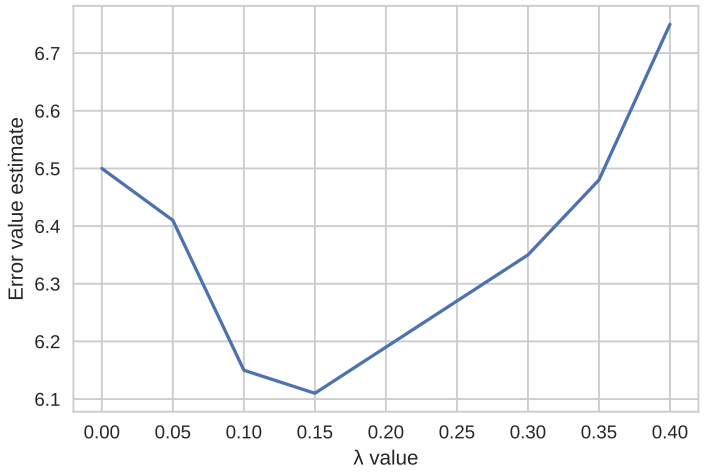


Figure 4.13: Impact of complex regularizer parameterization (λ) on the noisy walk using exponential smoothing.

Fig. 4.13 shows the absolute distance from the regularized state estimate (using exponential smoothing) to the optimal value while varying λ (higher λ = more smoothing). Increasing smoothing improves performance up to some point, but when λ is not well fit the bias becomes too strong, and performance declines. It is

a classic bias-variance tradeoff. This experiment highlights a case where temporal regularization is effective even in the absence of smoothness in the state space (which other regularization methods would target). This is further highlighted in the next experiments.

It is often assumed that the full state can be determined, while in practice, the Markov property rarely holds. It is the case, for example, when taking the four last frames to represent the state in Atari games (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015).

A problem that arises when treating a partially observable MDP (POMDP) as a fully observable is that it may no longer be possible to assume that the value function is smooth over the state space (Singh, Jaakkola, and Jordan, 1994).

For example, the observed features may be similar for two states that are intrinsically different, leading to highly different values for states that are nearby in the state space. Previous experiments on noisy state representation (Sec. 4.2.5) and on the Atari games (Sec. 4.3.2) show that temporal regularization provides robustness to those cases. This makes it an appealing technique in real-world environments, where it is harder to provide the agent with the full state.

4.3 Control experiments

In the previous sections, we demonstrate how temporal regularization can be used for policy evaluation to reduce variance. While we briefly discussed regularizing actions temporally, we do not theoretically nor experimentally explore these directions in this thesis and leave it as future work. In practice, this restricts the class of algorithm we can consider for control as we can not use Q-values. If the agent has access to the dynamics of the environment, it is possible to use value functions to select the optimal actions by marginalizing over all possible future states. We first study a simple toy control experiment with known dynamics to analyze the impact of temporal

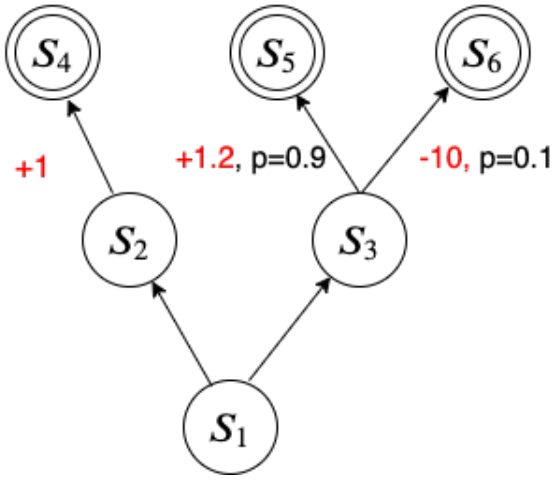


Figure 4.14: Toy control experiment.

regularization in control. However, in practice, the agent does not know the dynamics of the environment. This is why, for the majority of the experiments, we consider actor-critic methods. For temporal value regularization to be useful for actor-critic methods, the argument is that better value estimation can lead to faster convergence of the policy to the optimal policy. In the next sections, we present a mix of positive and negative results using temporal regularization for continuous and discrete control. The results suggest that temporal regularization may be useful to improve convergence speed in some discrete settings but hinders learning in continuous environments.

4.3.1 Toy control experiment

In this experiment, we demonstrate that better value estimation can lead to better decision making. In particular, we extend the experiment described in section 4.2.3 and consider the MDP described in figure 4.14. The states S_4, S_5, S_6 are terminal states. At state S_1, the agent can choose between going left or going right and the transition is deterministic. At state S_3 the agent has a probability of 0.1 to go left and 0.9 to go right regardless of the action. The exploration policy of the agent follows a

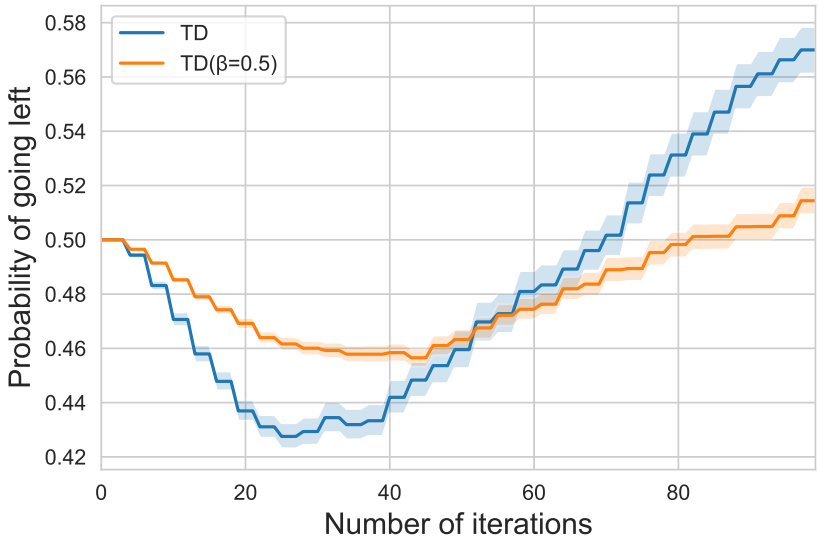


Figure 4.15: Probability of going left to S_2 during training on the toy control experiment.

Boltzmann distribution such that for S_1:

$$p(s_2|s_1) = \frac{e^{\frac{V(S_2)}{\tau}}}{e^{\frac{V(S_2)}{\tau}} + e^{\frac{V(S_3)}{\tau}}}$$
$$p(s_3|s_1) = \frac{e^{\frac{V(S_3)}{\tau}}}{e^{\frac{V(S_2)}{\tau}} + e^{\frac{V(S_3)}{\tau}}} \tag{4.14}$$

where τ is the temperature parameter. The only decision point is S_1. This MDP could also be phrased as a bandit problem. In this experiment the left branch has the highest reward, however when first exploring the environment the agent will believe that the right branch is better until it falls on the bad state S_6. In this setting temporally regularizing the value S_3 with $V(S_1)$ will prevent the agent from being overoptimistic about this path. With sufficient exploration, the agent converges to the right solution(left branch). The Figure 4.14 illustrates this phenomena by showing the probability of going left from state S_1.

While the agent is stuck in the local optima, temporal regularization will improve performance by encouraging exploration. Effectively this is done by bringing the value of S_3 closer to the value of S_1. However, when the agent figures out that S_2 is actually better, the regularization actually hinders learning. The update on $V(S_2)$

is more conservative due to the regularization. In this setting, due to Boltzmann exploration, there is an intimate relationship between temporal regularization and exploration. Regularizing the value will have a tendency to yield more exploratory behaviour. Similar results could be obtained using a state-dependent exploration or learning rate. However, in practice, those methods differ as temporal regularization will only occur along the trajectory. In contrast, changing the exploration coefficient or learning rate will have a global effect on the learning process.

The learning rate used is 0.5, the exploration temperature $\tau = 1$ and the temporal regularization $\beta = 0.5$. The results are averaged over 1000 trials. In this experiment, the comparison with TD(λ) yields a similar result due to the fact that S_4, S_5, S_6 are terminal states.

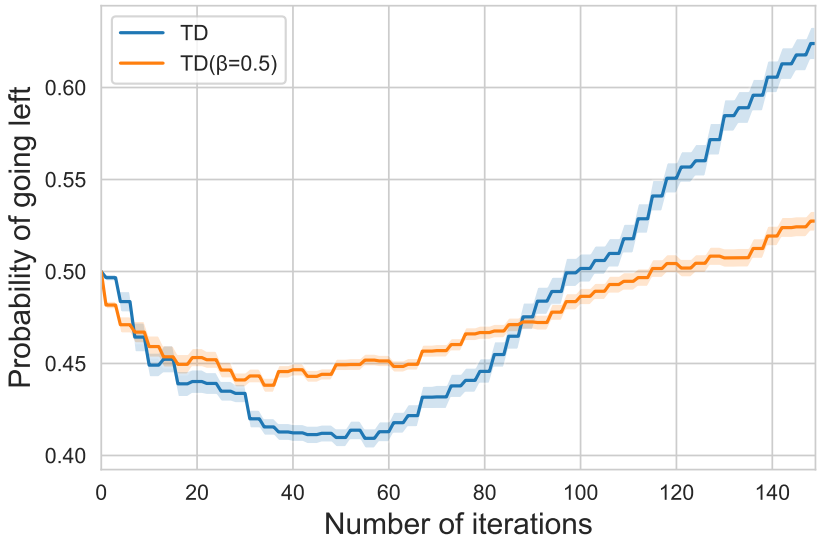


Figure 4.16: Probability of going left to S_2 during training on the toy control experiment with optimistic initialization.

In the previous experiment, the values were initialized to zero. We also explore the impact of initialization on the results observed. In particular, we explore whether the previous results are an artefact of the initialization scheme. Figure 4.16 demonstrates that even when optimistically initializing the values (to 5) the results remains the same. This can be explained as follows: both values $V(S_1)$ and $V(S_2)$ will get smaller

than $V(S_3)$ until the agent explores S_6. This means that temporally regularizing $V(S_3)$ using $V(S_1)$ will yield more conservative value update and exploratory behaviour.

4.3.2 Deep reinforcement learning

To showcase the potential of temporal regularization in high dimensional settings, we adapt an actor-critic based method (PPO (Schulman, Wolski, et al., 2017)) using temporal regularization. More specifically, we incorporate temporal regularization as exponential smoothing in the target of the critic. PPO uses the general advantage estimator $\hat{A}_t = \delta_t + \gamma\lambda\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_T$ where $\delta_t = r_t + \gamma v(s_{t+1}) - v(s_t)$. We regularize δ_t such that $\delta_t^\beta = r_t + \gamma((1-\beta)v(s_{t+1}) + \beta\widetilde{v}(s_{t-1}))) - v(s_t)$ using exponential smoothing $\widetilde{v}(s_t) = (1-\lambda)v(s_t) + \lambda\widetilde{v}(s_{t-1})$ as described in Eq. (4.9). $\widetilde{v}$ is an exponentially decaying sum over all t previous state values encountered in the trajectory. We evaluate the performance in the Arcade Learning Environment (Bellemare et al., 2013), where we consider the following performance measure:

$$\frac{\text{regularized} - \text{baseline}}{\text{baseline} - \text{random}}. \tag{4.15}$$

The hyper-parameters for the temporal regularization are $\beta = \lambda = 0.2$ and a decay of $1e^{-5}$. Those are selected on 7 games and 3 training seeds. All other hyper-parameters correspond to the one used in the PPO paper. Our implementation[1] is based on the publicly available OpenAI codebase (Dhariwal et al., 2017). The previous four frames are considered as the state representation (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015). For each game, 10 independent runs (10 random seeds) are performed.

The results reported in Figure 4.17 show that adding temporal regularization improves the performance on multiple games. This suggests that the regularized optimal value function may be smoother and thus easier to learn, even when using function approximation with deep learning. Also, as shown in previous experiments (Sec. 4.2.5), temporal regularization being independent of spatial representation makes

[1]The code can be found https://github.com/pierthodo/temporal_regularization.

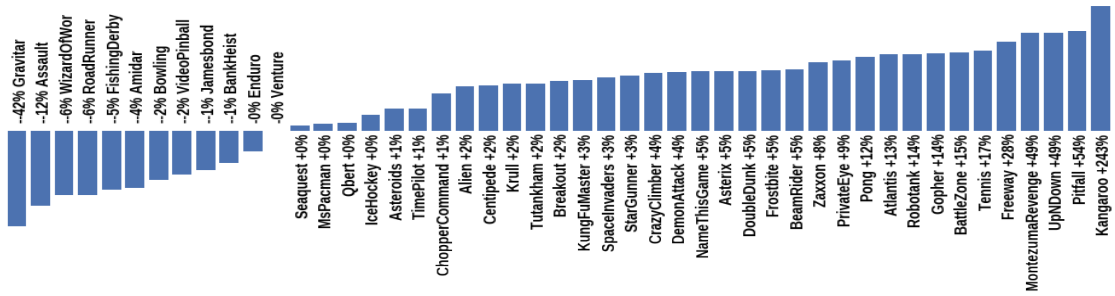


Figure 4.17: Performance (Eq. 4.15) of a temporally regularized PPO on a suite of Atari games.

it more robust to misspecification of the state features, which is a challenge in some of these games (e.g., when assuming full state representation using some previous frames).

The full performance for each game can be found in figure 5.10.

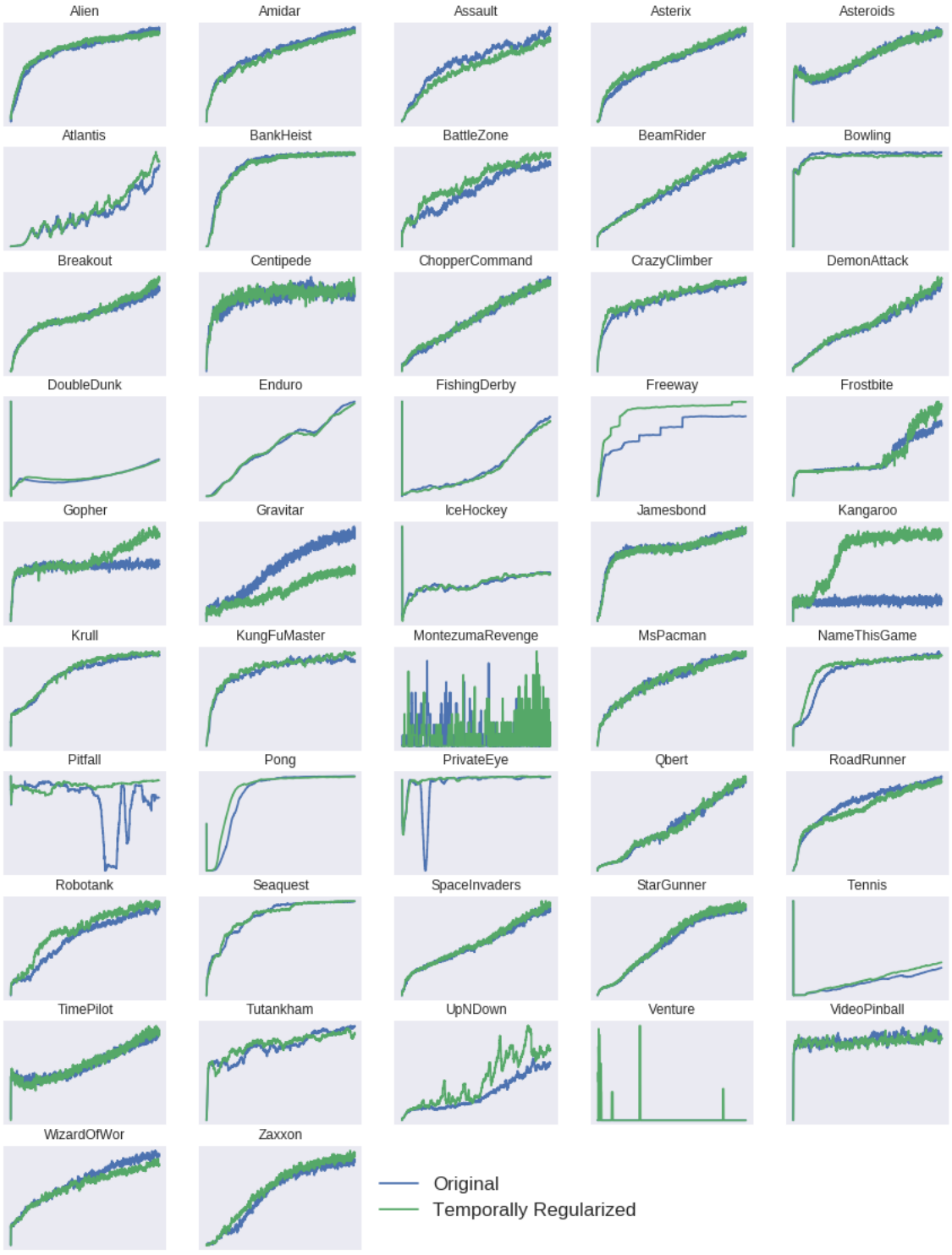


Figure 4.18: Average reward per episode on Atari games.

Previous work (Laroche, 2018) looked at how the smoothness of the objective function relates to the convergence speed of RL algorithms. An analogy can be drawn with convex optimization where the rate of convergence is dependent on the Lipschitz (smoothness) constant (Boyd and Vandenberghe, 2004). By smoothing the value temporally, we hypothesize that the optimal value function can be smoother. It would be beneficial in high-dimensional state space where the use of deep neural network is required. This could explain the performance displayed using temporal regularization on Atari games (Sec. 4.3.2).

4.3.3 Negative results on continuous control

In this section, we evaluate the potential of Temporal Regularization in a continuous control setting (Todorov, Erez, and Tassa, 2012). In a similar manner than the last section, we modify a PPO architecture to introduce Temporal Regularization. We also evaluate the robustness of different algorithms by adding ϵ sensor noise (drawn from a normal distribution $\epsilon \sim N(0, 1)$) to the observations as presented in (Zhang, Ballas, and Pineau, 2018). The first experiment considers the popular continuous benchmark, namely, Cartpole. The goal is to balance a pole for 200 time steps. As we can observe in Figure 4.19 temporal regularization hinders learning in the noiseless setting and perform similarly to PPO in the noisy setting. We also benchmark temporal regularization in more complex continuous control tasks(Mujoco). In this setting either, temporal regularization does not appear to perform well even when Gaussian noise is introduced in the observation. We hypothesize that the variance of the value function is not a *big enough* issue compared to the bias induced by smoothing *important states*. The bias induced by the regularization parameter β can be detrimental for the learning in the long run. A first attempt to mitigate this bias is to decay the regularization as learning advances, as it is done in the deep learning experiment (Sec. 4.3.2). However, this proved to be unsuccessful in continuous control

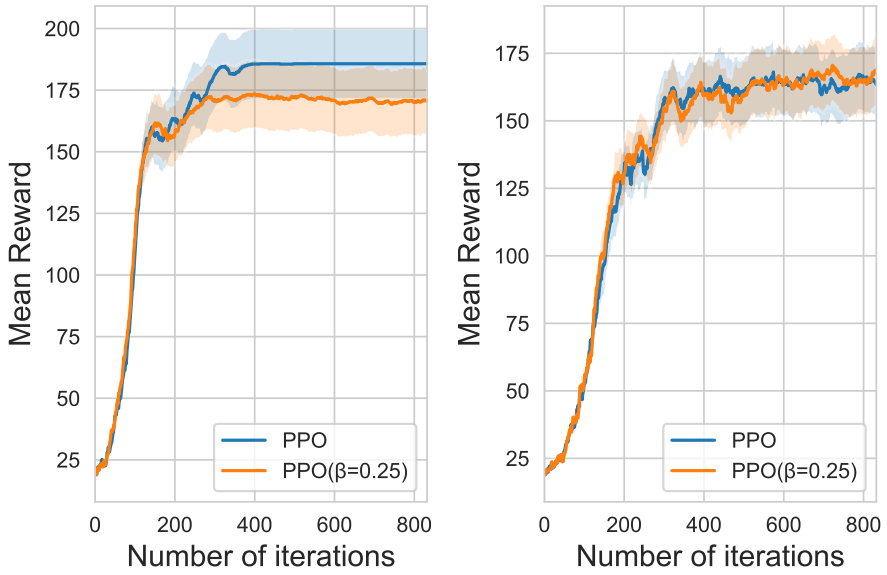


Figure 4.19: Performance of PPO with and without regularization on cartpole. The left graph is without noise and the right one with noise $\epsilon \sim N(0, 1)$.

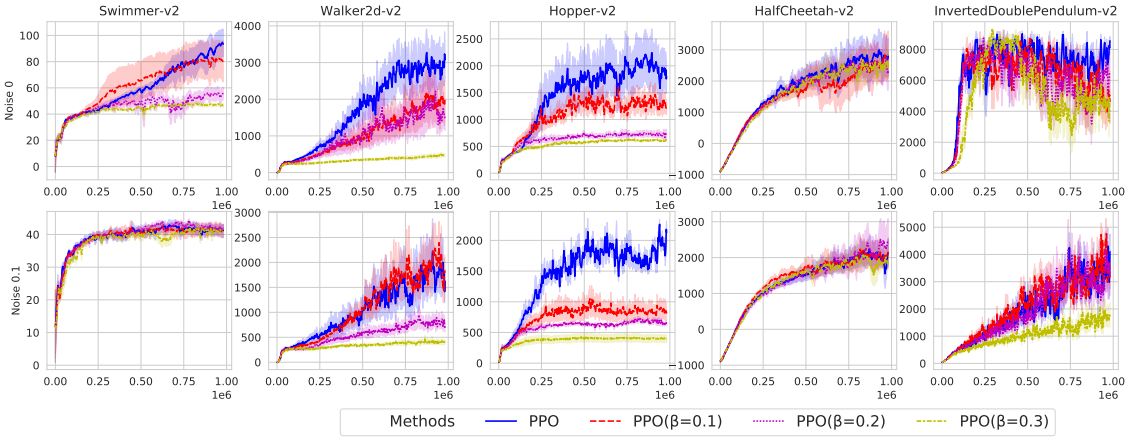


Figure 4.20: Performance on Mujoco tasks. Results on the first row are generated without noise and on the second row by inducing a Gaussian noise ($\epsilon \sim \mathbb{N}(0, 0.1)$) in the sensor inputs.

tasks.

The results throughout this section are averaged over 20 random seeds and $\beta = 0.25$ for temporal regularization.

Indeed in Temporal regularization, there is no way to decide which state to smooth or not. For example, when falling off a cliff, one would want an algorithm that decides not to smooth the estimates. This issue is tackled in the next chapter of this thesis by

introducing an algorithm that can learn such a state-dependent smoothing coefficient.

5

Recurrent Value Function

In the previous chapter, we regularize the value estimate by adding a regularization term to the objective (target). In this chapter, we explore explicitly averaging the value estimate using exponential smoothing.

The problem of disentangling signal from noise in sequential domains is not specific to Reinforcement Learning and has been extensively studied in the Supervised Learning literature. In this work, we leverage ideas from time series literature (Brockwell, Davis, and Fienberg, 1991; Brockwell, Davis, and Calder, 2002) and Recurrent Neural Networks (Hochreiter and Schmidhuber, 1997) to address the robustness of value functions in Reinforcement Learning. We propose Recurrent Value Functions (RVFs): an exponential smoothing of the value function. The value function of the current state is defined as an exponential smoothing of the values of states visited along the trajectory where the value function of past states is summarized by the previous RVF.

However, exponential smoothing along the trajectory can result in bias when the value function changes dramatically through the trajectory (non-stationarity). This is an issue encountered in the last chapter with Temporal Regularization. To alleviate this issue, we propose to use exponential smoothing on value functions using a trainable state-dependent emphasis function which controls the smoothing coefficients. Intuitively, the emphasis function adapts the amount of emphasis required on the current value function and the past RVF to reduce bias with respect to the optimal

value estimate. In other words, the emphasis function identifies important states in the environment. An important state can be defined as one where *its value differs significantly from the previous values along the trajectory.* For example, when falling off a cliff, the value estimate changes dramatically, making states around the cliff more salient. This emphasis function serves a similar purpose to a gating mechanism in a Long Short Term Memory cell of a Recurrent Neural Network (Hochreiter and Schmidhuber, 1997).

5.1 Recurrent Value Functions (RVFs)

As mentioned earlier, performance of value-based methods are often heavily impacted by the quality of the data obtained (Fox, Pakman, and Tishby, 2015; Pendrith, 1994). For example, in robotics, noisy sensors are common and can significantly hinder performance of popular methods (Romoff et al., 2018). In this work, we propose a method to improve the robustness of value functions by estimating the value of a state s_t using the estimate at time step t and the estimates of previously visited states s_i where $i < t$.

5.1.1 Algorithm

Let's define the trajectory up until time t as $\tau = \{s_0, s_1, ..., s_t\}$. Mathematically, the Recurrent Value Function (RVF) of a state s at time step t is given by:

$$\begin{aligned} V^{\beta}(s_t, \tau) &= \beta(s_t)V(s_t) + (1 - \beta(s_t))V^{\beta}(s_{t-1}, \tau), \\ V^{\beta}(s_0, \tau) &= V(s_0) \end{aligned} \tag{5.1}$$

where $\beta(s_t) \in [0, 1]$. V^{β} estimates the value of a state s_t as a convex combination of current estimate $V(s_t)$ and previous estimate $V^{\beta}(s_{t-1}, \tau)$. $V^{\beta}(s_{t-1}, \tau)$ can be recursively expanded further, hence the name Recurrent Value Function. β is the emphasis function which updates the recurrent value estimate. To initialize the

recursion we set the value of the first state as $V^{\beta}(s_0, \tau) = V(s_0)$. This is equivalent to saying $\beta(s_0) = 1$. The definition of $V^{\beta}(s_t)$ actually depends on the entire trajectory τ. To simplify the notation throughout this chapter we will omit the τ and define $V^{\beta}(s_t, \tau) = V^{\beta}(s_t)$. In contrast to traditional methods that attempt to minimize Eq. 3.16, the goal here is to find a set of parameters θ, ω that minimize the following error:

$$\begin{aligned} \mathcal{L}(\theta, \omega) &= \mathop{\mathbb{E}}_{\pi}[(\widetilde{V}^{\pi} - V^{\beta}_{\theta,\omega})^2], \\ V^{\beta}_{\theta,\omega}(s_t) &= \beta_{\omega}(s_t)V(s_t) + (1 - \beta_{\omega}(s_t))(V^{\beta}_{\theta,\omega}(s_{t-1})), \end{aligned} \tag{5.2}$$

where V is a function parametrized by θ, and β_{ω} is a function parametrized by ω. This error is similar to the traditional error in Eq. 3.16, but we replace the value function with $V^{\beta}_{\theta,\omega}$. In practice, $\widetilde{V}^{\pi}$ can be any target such as TD(0), TD(N), TD(λ) or Monte Carlo (R. S. Sutton and Barto, 1998). In this thesis, for temporal difference methods, we use V_{θ} as a bootstrap value function in the target. Another possibility would be to replace V_{θ} in the target by $V^{\beta}_{\theta,\omega}$ directly. With a fixed β this would be equivalent to using temporal regularization. If β are adjusted online, this could also have a similar effect than reward shaping or state-dependent λ. We did not explore this possibility in this thesis and left this for future work. One of the reasons we did not explore this avenue is that adapting β will effectively change the solution of the MDP and might lead to unstable learning.

We minimize Eq. 5.2 by updating θ and ω using the semi-gradient technique which results in the following update rule:

$$\begin{aligned} \theta &= \theta + \alpha\delta_t \nabla_{\theta} V^{\beta}_{\theta,\omega}(s_t), \\ \omega &= \omega + \alpha\delta_t \nabla_{\omega} V^{\beta}_{\theta,\omega}(s_t), \end{aligned} \tag{5.3}$$

where $\delta_t = \widetilde{V}^{\pi}(s_t) - V^{\beta}_{\theta,\omega}(s_t)$ is the TD error with RVF in the place of the usual value function. The complete algorithm using the above update rules can be found in Algorithm 8. We present RTD using TD(0) as a target.

Algorithm 8 Recurrent Temporal Difference(0)

Input: π,γ,θ,ω
Initialize: $V^{\beta}_{\theta,\omega}(s_0) = V(s_0)$
OUTPUT: θ, ω
for t **do**
    Take action $a \sim \pi(s_t)$, observe $r(s_t), s_{t+1}$
    $V^{\beta}_{\theta,\omega}(s_t) = \beta_\omega(s_t)V(s_t) + (1 - \beta_\omega(s_t))V^{\beta}_{\theta,\omega}(s_{t-1})$
    $\delta_t = r(s_t) + \gamma V(s_{t+1}) - V^{\beta}_{\theta,\omega}(s_t)$
    $\theta = \theta + \alpha\delta_t \nabla_\theta V^{\beta}_{\theta,\omega}(s_t)$
    $\omega = \omega + \alpha\delta_t \nabla_\omega V^{\beta}_{\theta,\omega}(s_t)$
end for

5.1.2 Learning β

As discussed earlier, β_ω learns to identify states whose value significantly differs from previous estimates. While optimizing for the loss function described in Eq. 5.2, the $\beta_\omega(s_t)$ learns to bring the RVF $V^{\beta}_{\theta,\omega}$ closer to the target $\widetilde{V}^{\pi}$. It does so by placing greater emphasis on whichever is closer to the target, either $V(s_t)$ or $V^{\beta}_{\theta,\omega}(s_{t-1})$. Concisely, the updated behavior is split into four scenarios Table 5.1. Intuitively, if the past does not align with the future, β will emphasize the present. Likewise, if the past aligns with the future, then β will place less emphasis on the present. This behavior is further explored in the experimental section.

Table 5.1: Behaviour of β based on the loss

	$\widetilde{V}^{\pi}(s_t) > V^{\beta}_{\theta,\omega}(s_t)$	$\widetilde{V}^{\pi}(s_t) < V^{\beta}_{\theta,\omega}(s_t)$
$V(s_t) > V^{\beta}_{\theta,\omega}(s_{t-1})$	$\beta \uparrow$	$\beta \downarrow$
$V(s_t) < V^{\beta}_{\theta,\omega}(s_{t-1})$	$\beta \downarrow$	$\beta \uparrow$

Note that, the gradients of $V^{\beta}_{\theta,\omega}$ take a recursive form (gradient through time) as shown in Eq. 5.4. The gradient form is similar to LSTM (Hochreiter and Schmidhuber, 1997), and GRU (J. Chung et al., 2014) where β acts as a gating mechanism that controls the flow of gradient. LSTM uses a gated exponential smoothing function on the hidden representation to assign credit more effectively. In contrast, we propose to exponentially smooth the outputs (value functions) directly rather than the hidden state.

This gradient can be estimated using backpropagation through time by recursively applying the chain rule where:

$$\nabla_\theta V^\beta_{\theta,\omega}(s_t) = \beta_\omega(s_t)\nabla_\theta V(s_t) + (1 - \beta_\omega(s_t)).\nabla_\theta V^\beta_{\theta,\omega}(s_{t-1}), \tag{5.4}$$

we can control the flow of gradient by using emphasis function $\beta_\omega(s_t)$ and pass gradient to the states that contributed to the reward but are located several time-steps earlier. We could potentially do credit assignment on states that are temporally far away by forcing the emphasis function between these states to be close to 0. This setting could be useful in problems with long horizons, such as lifelong learning and continual learning. However, calculating this gradient can become computationally expensive in environments with a large episodic length, such as continual learning. Therefore, we could approximate the gradient $\nabla_\theta V^\beta_{\theta,\omega}(s_t)$ using a recursive *eligibility* trace:.

$$e_t = \beta_\omega(s_t)\nabla_\theta V(s_t) + (1 - \beta_\omega(s_t))e_{t-1}. \tag{5.5}$$

This induces a bias as the gradient stored in the trace is with respect to the weights that are continuously modified. This bias is of a similar nature to the one encountered in Real Time Recurrent Learning (Williams and Zipser, 1995) and the online-backward implementation of the lambda-return (R. S. Sutton, 1985). A more in-depth discussion of this bias can be found in (Seijen and R. Sutton, 2014; Williams and Zipser, 1995).

5.1.3 Adjusting for the reward:

In practice, some environments in Reinforcement Learning have a constant reward at every time step, potentially inducing bias in $V^\beta_{\theta,\omega}$ estimates. It would be possible to modify the RVF formulation to account for the immediate reward, such that:

$$V^\beta_{\theta,\omega}(s_t) = \beta V(s_t) + (1 - \beta)(V^\beta_{\theta,\omega}(s_{t-1}) - r_{t-1}). \tag{5.6}$$

Whether or not subtracting the reward can reduce the bias will depend on the environment considered. In Natural Value Approximators (Z. Xu et al., 2017) they

consider a similar quantity but also adjust for the discount factor γ. This adjustment intuitively could be useful but can induce divergence in many cases with a small γ. In the following section, we present the asymptotic convergence proof of RVF.

5.1.4 Asymptotic convergence

For this analysis, we consider the simplest case: a tabular setting with TD(0) and a fixed set of β. In the tabular setting, each component of θ and ω estimates one particular state, allowing us to simplify the notation. In this section, we simplify the notation by dropping θ and ω such that $V(s_t) = V(s_t)$ and $\beta_\omega(s_t) = \beta_t$. In the tabular setting, convergence to the fixed point of an operator is proved by casting the learning algorithm as a stochastic approximation (Tsitsiklis, 1994; Borkar, 2009; Borkar and Meyn, 2000) of the form:

$$\theta_{t+1} = \theta_t + \alpha(\mathcal{T}\theta_t - \theta_t + w(t)), \tag{5.7}$$

where $\mathcal{T} : \mathbb{R}^{|\mathbb{S}|} \to \mathbb{R}^{|\mathbb{S}|}$ is a contraction operator and $w(t)$ is a noise term. The main idea is to cast the Recurrent Value Function as an asynchronous stochastic approximation (Tsitsiklis, 1994) with an additional regularization term. By bounding the magnitude of this term, we show that the operator is a contraction. The algorithm is asynchronous because the eligibility trace only updates certain states at each time step.

We consider the stochastic approximation formulation described in Eq. 5.7 with the following operator $\mathcal{T}^\beta : \mathbb{R}^{|\mathbb{S}|} \to \mathbb{R}^{|\mathbb{S}|}$ for any $i \leq t$:

$$\mathcal{T}^\beta V(s_i) = \mathop{\mathbb{E}}_{\pi}[r_t + \gamma V(s_{t+1}) + \Delta_t(s_i)] \tag{5.8}$$

for all states s_i with $\beta_i \in (0, 1]$. $\Delta_t(s_i)$ can be interpreted as a regularization term composed of the difference between $V(s_i)$ and $V^\beta(s_t)$.

To obtain this operator we first examine the update to $V(s_i)$ made during the trajectory

at time step t:

$$\begin{aligned} V(s_i) &= V(s_i) + \alpha e_t(s_i)(r_t + \gamma V(s_{t+1}) - V^{\beta}(s_t)) \\ &= V(s_i) + \alpha e_t(s_i)(r_t + \gamma V(s_{t+1}) + \Delta_t(s_i) - V(s_i)), \end{aligned} \tag{5.9}$$

where $\Delta_t(s_i) = (1 - C_t(s_i))(V(s_i) - \widetilde{V}_t(s_i))$ and $C_t(s_i) = \beta_i \prod_{p=i+1}^{t}(1-\beta_p)$. $\widetilde{V}_t(s_t)$ is a convex combination of all V encountered in the trajectory, with the exception of $V(s_i)$, weighted by their respective contribution(β) to the estimate $V^{\beta}(s_t)$. For example, if we consider updating $V(s_2)$ at $t = 3$ and have the following $\beta_1 = 0.9, \beta_2 = 0.1, \beta_3 = 0.1$, the value of $\widetilde{V}_3(s_2)$ will be mainly composed of $V(s_1)$. The main component of the error will be $r_t + \gamma V(s_4) - V(s_1)$. We take an example with $t = 3$ and consider $i = 2$:

$$\begin{aligned} V^{\beta}(s_3) &= \beta_3 V(s_3) + (1-\beta_3)\beta_2 V(s_2) + (1-\beta_3)(1-\beta_2)V(s_1) \\ &= V(s_2) - (1 - (1-\beta_3)\beta_2)(V(s_2) - \frac{\beta_3 V(s_3) + (1-\beta_3)(1-\beta_2)V(s_1)}{(1-(1-\beta_3)\beta_2)}) \\ &= V(s_2) - (1 - (1-\beta_3)\beta_2)(V(s_2) - \widetilde{V}_t(s_i)) \end{aligned} \tag{5.10}$$

$\widetilde{V}$ is a convex combination of all the V encountered along the trajectory weighted by β apart from $V(s_2)$. This can be observed mathematically:

$$\begin{aligned} &\frac{\beta_3 + (1-\beta_3)(1-\beta_2)}{(1-(1-\beta_3)\beta_2)} = 1 \\ &\equiv \beta_3 + (1-\beta_3)(1-\beta_2) = (1-(1-\beta_3)\beta_2) \\ &\equiv \beta_3 + (1-\beta_3)\beta_2 + (1-\beta_3)(1-\beta_2) = 1 \end{aligned} \tag{5.11}$$

where the last line is true because $\beta \in (0, 1]$ In practice, one can observe an increase in the magnitude of this term with a decrease in *eligibility*. This suggests that the biased updates contribute less to the learning. Bounding the magnitude of Δ to ensure contraction is the key concept used in this work to ensure asymptotic convergence.

Assuming a fixed β for every state, this Δ can be interpreted as a temporal regularization factor. It is possible to decompose the update based on the reversal Markov chain similarly to the last chapter. The main difference, however, is when learning β, RVFs have the capacity of RVFs to emphasize important(high β) state.

We consider the following assumptions to prove convergence: The first assumption deals with the ergodic nature of the Markov chain. It is a common assumption in theoretical Reinforcement Learning that guarantees an infinite number of visits to all states, thereby avoiding chains with transient states (Mahadevan, 1996).

Assumption 2. *The Markov chain is ergodic.*

The second assumption concerns the relative magnitude of the maximum and minimum reward and allows us to bound the magnitude of the regularization term.

Assumption 3. *We define $R_{\max}$ and $R_{\min}$ as the maximum and minimum reward in an MDP. All rewards are assumed to be positive and scaled in the range $[R_{\min}, \widetilde{R}_{\max}]$ such that the scaled maximum reward $\widetilde{R}_{\max}$ satisfies the following:*

$$D\widetilde{R}_{max} \leq R_{min}, \qquad D > \gamma, \tag{5.12}$$

where $D \in (0.5, 1]$ is a constant to be defined based on γ.

In theory, scaling the reward is reasonable as it does not change the optimal solution of the MDP (Hasselt et al., 2016). In practice, however, this may be constraining as the range of the reward may not be known beforehand. It is possible to relax this assumption by considering the trajectory's information to bound Δ. As an example, one could consider any physical system where transitions in the state space are smooth (continuous state space) and bounded by some Lipschitz constant in a similar manner than (Shah and Xie, 2018).

As mentioned earlier, the key component of the proof is to control the magnitude of the term in Eq. 5.9: $\Delta_t(s_i) = (1 - C_t(s_i))(V(s_i) - \widetilde{V}_t(s_i))$. As the eligibility of this update gets smaller, the magnitude of the term gets bigger. This suggests that not updating certain states whose eligibility is less than the threshold C can help mitigate biased updates. Depending on the values of γ and D, we may need to set a threshold C to guarantee convergence.

Theorem 3. *Define* $V_{\max} = \frac{\widetilde{R}_{\max}}{1-(\gamma+(1-D))}$ *and* $V_{\min} = \frac{R_{\min}}{1-(\gamma-(1-D))}$. $\mathcal{T}^\beta : X \to X$ *is a contraction operator if the following holds:*

- *Let* X *be the set of* V *functions such that* $\forall s \in \mathbb{S} \quad V_{\min} \leq V(s) \leq V_{\max}$. *The functions* V *are initialized in* X.
- *For a given* D *and* γ *we select* C *such that* $\Delta \leq (1-C)(V_{\max}-V_{\min}) \leq (1-D)V_{\min}$.

Proof. The first step is to prove that $\mathcal{T}^\beta$ maps to itself for any noisy update $\widetilde{\mathcal{T}^\beta}$. From 2) we know that $(1-C)(V_{\max}-V_{\min}) < DV_{\min} \leq DV_{\max}$ we can then deduce that

$$\begin{aligned}\widetilde{\mathcal{T}^\beta}V(s) &\leq \widetilde{R}_{\max} + \gamma V_{\max} + (1-C)(V_{\max}-V_{\min}) \\ &\leq \widetilde{R}_{\max} + (\gamma + (1-D))V_{\max} \\ &\leq V_{\max}\end{aligned} \tag{5.13}$$

and

$$\begin{aligned}\widetilde{\mathcal{T}^\beta}V(s) &\geq R_{\min} + \gamma V_{\min} + (1-C)(V_{\min}-V_{\max}) \\ &\geq R_{\min} + (\gamma - (1-D))V_{\min} \\ &\geq V_{min}\end{aligned} \tag{5.14}$$

The next step is to show that $\mathcal{T}^\beta$ is a contractive operator:

$$\begin{aligned}&\left\|\mathcal{T}^\beta V - \mathcal{T}^\beta U\right\|_\infty \\ &\leq \max_{s,s'} \mathop{\mathbb{E}}_{\pi}[\gamma V(s) + \Delta^V(s') - (\gamma U(s) + \Delta^U(s'))] \\ &\leq \max_{s,s'} \mathop{\mathbb{E}}_{\pi}[\gamma(V(s)-U(s)) + (1-D)(V(s')-U(s'))] \\ &\leq \max_{s} \mathop{\mathbb{E}}_{\pi}[((1-D)+\gamma)(V(s)-U(s))] \\ &\leq ((1-D)+\gamma)\left\|V-U\right\|_\infty\end{aligned} \tag{5.15}$$

and from the assumption we know that $(1-D)+\gamma < 1$. □

We can guarantee that V converges to a fixed point of the operator $\mathcal{T}^\beta$ with probability $= 1$ using Theorem 3 of (Tsitsiklis, 1994). We now discuss the assumptions of theorem 3 in (Tsitsiklis, 1994)

Assumption 1: Allows for delayed update that can happen in distributed system for example. In this algorithm all V's are updated at each time step t and is not an issue here.

Assumption 2: As described by (Tsitsiklis, 1994) assumption 2 "allows for the possibility of deciding whether to update a particular component x_i at time t, based on the past history of the process.". This assumption is defined to accommodate for ϵ-greedy exploration in Q-learning. In this work we only consider policy evaluation hence this assumptions holds.

Assumption 3: The learning rate of each state $s \in \mathbb{S}$ must satisfy Robbins Monroe conditions (Robbins and Monro, 1951) such that there exists $C \in \mathbb{R}$:

$$\begin{aligned} \sum_{i=0}^{\infty} \alpha_t(s) e_t(s) &= \infty \quad \text{w.p.1} \\ \sum_{i=0}^{\infty} (\alpha_t(s) e_t(s))^2 &\leq C \end{aligned} \tag{5.16}$$

This can be verified by assuming that each state gets visited infinitely often and an appropriate decaying learning rate based on $\#_s$ (state visitation count) is used (linear for example).

Assumption 5: This assumption requires $\mathcal{T}$ to be a contraction operator. This has been proven in Theorem 3 of this thesis.

To select C based on γ and D it suffice to solve analytically for:

$$\begin{aligned} & (1-C)(V_{\max} - V_{\min}) \leq (1-D)V_{\min} \\ & \equiv (1-C)\frac{\widetilde{R}_{\max}}{1-(\gamma+(1-D)} \leq ((1-D)+(1-C))\frac{R_{\min}}{1-(\gamma-(1-D)} \\ & \equiv \frac{(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)}\widetilde{R}_{\max} \leq R_{\min} \\ & \equiv \frac{D(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)}R_{\min} \leq R_{\min} \end{aligned} \tag{5.17}$$

which is satisfied only if:

$$\frac{D(1-C)(1-(\gamma-(1-D)))}{(1-(\gamma+(1-D))((1-D)+(1-C)} \leq 1. \tag{5.18}$$

As an example for $D = 0.8$ and $\gamma = 0.5$ any $C \geq 0.33$ satisfies this inequality.

5.1.5 Complex time-series model

As mentioned earlier in this thesis for temporal regularization, one could use more complex time series model such as ARIMA (Makridakis, Wheelwright, and Hyndman, 2008) to estimate the value function directly. For example, Kallman filter (Kalman, 1960; Welch and Bishop, 1995) is one of the most widely used models in the real world and could be an interesting model to consider. In this thesis, we chose for exponential smoothing for its simplicity.

5.1.6 Related work

As we discussed earlier, RVFs can be viewed as temporal regularization when used with fixed β. However, when learning β, RVFs have the capacity to emphasize important state. As a result of modifying the estimate, RVFs can choose to ignore a gradient while updating, which is not possible in other works. For example, in settings where the capacity is limited, updating on noisy states can be detrimental for learning. One important similarity of RVFs is with respect to the online implementation of λ return (R. S. Sutton and Barto, 1998; Dayan, 1992). Both RVF and online λ returns have an eligibility trace form, but the difference is in RVF's capacity to ignore a state based on β. In this thesis, we argue that this can provide more robustness to noise and partial observability. The ability of RVF to emphasize a state is similar to the interest function in emphatic TD (Mahmood et al., 2015), however, learning a state-dependent interest function and λ remains an open problem. In contrast, RVF has a natural way of learning β by comparing the past and the future. The capacity to ignore

states shares some motivations to semi-Markov decision process (Puterman, 1990). Learning β and ignoring states can be interpreted as learning temporal abstraction over the trajectory in policy evaluation. RVFs shares motivation with Natural Value Approximators (Z. Xu et al., 2017) but differs significantly in its design choices. The most important one is that they cut the gradients on $V^{\beta}_{\theta,\omega}$ after one-time step regardless of β. This effectively prevents any credit assignment. As described in the previous chapter on Temporal Regularization, the loss considered is also different. Finally, RVF can also be considered as a partially observable method (Kaelbling, Littman, and Cassandra, 1998). However, it differs significantly from the literature as it does not attempt to infer the underlying hidden state explicitly, but rather only decides if the past estimates align with the target. We argue that inferring an underlying state may be significantly harder than learning to ignore or emphasize a state based on its value. This is illustrated in the next section.

5.2 Experiments

In this section, we perform experiments on various tasks to demonstrate the effectiveness of RVF. First, we explore RVF robustness to partial observability on a synthetic domain. We then showcase RVF's robustness to noise on several complex continuous control tasks from the Mujoco suite (Todorov, Erez, and Tassa, 2012).

5.2.1 Partially observable multi-chain domain

We consider the simple chain MDP described in Figure 5.1. This MDP has three chains connected to form a Y. Each of the three chains (left of S_1, right of S_2, right of S_3) is made up of a sequence of states. The agent starts at S_0 and navigates through the chain. At the intersection S_1, there is a 0.5 probability of going up or down. The chain on top receives a reward of $+1$ while the one at the bottom receives a reward of -1. Every other transition has a reward of 0 unless specified otherwise.

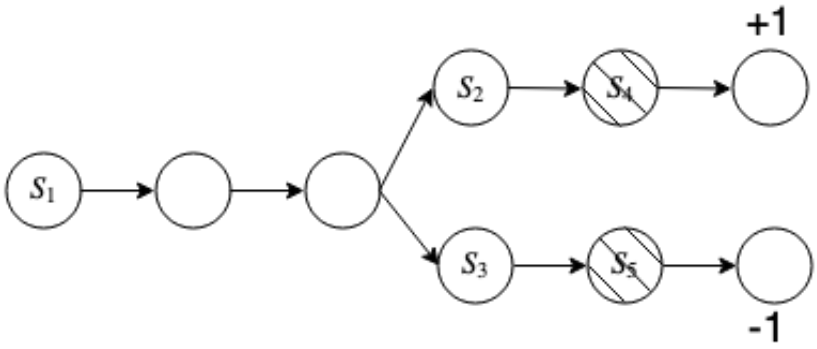


Figure 5.1: Simple chain MDP: The agent starts at state S_0 and navigates along the chain. States S_4 and S_5 are aliased.

We explore the capacity of recurrent learning to solve a partially observable task in the Y chain. In particular, we consider the case where some states are aliased (share a common representation). The representation of the states S_4 and S_5 in Figure 5.1 are aliased. Practically this means that the observation o used by the agent will be the same for both states $o(S_3) = o(S_4)$. The goal of this environment is to correctly estimate the value of the aliased state $V^\pi(S_4) = 0.9, V^\pi(S_5) = -0.9$ (due to the discount factor(0.9) and the length of each chain being 3) using the observations $o(S_3), o(S_4)$. When TD methods such as TD(0) or TD(λ) are used, the values of the aliased states S_4 and S_5 are close to 0 as the reward at the end of the chain is $+1$ and -1. However, when learning β (emphasis function β is modeled using a sigmoid function), Recurrent Value Functions achieve almost no error in their estimate of the aliased states as illustrated in Figure 5.2. For RTD we consider the error of $V^\beta_{\theta,\omega}$ with respect to the target and not the underlying V_θ.

It can be explained by observing that $\beta \to 0$ on the aliased state since the previous values along the trajectory are better estimates of the future than those of the aliased state. As $\beta \to 0$, $V^\beta_{\theta,\omega}(S_4)$ and $V^\beta_{\theta,\omega}(S_5)$ tend to rely on their more accurate previous estimates, $V^\beta_{\theta,\omega}(S_2)$ and $V^\beta_{\theta,\omega}(S_3)$. We see that learning to ignore certain states can, at times, be sufficient to solve an aliased task. We also compare with a recurrent version (O-RTD) where optimal values of β are used. In this setting,

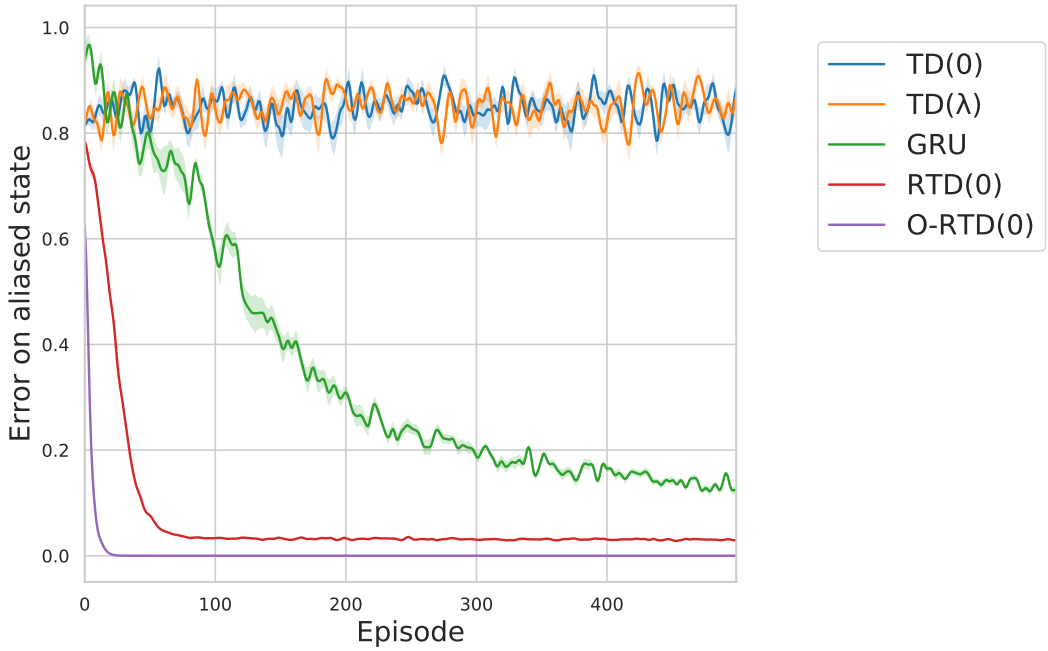


Figure 5.2: Results on the aliased Y-chain of various methods, such as TD(0), TD(λ), GRU, RTD(0), and Optimal RTD(0) (O-RTD(0)) averaged over 20 random seeds.

$\beta(S_1) = \beta(S_2) = \beta(s_3) = 1$ and other states have $\beta = 0$. Another interesting observation is with respect to Recurrent Neural Networks. RNNs are known to solve tasks which have partial observability by inferring the underlying state. LSTM and GRU have many parameters that are used to infer the hidden state. Correctly learning to keep the hidden state intact can be sample-inefficient. In comparison, β can estimate whether or not to put emphasis (confidence) on a state value using a single parameter. It is illustrated in Figure 5.2, where RNNs take ten times more episodes to learn the optimal value when compared to RVF. It illustrates a case where learning to ignore a state is easier than inferring its hidden representation.

The results displayed in Figure 5.2 are averaged over 20 random seeds. We noticed that the emphasis function is easier to learn if the horizon of the target is longer since a longer horizon provides a better prediction of the future. To account for this, we use λ-return as a target. For every method, the learning rate and λ is tuned for optimal performance in the range $[0, 1]$.

For RTD a learning rate of 0.5 for the value function and 1 for the beta function was found to be optimal with a lambda of 0.9.

For the GRU model, we explored different amount of cell ($\{1, 5, 10, 15, 20, 25\}$) to vary the capacity of the model. The optimal number of hidden cells we found is 10, learning rate 0.5 and lambda 0.9.

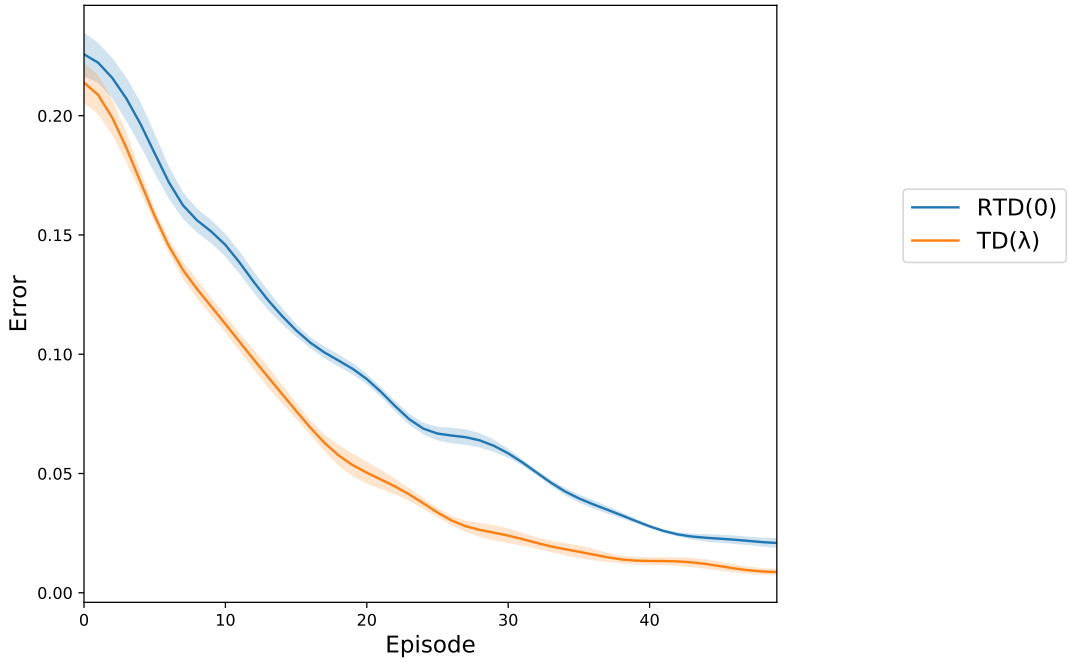


Figure 5.3: Results on the fully observable Y-chain of RTD(0) and TD(λ=0.9) averaged over 20 random seeds.

Non-Markovian behavior: One important thing to notice is the potential non-Markovian behavior of RTD. This is the reason we compare RTD with another non-Markovian method, namely, recurrent neural networks. As we mentioned earlier in the thesis, V^{β} is actually a function of the entire trajectory. In this scenario, it allows solving a problem that purely Markovian methods would not be able to solve. However, this can also have drawbacks. The same states can have different values depending on the rest of the trajectory. In fully-observable setting's RTD can slow learning compared to TD methods in a similar way that we showed temporal regularization could. However, one important aspect is that V^{β} attempts to be as close as possible

to the target (TD or monte-carlo). This means that in practice, if the trajectory is not informative, $\beta \to 1$ and $V^{\beta}(S_t) = V(S_t)$. This means that RTD will converge to the behavior of a Markovian methods if needed. As opposed to temporal regularization, RTD will not be biased in the limit as its regularization coefficient is learned. A second potential negative effect is that in fully observable setting, learning β could slow down learning compared to Markovian methods such as TD(λ). This is illustrated in Figure 5.3 where we modify the MDP to be fully observable.

5.2.2 Continuous control

Next, we test RVF on several environments of the Mujoco suite (Todorov, Erez, and Tassa, 2012). We also evaluate the robustness of different algorithms by adding ϵ sensor noise (drawn from a normal distribution $\epsilon \sim N(0, 1)$) to the observations as presented in (Zhang, Ballas, and Pineau, 2018). We modify the critic of A2C (Wu et al., 2017) (R-A2C) and Proximal Policy Optimization (R-PPO) (Schulman, Wolski, et al., 2017) to estimate the recurrent value function parametrized by θ. We parametrize β using a separate network with the same architecture as the value function (parametrized by ω). We minimize the loss mentioned in Eq. 5.2 but replace the target with generalized advantage function (V^{λ}) (Schulman, Moritz, et al., 2015) for PPO and TD(n) for A2C. Using an automatic differentiation library (Pytorch (Paszke et al., 2017)), we differentiate the loss through the modified estimate to learn θ and ω. The default optimal hyperparameters of PPO and A2C are used. Due to the batch nature of PPO, obtaining the trajectory information to create the computational graph can be costly. In this regard, we cut the backpropagation after N timesteps in a similar manner to truncated backpropagation through time. The number of backpropagation steps is obtained using a hyperparameter search. The best hyperparameters are selected on ten random seeds. The following values were considered for the learning rate

$\{3E-05, 6E-05, 9E-05, 3E-04, 6E-04\}$ and $N = \{2, 5, 10\}$. The optimal value for learning rate is the same one obtained in the original PPO paper $3E-4$ and $N = 5$. We also compare with a larger network for PPO to adjust for the additional parameter of β the performance of vanilla PPO were found to be similar. In terms of computational cost, RVF introduces a computational overhead slowing down training by a factor of 2 on a CPU(Intel Skylake cores 2.4GHz, AVX512) compared to PPO. The results are reported on 20 new random seeds, and a confidence interval of 68% is displayed.

We use a truncated backprop of $N = 5$ in our experiments as we found no empirical improvements for $N = 10$. For a fairer comparison in the noisy case, we also compare the performance of two versions of PPO with an LSTM. The first version processes one trajectory every update. The second uses a buffer in a similar manner to PPO, but the gradient is cut after five steps as the computation overhead from building the graph every time is too large. The performance reported is averaged over 20 different random seeds with a confidence interval of 68% displayed [1]

5.2.2.1 Performance

As demonstrated in Figure 5.4, we observe a marginal increase in performance on several tasks such as Swimmer, Walker, Hopper, Half Cheetah, and Double Inverted Pendulum in the fully observable setting. However, severe drops in performance were observed in the vanilla PPO when we induced partial observability by introducing a Gaussian noise to the observations. On the other hand, R-PPO (PPO with RVF) was found to be robust to the noise, achieving significantly higher performance in all the tasks considered. In both cases, R-PPO outperforms the partially observable models (LSTM). The mean and standard deviation of the emphasis function for both noiseless and noisy versions can be found in Appendix(5.5, 5.6). At the same time, A2C performance on both vanilla and recurrent versions (referred to as R-A2C) were

[1]The base code used to develop this algorithm can be found here (Kostrikov, 2018)

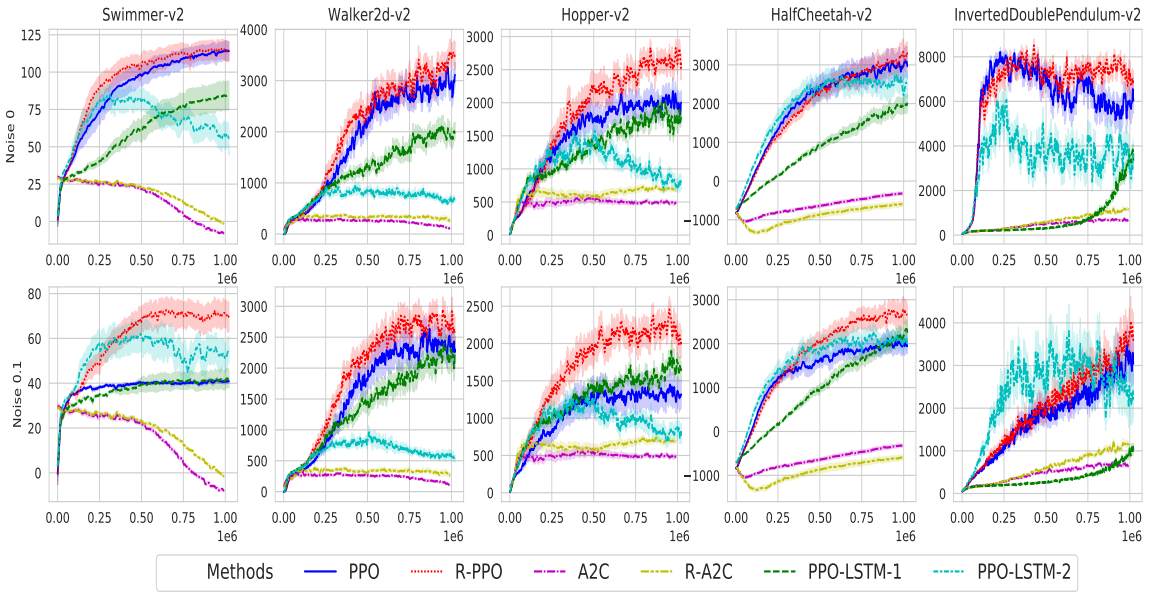


Figure 5.4: Performance on Mujoco tasks. Results on the first row are generated without noise and on the second row by inducing a Gaussian noise ($\epsilon \sim \mathbb{N}(0, 0.1)$) in the sensor inputs.

found to be poor. We increased the training steps on both versions and noticed the same observations as mentioned above, once A2C started to learn the task. The mean and standard deviation of the emphasis function during training, can be found in Fig (5.5, 5.6).

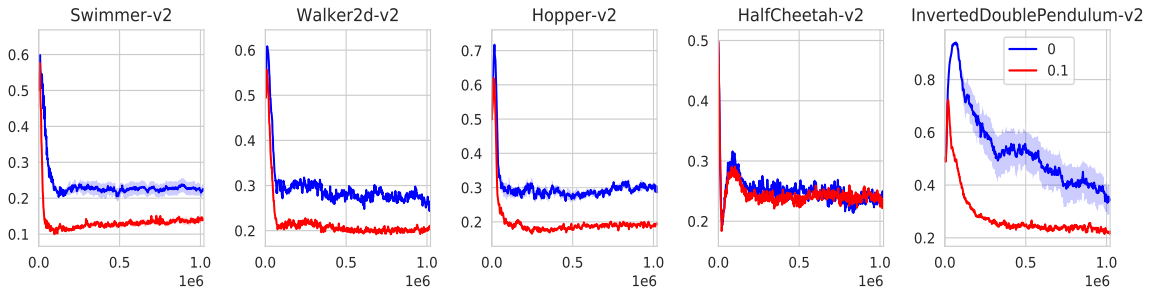


Figure 5.5: Mean beta values using recurrent PPO on Mujoco domains

5.2.3 Ablation study

In this section, we perform an ablation study to evaluate the impact of learning a smoothing coefficient. As a reminder in Value-Based Temporal Regularization,

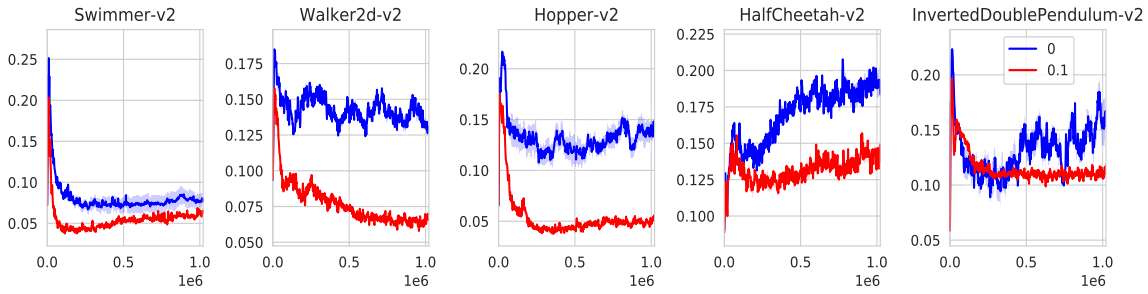


Figure 5.6: Standard deviation of beta using recurrent PPO on Mujoco domains

we made the hypothesis that a fixed smoothing coefficient might hinder learning on environment lacking stochasticity like Mujoco. As illustrated in Figure 5.7, the performance significantly drops when β is not learned. Without learning β a vanilla version of PPO outperform the recurrent one in almost all environment.

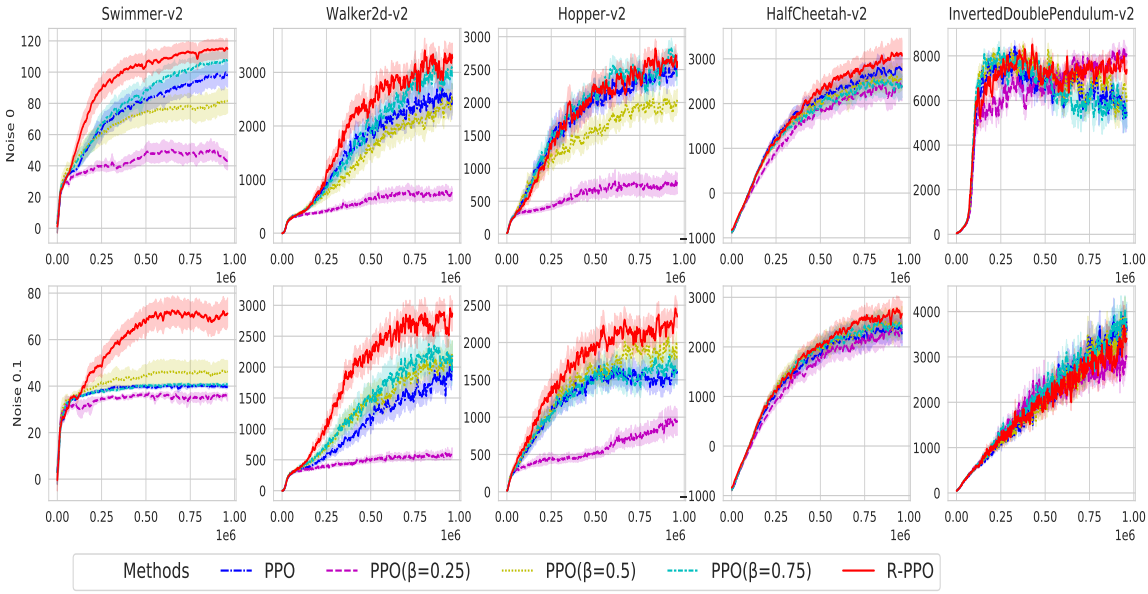


Figure 5.7: Ablation study with fixed β and state dependent learned β(R-PPO).

5.2.4 Qualitative interpretation of the emphasis function β

In this section, we can qualitatively analyze the learned emphasis function (β) through the trajectory.

5.2.4.1 Hopper

At the end of the training, We observe cyclical behavior shown in Figure 5.8, where different colors describe various stages of the cycle.

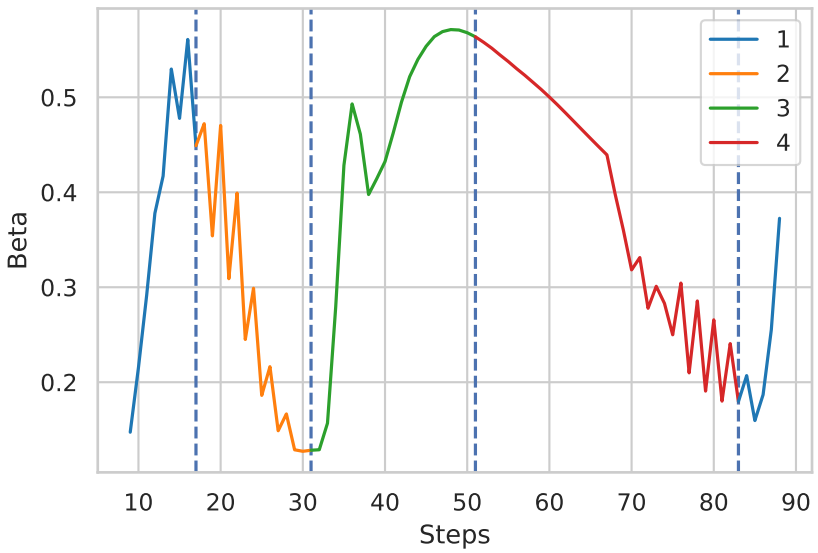
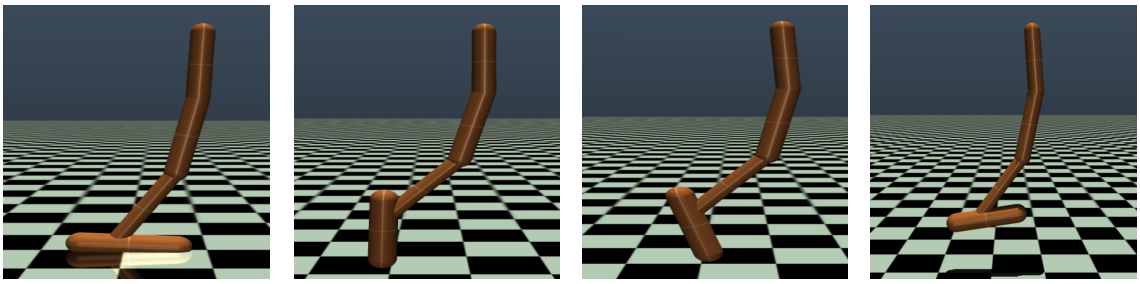


Figure 5.8: Behaviour of β through the trajectory.

The emphasis function learned to identify *important states* and to ignore the others. One intuitive way to look at the emphasis function(β) is: *If I were to give a different value to a state, would that alter my policy significantly?* We observe an increase in the value of the emphasis function (β) when the agent must make an important decision, such as jumping or landing. We see a decrease in the value of the emphasis function (β) when the agent must perform a trivial action. This pattern is illustrated in Figure 5.9 and 5.8. This behavior is cyclic and repetitive, a video of which can be found in the following link[2].

[2] https://youtu.be/0bzEcrxNwRw



Phase 1: high β Phase 2: low β Phase 3: high β Phase 4: low β

Figure 5.9: Cyclical behaviour of β on Hopper.

(a) The emphasis function learns to emphasize critical states in the environment, as demonstrated in Figure 5.9. The emphasis function is high when the agent is making important decisions, such as landing or taking off (Phase 1 and Phase 3). The emphasis function is low when the agent is making decisions while it is in the air (Phase 2 and Phase 4). (b) Behavior of the emphasis function along the trajectory for various phases described in (a) for one period. The emphasis function keeps repeating the behavior.

5.2.4.2 Mountain car

Two scenarios may happen when the agent is climbing up the hill on the right side. Either the agent has enough velocity to finish the game and obtain a high reward, or it does not have enough velocity and goes back down the hill. During the early stages of training, the function approximator is confused about the scenarios mentioned earlier, resulting in a drop in value function around step 100, as shown in figure 5.10.

The value increases again once the agent climbs the hill with more velocity. In PPO, we can obtain a more accurate target by setting τ to a high value, thereby eliminating a drop in value. This enables the β network to learn to trust its past estimate rather than the noisy point estimate, hence a significant drop in the β value. As a result, $V^{\beta}_{\theta,\omega}$ becomes a much better estimate of the target than V in this scenario. After training PPO for a while, this drop disappears, and the β mean goes to 1. This experiment shows the potential of β to smooth out noisy estimates in the trajectory.

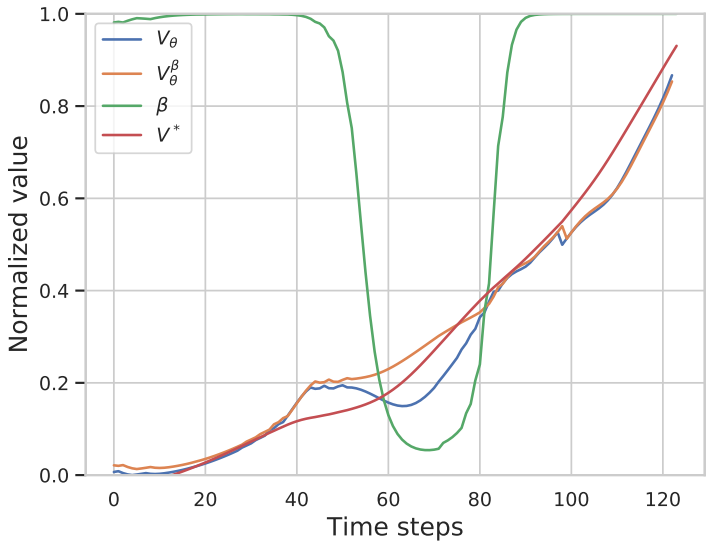


Figure 5.10: Behavior of β and the value function on Mountain-Car

One caveat to consider is the feedback loop induced by ignoring a state in control. When the policy changes a state that can be ignored at the beginning may be essential later on. One way to address this is to avoid saturating β such that learning remains possible later on.

In Reinforcement Learning, having access to a function quantifying the *interest* (Mahmood et al., 2015) of a state can be helpful. For example, one could decide to explore from those states, prioritize experience replay based on those states, and use β to set the λ to bootstrap from interesting states. Indeed, bootstrapping on states with a similar value (low β) than the one estimated will only result in added variance. The most informative updates come from bootstrapping on states with different values (high β). We also believe β to be related to the concepts of bottleneck states (Tishby and Polani, 2011) and reversibility.

6

Conclusion

Variance in value-based method remains a central issue in Reinforcement Learning. In this thesis, we propose to use the value estimates computed along the trajectory to reduce the variance of the value function. In particular, we propose to use time-series models to smooth value function's estimates. First, we propose to smooth the target (Temporal Regularization) used in RL then we directly alter the value estimate (Recurrent Value Functions).

When modifying the target used in Reinforcement Learning, this leads to a new perspective on regularization in RL, that is from a temporal perspective. In contrast with typical spatial regularization, where one assumes that rewards are close for nearby states in the state space, temporal regularization instead assumes that rewards are close for states *visited closely in time*. Regularizing based on the past seems like a fundamental inductive bias we as humans exploit. However, as demonstrated, smoothing estimates when significant changes occur can introduce too much bias, hindering learning. To tackle this problem, we introduced a new way to estimate value functions, namely Recurrent Value Functions (RVFs). The critical component of RVFs is a learned state-dependent β coefficient controlling the smoothing coefficient. There exists many interesting directions for future work. We describe some of them below:

State-dependent β: As demonstrated in this thesis, β seem to encode important information about the structure of the MDP and the learning algorithm. In particular,

in robotics, identifying key states has been an issue for a long time. We argue that important states are not just based on the dynamics of the environment but also a function of the learning algorithm. This is illustrated in RVF's where a state is only deemed important if its value estimate varies from the past. Having access to a function defining the *importance* of a state in RL can provide useful insights into the inner-working of RL algorithms. A new direction would be to exploit this emphasis function to fasten learning and improve sample efficiency. One could prioritize experience replay based on β or use it to define an initiation set for options. Finally, the motivation behind RVF is the success of the gating mechanism in Recurrent Neural Network for credit assignment. In this thesis, we did not study the problem of credit assignment. However, it is an exciting avenue for further research. For example, enforcing β to be sparse using temporal regularization could effectively filter out the noisy gradient and better temporal credit assignment in Reinforcement Learning.

Control: In this thesis, we only considered smoothing value estimates. However, in a control setting, one may exploit previous actions as well. Several work started exploring this avenue (Harb et al., 2017; Mladenov et al., 2019; Korenkevych et al., 2019), in particular in the context of structured exploration. Those work may be generalized and understood better under the framework of temporal regularization. Similar idea's of reversibility and reversal Markov chains could be used to develop theoretical results in the control setting. Practically the framework of temporal regularization can shed light on the bias-variance induced by smoothing action temporally.

Complex time-series model: Overall, both temporal regularization and RVFs can be seen as using time-series models on the value function through the trajectory both for the target and the estimate. Many choices made along the way leads to different properties, but this remains the central idea. In this thesis, we focused on exponential smoothing for its simplicity. However, a natural extension would be to consider more

powerful time-series models such as Kalman filter.

Supervised Learning: As detailed earlier, it is possible to view RVF's as a simple model for partially observable environments. We demonstrate that, in some environments, RVFs can outperform commonly used models such as RNNs, due to its simplicity (low capacity). It could be interesting to consider this idea in the supervised learning setting. More concretely one could estimate the output of a sequential regression problem by exponentially smoothing the output along the time axis.

Backward and forward view: The two work proposed in this thesis can be interpreted as forward smoothing (Temporal Regularization) and backward smoothing (Recurrent Value Functions). As illustrated by the equivalence between eligibility traces and the lambda return (R. S. Sutton, 1985), there exists a connection between both in the offline setting. It could be interesting to study if such a connection exists in our case. More generally, this suggests that there exists a deep connection between the reward shaping literature and gradient-based learning.

Overall, we believe that this line of work presents many exciting directions that may prove to be essential for efficient Reinforcement Learning. The results presented in this thesis demonstrate how the trajectories information can be leveraged for better performance and insights on the inner working of RL algorithms. We hope that it motivates further work in this area.

Bibliography

Abbeel, Pieter, Adam Coates, and Andrew Y Ng (2010). "Autonomous helicopter aerobatics through apprenticeship learning". In: *The International Journal of Robotics Research* 29.13, pp. 1608–1639.

Asif, M Salman, Lei Hamilton, Marijn Brummer, and Justin Romberg (2013). "Motion-adaptive spatio-temporal regularization for accelerated dynamic MRI". In: *Magnetic Resonance in Medicine* 70.3, pp. 800–812.

Åström, Karl Johan (1965). "Optimal control of Markov processes with incomplete state information". In: *Journal of Mathematical Analysis and Applications* 10.1, pp. 174–205.

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

Banach, Stefan (1922). "Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales". In:

Bartlett, Peter L and Ambuj Tewari (2009). "REGAL: A regularization based algorithm for reinforcement learning in weakly communicating MDPs". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, pp. 35–42.

Baxter, Jonathan and Peter L Bartlett (2001). "Infinite-horizon policy-gradient estimation". In: *Journal of Artificial Intelligence Research* 15, pp. 319–350.

Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). "The arcade learning environment: An evaluation platform for general agents". In: *Journal of Artificial Intelligence Research* 47, pp. 253–279.

Bellman, Richard (1957). "A Markovian decision process". In: *Journal of Mathematics and Mechanics*, pp. 679–684.

Bellman, Richard et al. (1954). "The theory of dynamic programming". In: *Bulletin of the American Mathematical Society* 60.6, pp. 503–515.

Bengio, Yoshua, Patrice Simard, Paolo Frasconi, et al. (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Borkar, Vivek S (2009). *Stochastic approximation: a dynamical systems viewpoint*. Vol. 48. Springer.

Borkar, Vivek S and Sean P Meyn (2000). "The ODE method for convergence of stochastic approximation and reinforcement learning". In: *SIAM Journal on Control and Optimization* 38.2, pp. 447–469.

Box, George, Gwilym M. Jenkins, and Gregory C. Reinsel (1994). *Time Series Analysis: Forecasting and Control (3rd ed.)* Prentice-Hall.

Boyd, Stephen and Lieven Vandenberghe (2004). *Convex optimization*. Cambridge university press.

Brémaud, Pierre (2013). *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. Vol. 31. Springer Science & Business Media.

Brockwell, Peter J, Richard A Davis, and Matthew V Calder (2002). *Introduction to time series and forecasting*. Vol. 2. Springer.

Brockwell, Peter J, Richard A Davis, and Stephen E Fienberg (1991). *Time Series: Theory and Methods: Theory and Methods*. Springer Science & Business Media.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio (2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555*.

Chung, Kai-Min, Henry Lam, Zhenming Liu, and Michael Mitzenmacher (2012). "Chernoff-Hoeffding bounds for Markov chains: Generalized and simplified". In: *arXiv preprint arXiv:1201.0559*.

Dayan, Peter (1992). "The convergence of TD (λ) for general λ". In: *Machine learning* 8.3-4, pp. 341–362.

Dhariwal, Prafulla, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu (2017). *OpenAI Baselines*. https://github.com/openai/baselines.

Diaconis, Persi and Daniel Stroock (1991). "Geometric bounds for eigenvalues of Markov chains". In: *The Annals of Applied Probability*, pp. 36–61.

Engl, Heinz Werner, Martin Hanke, and Andreas Neubauer (1996). *Regularization of inverse problems*. Vol. 375. Springer Science & Business Media.

Farahmand, Amir-massoud (2011). "Regularization in reinforcement learning". PhD thesis. University of Alberta.

Farahmand, Amir-massoud, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor (2009). "Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems". In: *American Control Conference*. IEEE, pp. 725–730.

Fill, James Allen et al. (1991). "Eigenvalue Bounds on Convergence to Stationarity for Nonreversible Markov Chains, with an Application to the Exclusion Process". In: *The Annals of Applied Probability* 1.1, pp. 62–87.

Fox, Roy, Ari Pakman, and Naftali Tishby (2015). "Taming the noise in reinforcement learning via soft updates". In: *arXiv preprint arXiv:1512.08562*.

François-Lavet, Vincent, David Taralla, Damien Ernst, and Raphaël Fonteneau (2016). "Deep reinforcement learning solutions for energy microgrids management". In: *European Workshop on Reinforcement Learning (EWRL 2016)*.

Gardner Jr, Everette S (1985). "Exponential smoothing: The state of the art". In: *Journal of forecasting* 4.1, pp. 1–28.

Gardner, Everette S (2006). "Exponential smoothing: The state of the art—Part II". In: *International journal of forecasting* 22.4, pp. 637–666.

Gläscher, Jan, Nathaniel Daw, Peter Dayan, and John P O'Doherty (2010). "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning". In: *Neuron* 66.4, pp. 585–595.

Golub, Gene H and Richard S Varga (1961). "Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order Richardson iterative methods". In: *Numerische Mathematik* 3.1, pp. 147–156.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Greensmith, Evan, Peter L Bartlett, and Jonathan Baxter (2004). "Variance reduction techniques for gradient estimates in reinforcement learning". In: *Journal of Machine Learning Research* 5.Nov, pp. 1471–1530.

Grinberg, Yuri, Doina Precup, and Michel Gendreau (2014). "Optimizing energy production using policy search and predictive state representations". In: *Advances in Neural Information Processing Systems*, pp. 2969–2977.

Groetsch, CW (1984). "The theory of Tikhonov regularization for Fredholm equations". In: *104p, Boston Pitman Publication*.

Harb, Jean, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup (2017). "When waiting is not an option: Learning options with a deliberation cost". In: *arXiv preprint arXiv:1709.04571*.

Harrigan, Cosmo (2016). "Deep reinforcement learning with regularized convolutional neural fitted q iteration". In:

Hasselt, Hado P van, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver (2016). "Learning values across many orders of magnitude". In: *Advances in Neural Information Processing Systems*, pp. 4287–4295.

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2017). "Deep reinforcement learning that matters". In: *arXiv preprint arXiv:1709.06560*.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hoerl, Arthur E and Robert W Kennard (1970). "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1, pp. 55–67.

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra (1998). "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2, pp. 99–134.

Kakade, Sham Machandranath et al. (2003). "On the sample complexity of reinforcement learning". PhD thesis.

Kalman, Rudolph Emil (1960). "A new approach to linear filtering and prediction problems". In: *Journal of basic Engineering* 82.1, pp. 35–45.

Kearns, Michael J and Satinder P Singh (2000). "Bias-Variance Error Bounds for Temporal Difference Updates." In:

Kemeny, John G and James Laurie Snell (1976). "Finite markov chains, undergraduate texts in mathematics". In:

Klopf, A Harry (1982). *The hedonistic neuron: a theory of memory, learning, and intelligence*. Toxicology-Sci.

Kober, Jens, J Andrew Bagnell, and Jan Peters (2013). "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11, pp. 1238–1274.

Konda, Vijay R and John N Tsitsiklis (2000). "Actor-critic algorithms". In: *Advances in neural information processing systems*, pp. 1008–1014.

Korenkevych, Dmytro, A. Rupam Mahmood, Gautham Vasan, and James Bergstra (2019). "Autoregressive Policies for Continuous Control Deep Reinforcement Learning". In: *CoRR* abs/1903.11524.

Kostrikov, Ilya (2018). *PyTorch Implementations of Reinforcement Learning Algorithms*. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105.

Laroche, Van Seijen (2018). "IN REINFORCEMENT LEARNING, ALL OBJECTIVE FUNCTIONS ARE NOT EQUAL". In: *ICLR Workshop*.

Lee, Ernest Bruce and Lawrence Markus (1967). *Foundations of optimal control theory*. Tech. rep. Minnesota Univ Minneapolis Center For Control Sciences.

Leike, Jan (2016). "Nonparametric General Reinforcement Learning". In: *arXiv preprint arXiv:1611.08944*.

Levin, David A and Yuval Peres (2008). *Markov chains and mixing times*. Vol. 107. American Mathematical Soc.

Liu, Bo, Sridhar Mahadevan, and Ji Liu (2012). "Regularized off-policy TD-learning". In: *Advances in Neural Information Processing Systems*, pp. 836–844.

Lütkepohl, Helmut (2005). *New introduction to multiple time series analysis*. Springer Science & Business Media.

Mahadevan, Sridhar (1996). "Average reward reinforcement learning: Foundations, algorithms, and empirical results". In: *Machine learning* 22.1-3, pp. 159–195.

— (1994). "To discount or not to discount in reinforcement learning: A case study comparing R learning and Q learning". In: *Machine Learning Proceedings 1994*. Elsevier, pp. 164–172.

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

Makridakis, Spyros, Steven C Wheelwright, and Rob J Hyndman (2008). *Forecasting methods and applications*. John wiley & sons.

Markov, Andrei Andreevich (1954). "The theory of algorithms". In: *Trudy Matematicheskogo Instituta Imeni VA Steklova* 42, pp. 3–375.

Mladenov, Martin, Ofer Meshi, Jayden Ooi, Dale Schuurmans, and Craig Boutilier (May 2019). "Advantage Amplification in Slowly Evolving Latent-State Environments". In: *arXiv e-prints*, arXiv:1905.13559, arXiv:1905.13559.

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*, pp. 1928–1937.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602*.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814.

Neu, Gergely, Anders Jonsson, and Vicenç Gómez (2017). "A unified view of entropy-regularized markov decision processes". In: *arXiv preprint arXiv:1705.07798*.

Norris, James R (1998). *Markov chains*. 2. Cambridge university press.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). "Automatic differentiation in PyTorch". In: *NIPS-W*.

Pazis, Jason and Ronald Parr (2011). "Non-Parametric Approximate Linear Programming for MDPs." In: *AAAI*.

Pendrith, Mark D (1994). *On reinforcement learning of control actions in noisy and non-Markovian domains*. Citeseer.

Petrik, Marek, Gavin Taylor, Ron Parr, and Shlomo Zilberstein (2010). "Feature selection using regularization in approximate linear programs for Markov decision processes". In: *arXiv preprint arXiv:1005.1860*.

Poggio, Tomaso, Vincent Torre, and Christof Koch (1987). "Computational vision and regularization theory". In: *Readings in Computer Vision*. Elsevier, pp. 638–643.

Precup, Doina (2000). "Eligibility traces for off-policy policy evaluation". In: *Computer Science Department Faculty Publication Series*, p. 80.

Puterman, Martin L (1990). "Markov decision processes". In: *Handbooks in operations research and management science* 2, pp. 331–434.

— (1994). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley Sons.

Robbins, Herbert and Sutton Monro (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Romoff, Joshua, Peter Henderson, Alexandre Piché, Vincent Francois-Lavet, and Joelle Pineau (2018). "Reward estimation for variance reduction in deep reinforcement learning". In: *arXiv preprint arXiv:1805.03359*.

Rumelhart, David E, Geoffrey E Hinton, Ronald J Williams, et al. (1987). "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3, p. 1.

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). "Trust region policy optimization". In: *International Conference on Machine Learning*, pp. 1889–1897.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2015). "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438*.

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347*.

Seijen, Harm and Rich Sutton (2014). "True online TD (lambda)". In: *International Conference on Machine Learning*, pp. 692–700.

Shah, Devavrat and Qiaomin Xie (2018). "Q-learning with Nearest Neighbors". In: *arXiv preprint arXiv:1802.03900*.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587, p. 484.

Singh, Satinder P, Tommi Jaakkola, and Michael I Jordan (1994). "Learning without state-estimation in partially observable Markovian decision processes". In: *Machine Learning Proceedings 1994*. Elsevier, pp. 284–292.

Singh, Satinder P and Richard S Sutton (1996). "Reinforcement learning with replacing eligibility traces". In: *Machine learning* 22.1-3, pp. 123–158.

Sondik, Edward J (1978). "The optimal control of partially observable Markov processes over the infinite horizon: Discounted costs". In: *Operations research* 26.2, pp. 282–304.

Sutton, Richard S (1990). "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming". In: *Machine Learning Proceedings 1990*. Elsevier, pp. 216–224.

— (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1, pp. 9–44.

— (1985). "Temporal Credit Assignment in Reinforcement Learning." In:

Sutton, Richard S and Andrew G Barto (n.d.). "Reinforcement Learning: An Introduction". In: ().

— (1998). *Reinforcement learning: An introduction*. 1st. MIT press Cambridge.

Sutton, Richard S and Andrew G Barto (2017). *Reinforcement learning: An introduction*. (in progress) 2nd. MIT press Cambridge.

Sutton, Richard S, David A McAllester, Satinder P Singh, and Yishay Mansour (2000). "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems*, pp. 1057–1063.

Sutton, Richard S and Satinder P Singh (n.d.). "On step-size and bias in temporal-difference learning". In: Citeseer.

Sutton, Richard Stuart (1984). "Temporal credit assignment in reinforcement learning". In:

Thodoroff, Pierre, Nishanth Anand, Lucas Caccia, Doina Precup, and Joelle Pineau (2019). "Recurrent Value Functions". In: *CoRR* abs/1905.09562.

Thodoroff, Pierre, Audrey Durand, Joelle Pineau, and Doina Precup (2018). "Temporal Regularization for Markov Decision Process". In: *Advances in Neural Information Processing Systems*, pp. 1779–1789.

Tishby, Naftali and Daniel Polani (2011). "Information theory of decisions and actions". In: pp. 601–636.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, pp. 5026–5033.

Tsitsiklis, John N (1994). "Asynchronous stochastic approximation and Q-learning". In: *Machine learning* 16.3, pp. 185–202.

Tsitsiklis, John N and Benjamin Van Roy (2002). "On average versus discounted reward temporal-difference learning". In: *Machine Learning* 49.2-3, pp. 179–191.

Varga, Richard S (2009). *Matrix iterative analysis*. Vol. 27. Springer Science & Business Media.

Vinyals, Oriol, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou,

Julian Schrittwieser, et al. (2017). "Starcraft ii: A new challenge for reinforcement learning". In: *arXiv preprint arXiv:1708.04782*.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine learning* 8.3-4, pp. 279–292.

Welch, Greg and Gary Bishop (1995). "An introduction to the Kalman filter". In:

Werbos, Paul J (1982). "Applications of advances in nonlinear sensitivity analysis". In: *System modeling and optimization*. Springer, pp. 762–770.

Williams, Ronald J and David Zipser (1995). "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: *Backpropagation: Theory, architectures, and applications* 1, pp. 433–486.

Wu, Yuhuai, Elman Mansimov, Roger B Grosse, Shun Liao, and Jimmy Ba (2017). "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation". In: *Advances in neural information processing systems*, pp. 5279–5288.

Xu, Bing, Naiyan Wang, Tianqi Chen, and Mu Li (2015). "Empirical evaluation of rectified activations in convolutional network". In: *arXiv preprint arXiv:1505.00853*.

Xu, Kevin S, Mark Kliger, and Alfred O Hero (2013). "A regularized graph layout framework for dynamic network visualization". In: *Data Mining and Knowledge Discovery* 27.1, pp. 84–116.

Xu, Zhongwen, Joseph Modayil, Hado P van Hasselt, Andre Barreto, David Silver, and Tom Schaul (2017). "Natural Value Approximators: Learning when to Trust Past Estimates". In: *Advances in Neural Information Processing Systems*, pp. 2117–2125.

Yu, Hsiang-Fu, Nikhil Rao, and Inderjit S Dhillon (2016). "Temporal regularized matrix factorization for high-dimensional time series prediction". In: *Advances in neural information processing systems*, pp. 847–855.

Zhang, Amy, Nicolas Ballas, and Joelle Pineau (2018). "A dissection of overfitting and generalization in continuous reinforcement learning". In: *arXiv preprint arXiv:1806.07937*.

Zhou, Kemin, John Comstock Doyle, Keith Glover, et al. (1996). *Robust and optimal control*. Vol. 40. Prentice hall New Jersey.