Abstract of “Smoothness in Reinforcement Learning with Large State and Action Spaces” by Kavosh Asadi, Ph.D., Brown University, October 2nd, 2020.

Reinforcement learning (RL) is the study of the interaction between an environment and an artificial agent that learns to maximize reward through trial-and-error. Owing to its generality, RL is used to formulate numerous applications, including those with high-dimensional state or action spaces. RL agents that are equipped with function approximation can tackle high-dimensional problems, but they are also notoriously difficult to study. Thus, a fundamental question in RL is how to design algorithms that are compatible with function approximation, but are also guaranteed to converge, can explore effectively, and can perform long-horizon planning. In this thesis I study RL through the lens of *smoothness* formally defined using Lipschitz continuity. I present theoretical results showing an essential role for smoothness in stability and convergence of RL, effective model learning and planning, and in value-function approximation in continuous control. Through many examples and experiments, I also demonstrate how to adjust the smoothness of key RL ingredients to improve performance in presence of function approximation.

Smoothness in Reinforcement Learning with Large State and Action Spaces

by
Kavosh Asadi
B. E., University of Tehran, 2013
M. Sc., University of Alberta, 2015

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
October 2nd, 2020

This dissertation by Kavosh Asadi is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date ______________ ______________________________
Michael L. Littman, Director

Recommended to the Graduate Council

Date ______________ ______________________________
George D. Konidaris, Reader

Date ______________ ______________________________
Ronald E. Parr, Reader
(Duke University)

Approved by the Graduate Council

Date ______________ ______________________________
Andrew G. Campbell
Dean of the Graduate School

To my cousin, Pouneh. I hope I will see you again.

Vita

Kavosh was born in Chalous, a tiny, green, and beautiful town in Northern Iran. As a child, he played video games (mostly FIFA) for an excessive amount of time, which prompted his interest in artificial minds. In 2008, he moved to Iran's capital, Tehran, to complete his undergraduate degree at the University of Tehran. As a senior undergraduate he was introduced to reinforcement learning, a field that Kavosh thought is best formulating the interaction between an artificial mind and the world. He obtained his Master's degree from the University of Alberta in Canada under Rich Sutton. Since then, he has been pursuing a PhD degree under Michael Littman at the Department of Computer Science at Brown University in the United States.

Acknowledgments

I am so lucky to be able to lean on my parents' support. Thank you, Alireza and Marzieh, for paving my path to success. I hope someday I will be a good father for your grand children.

Thank you, my sweet and kind partner Farzaneh. You never stopped believing in me.

Writing this thesis with no intellectual support would have been impossible. Special credit goes to my mentor, Michael Littman, who cultivated an environment in which I felt rewarded for exploring my own interests. I am equally thankful of my second mentor, George Konidaris, who supported me in the darkest of my times. For George's support I will forever remain grateful. It has also been a great pleasure to work with Ron Parr, who is a deep thinker.

Some of the ideas presented here are shaped by my experience under Rich Sutton at Alberta. Rich believed in my potential, and in doing so, influenced my career in a profound way.

I feel privileged that I shared my office with Dave Abel, whose positive attitude towards life is a great example to imitate. My second office-mate was Matt Corsaro, who was as kind as an office-mate can be. I was also fortunate to get to know Chris Tanner and Cam Allen. Chatting with them always made me feel more educated, and I am looking forward to deepening my friendship with them.

I praise the CS Department at Brown for fostering a welcoming environment for international students. More specifically, I thank Lauren Clarke, Shriram Krishnamurthi, and Ugur Cetintemel.

I was lucky to mentor junior researchers at Brown, in particular Seungchan Kim, Neev Parikh, and Evan Cater. I am certain that they have a fantastic career ahead of them.

Lastly, I am thankful of my close friends Grant and Elizabeth Achilles, Sadjad Asghari, Pouyan Behnam, Neda Derakhshani, Babak Hemmatian, Sina Ghiassian, Sina Miran, Mostafa Vafadoust, Dipendra Misra, Vahab Vahdat, and Remo van der Heiden.

Contents

List of Tables

List of Figures

Abstract

Reinforcement learning (RL) is the study of the interaction between an environment and an artificial agent that learns to maximize reward through trial-and-error. Owing to its generality, RL is used to formulate numerous applications, including those with high-dimensional state or action spaces. RL agents that are equipped with function approximation can tackle high-dimensional problems, but they are also notoriously difficult to study. Thus, a fundamental question in RL is how to design algorithms that are compatible with function approximation, but are also guaranteed to converge, can explore effectively, and can perform long-horizon planning. In this thesis I study RL through the lens of *smoothness* formally defined using Lipschitz continuity. I present theoretical results showing an essential role for smoothness in stability and convergence of RL, effective model learning and planning, and in value-function approximation in continuous control. Through many examples and experiments, I also demonstrate how to adjust the smoothness of key RL ingredients to improve performance in presence of function approximation.

Thesis Statement

In high-dimensional reinforcement learning, Lipschitz continuity is key to ensuring convergence to a unique fixed point, effective value-function approximation in continuous control, and long-horizon planning.

Chapter 1

Introduction

Reinforcement learning (RL) is the scientific study of the interaction between an environment and an agent that learns to achieve a goal. The RL agent faces a sequential decision-making problem where the goal is often defined as maximizing the long-term sum of future rewards provided to the agent by its environment. Owing to this remarkably general definition, RL has enjoyed many applications including in settings with large (sometimes infinite) state or action spaces. Examples of these applications include learning to play games [66], robotics [58, 89], dialog systems [118, 11], healthcare [46], and optimization [60].

Tackling high-dimensional RL problems presents new challenges. Chief among them is the curse of dimensionality, which describes the notion that the difficulty of solving a problem increases rapidly with the number of dimensions. A promising approach to combat the curse of dimensionality is to endow RL agents with function approximators in order to effectively generalize experience from observed interactions to yet unseen states and actions. However, introducing function approximation comes at its own cost, namely that it becomes notoriously more difficult to study longstanding challenges in RL such as convergence guarantees, effective exploration, and long-horizon planning.

In this thesis I develop a theory that demonstrates a key role for smoothness in high-dimensional reinforcement learning. Informally, I define a smooth function as any function with the following property: For any two distinct points on its domain, the line that connects the surface of the function at the two points should have a finite slope. The Lipschitz constant of the function is defined as the maximum value of this slope for any pair of points on its domain. A function with a Lipschitz constant K is said to be K-Lipschitz. This notion of smoothness can be further generalized to functions with high-dimensional outputs, as well as functions that output probability distributions, as I show later.

My first result pertains to convergence of RL to a unique fixed-point. I study this property in the context of the exploration-exploitation problem. Rather than defining the value of a state based on its maximum action-value, it is common to use *softmax* operators, those that provide a soft approximation of maximum by putting some weight behind non-maximal actions. Convergence of RL in this case hinges on using operators that are K-Lipschitz with $K \leq 1$. A Lipschitz operator with this property is sometimes known as a non-expansion. I show that Boltzmann softmax operator, commonly used for exploration in large RL problems, is not a non-expansion and is prone to misbehavior. I then introduce an alternative softmax operator which has several nice properties including non-expansion. The new operator exhibits convergent behavior, and also yields stable and effective exploration in large problems such as the standard Atari games.

I then move to the setting with infinite actions, where finding an action that is optimal with respect to a learned state–action value function can be difficult. I introduce deep radial-basis value functions (RBVFs): state–action value functions learned using a deep neural network with a radial-basis function (RBF) output layer. I show that the optimal action with respect to a deep RBVF can be easily approximated up to any desired accuracy without impeding universal function approximation. I show that controlling the smoothness of deep RBVFs yields state-of-the-art performance in infinite-action RL problems.

Third, I focus on model-based reinforcement learning, where I show that in absence of model smoothness even a near-perfect model, defined as a model with low one-step error, can have arbitrarily large multi-step errors. I show novel bounds highlighting the key role of Lipschitz continuity in the compounding error phenomena, a problem that plagues long-horizon planning. As an alternative for single-step models, I then introduce a multi-step model that combats the compounding error problem by removing a major source of error in long-horizon planning.

Chapter 2

Background

In this chapter, I present the background that this thesis leans on. I first explain and formulate the reinforcement learning problem, and then present some fundamental reinforcement learning algorithms along with some basic properties needed for convergence of reinforcement learning. I also formulate the notion of smoothness and the notion of convexity.

2.1 The Reinforcement Learning Problem

Reinforcement learning (RL) is an area of artificial intelligence (AI) that revolves around the interaction between an environment and an agent whose goal is to collect as much reward as possible.

A core piece of the RL problem is the notion of a timestep. While time is inherently a continuous variable, in this thesis I assume that time is discrete. What I mean is that the agent is provided a snapshot of its state at a particular value of time t. I denote the provided snapshot at time t as the state s_t. The agent then takes an action a_t. This action is held constant until the next state is presented to the agent at time $t+1$. I refer to $[t,t+1)$ as the timestep t. The agent receives a scalar reward signal r_t which depends on s_t and a_t. The goal of the RL agent is to maximize the sum of rewards across future timesteps by finding a good action-selection strategy. The interaction is depicted in Figure 2.1.

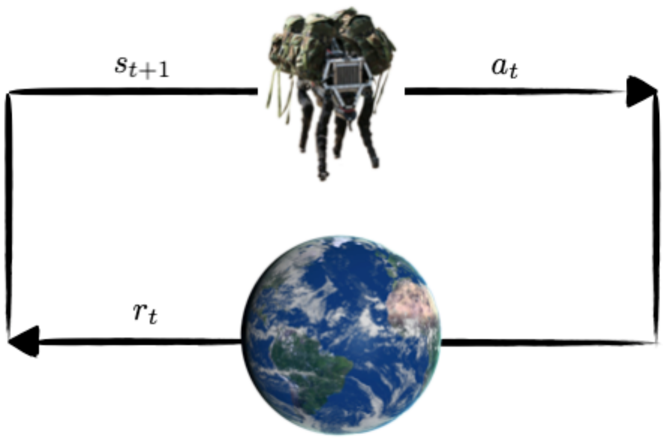


Figure 2.1: An illustration of the interaction between the RL agent and its environment.

The RL problem is typically formulated using Markov Decision Processes (MDPs) [83]. An MDP is usually specified by the following tuple: $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$. Here $\mathcal{S}$ and $\mathcal{A}$ denote the state space and the action space of the MDP. For some limited RL problems $\mathcal{S}$ and $\mathcal{A}$ are discrete and finite, but more generally one or both are continuous. When they are continuous, I assume that they are a subset of a Euclidean space, are closed and finite, therefore compact, and also that they are endowed by a distance metric.

The MDP model is comprised of two functions, namely the transition model $T : \mathcal{S} \times \mathcal{A} \rightarrow P(\mathcal{S})$, and the reward model $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$. The discount factor, $\gamma \in [0, 1)$, determines the importance of immediate reward as opposed to rewards received in the future. The goal of an RL agent is to find a policy $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ that collects highest sum of future discounted rewards. It is well-known that there exists a deterministic policy that is optimal in the sense of achieving maximum possible reward in all states.

For a state $s \in \mathcal{S}$, action $a \in \mathcal{A}$, and a policy π, we define the state-action value (Q) function:

$$Q^{\pi}(s,a) := \mathbf{E}_{\pi}\big[\sum_{i=t}^{T} \gamma^{i-t} r_i \mid s_t = s, a_t = a\big] \ .$$

We define Q^* as the maximum Q value of a state-action pair among all policies:

$$Q^*(s,a) := \max_{\pi} Q^{\pi}(s,a) \ .$$

Under a discrete state and action space, Bellman showed that the quantity Q^*, known as the optimal state–action value function, can be written recursively [22]:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \ \max_{a' \in \mathcal{A}} Q^*(s',a') \ . \tag{2.1}$$

Many RL algorithms could be thought of as different approaches to estimating Q^* from environmental interactions. Algorithms differ not just in terms of how they compute Q^*, but also in terms of they way they act in the environment based upon their experience. Regardless, these algorithms compute the fixed point of equation (2.1), known as the Bellman equation.

An analog of the above equation exists for problems with continuous states and actions:

$$Q^*(s,a) = R(s,a) + \gamma \int_{s' \in \mathcal{S}} T(s' \mid s,a) \ \max_{a' \in \mathcal{A}} Q^*(s',a') \ ds' \ . \tag{2.2}$$

Note that I have assumed that the action space is compact, and that Q^* is continuous in $\mathcal{A}$, which implies that at least one maximum exists due to the extreme-value theorem.

2.2 Planning versus Learning

Throughout this thesis, I frequently use two words, namely learning and planning, to refer to specific operations that an agent performs. In the context of RL, learning refers to the process of going from environmental interactions to an estimate of the value function (and/or a policy). In contrast, by planning I mean going from a model to the value function. The model could be provided to the agent a priori, or it could be an estimate learned from environmental interaction.

In majority of RL problems, the agent does not have access to the model, and should estimate the model from experience if necessary. In this sense, planning and model learning could be thought of as intermediate steps of some learning algorithms, though there exists learning algorithm that can learn a value function without a model as we shall see next.

2.3 Value Iteration

Value Iteration (VI) is a planning algorithm, meaning that it takes a model of the MDP $\langle T, R\rangle$ as input and computes the value function Q^*. VI is an intuitive and easy-to-implement algorithm, yet showing that it is a sound algorithm requires some work.

VI starts by a value function, denoted by $\widehat{Q}_0$, which is often initialized arbitrarily. It then proceeds by performing the following update at a specific iteration i:

$$\widehat{Q}_{i+1}(s,a) \leftarrow R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \max_{a' \in \mathcal{A}} \widehat{Q}_i(s',a') \ . \tag{2.3}$$

VI continues to perform these iterations until no progress is made. Lack of progress could be formulated as an extremely small norm difference between two consecutive value-function estimates.

While action maximization is a core component of Bellman equation, in some situations we may not be interested in finding the optimal value function. It is also possible that computing the optimal value function may be slow. We may want to use operators other than $\max_{a \in \mathcal{A}}$ in these cases.

Littman and Szepesvári generalized VI by replacing $\max_{a' \in \mathcal{A}}$ by any arbitrary operator $\bigotimes$ [63]. The resultant algorithm, which they coined as Generalized Value Iteration (GVI), proceeds as follows:

$$\widehat{Q}_{i+1}(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \bigotimes \widehat{Q}_i(s',\cdot) \ . \tag{2.4}$$

GVI, which collapses to VI when $\bigotimes = \max_{a'}$, will be a core piece of my thesis.

In absence of any prior knowledge about the problem at hand, the initialized value function $\widehat{Q}_0$ can be extremely far from the correct answer. It is natural, then, to ask whether this incredibly simple algorithm converges at all, and more importantly whether it can find the right solution when it does converge. In the next section we see that we can answer these questions in the affirmative, provided that $\bigotimes$ is sufficiently smooth.

2.3.1 Convergence of Value Iteration

It would be useful to know under which conditions, if at all, GVI can converge. To answer this question, I begin with the following lemma:

Lemma 2.3.1. *Assume that we have two Q functions denoted by $\widehat{Q}_i$ and $\widehat{Q}'_i$. Denote, by $\widehat{Q}_{i+1}$ and $\widehat{Q}'_{i+1}$, the resultant value functions having performed one iteration of GVI. The following property holds:*

$$\max_{s \in \mathcal{S}, a \in \mathcal{A}} \left| \widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a) \right| \leq \gamma \max_{s \in \mathcal{S}} \left| \bigotimes_a \widehat{Q}_i(s,\cdot) - \bigotimes_a \widehat{Q}'_i(s,\cdot) \right|$$

Proof.

$$\begin{aligned}
\big|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\big| &= \big|\mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \bigotimes \widehat{Q}_i(s',\cdot) \\
&\quad - \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \bigotimes \widehat{Q}'_i(s',\cdot)\big| \\
&= \gamma \Big| \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \big(\bigotimes \widehat{Q}_i(s',\cdot) - \bigotimes \widehat{Q}'_i(s',\cdot)\big)\Big| \\
&\leq \gamma \Big| \max_{s' \in \mathcal{S}} \big(\bigotimes \widehat{Q}_i(s',\cdot) - \bigotimes \widehat{Q}'_i(s',\cdot)\big)\Big| \\
&\leq \gamma \max_{s' \in \mathcal{S}} \Big| \bigotimes_{a'} \widehat{Q}_i(s',\cdot) - \bigotimes_{a'} \widehat{Q}'_i(s',\cdot)\Big|
\end{aligned}$$

So we can rewrite:

$$\max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\big| < \max_{s \in \mathcal{S}} \big| \bigotimes_{a} \widehat{Q}_i(s,\cdot) - \bigotimes_{a} \widehat{Q}'_i(s,\cdot)\big|$$

□

The next result we need is Banach's fixed-point theorem, which I have tailored to GVI:

Theorem 2.3.2. *Assume that we have two Q functions denoted by $\widehat{Q}_i$ and $\widehat{Q}'_i$. Denote by $\widehat{Q}_{i+1}$ and $\widehat{Q}'_{i+1}$ the resultant value functions having performed one iteration of GVI:*

$$\widehat{Q}_{i+1}(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \bigotimes \widehat{Q}_i(s',\cdot)\ . \tag{2.5}$$

If

$$\max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\big| \leq \gamma \max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\big|\ ,$$

then GVI is guaranteed to converge to the unique fixed-point of the equation:

$$\widehat{Q}(s,a) = \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \bigotimes \widehat{Q}(s',\cdot)\ . \tag{2.6}$$

Proof. I first show that there can't be more than one fixed point. To this end, let's assume two fixed points $\widehat{Q}$ and $\widehat{Q}'$ co-exist, and that the two fixed points are distinct. Then we have:

$$\begin{aligned}
\max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\big| &= \max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\big| \\
&\leq \gamma \max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\big|
\end{aligned}$$

From above, we can conclude that:

$$(1-\gamma) \max_{s \in \mathcal{S}, a \in \mathcal{A}} \big|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\big| \leq 0\ .$$

Because $(1-\gamma)$ is strictly positive, this implies that $\widehat{Q} = \widehat{Q}'$ which is clearly a contradiction, thus falsifying the hypothesis that two (or more) fixed-points can co-exist.

Next, I show that a fixed point exists. It is straightforward to show, using induction, that the following property holds for any two functions $\widehat{Q}$ and $\widehat{Q}'$:

$$\max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_{i+n}(s,a) - \widehat{Q}'_{i+n}(s,a)\right| \leq \gamma^n \max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right| \ ,$$

Also, for the sequence $\widehat{Q}_0, \widehat{Q}_1, \widehat{Q}_2, \ldots$ we have:

$$\begin{aligned}
\max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_{m+n}(s,a) - \widehat{Q}_m(s,a)\right| &\leq \sum_{j=0}^{n-1} \gamma \max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_{m+j+1}(s,a) - \widehat{Q}_{m+j}(s,a)\right| \\
&\leq \sum_{j=0}^{n-1} \gamma^{m+j+1} \max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_1(s,a) - \widehat{Q}_0(s,a)\right| \\
&\leq \gamma^m \sum_{j=0}^{n-1} \gamma^{j+1} \max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_1(s,a) - \widehat{Q}_0(s,a)\right|
\end{aligned}$$

As $m \to \infty$, the limit of the right hand side will be zero, therefore the sequence $\widehat{Q}_0, \widehat{Q}_1, \widehat{Q}_2, \ldots$ is a Cauchy sequence which is well-known to be convergent to some fixed-point. $\square$

I can now present the essential property that $\bigotimes$ needs to have so as to ensure convergence for GVI. Our desire is to have:

$$\max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\right| \leq \gamma \max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right| \ , \tag{2.7}$$

so that we could lean on theorem 2.3.1, but as lemma 2.3.2 showed we so far only have:

$$\max_{s\in\mathcal{S},a\in\mathcal{A}} \left|\widehat{Q}_{i+1}(s,a) - \widehat{Q}'_{i+1}(s,a)\right| \leq \gamma \max_{s\in\mathcal{S}} \left|\bigotimes_a \widehat{Q}_i(s,\cdot) - \bigotimes_a \widehat{Q}'_i(s,\cdot)\right| \ . \tag{2.8}$$

Notice that we assume $\gamma < 1$, so to go from (2.7) to (2.8), we just need to ensure that:

$$\left|\bigotimes_a \widehat{Q}_i(s,\cdot) - \bigotimes_a \widehat{Q}'_i(s,\cdot)\right| \leq \max_{a\in\mathcal{A}} \left|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right| \ ,$$

Or stated differently that

$$\frac{\left|\bigotimes_a \widehat{Q}_i(s,\cdot) - \bigotimes_a \widehat{Q}'_i(s,\cdot)\right|}{\max_{a\in\mathcal{A}} \left|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right|} \leq 1. \tag{2.9}$$

Any operator with the above property would be convergent when used in conjunction with GVI.

An example of such operator is $\max_{a\in\mathcal{A}}$:

$$\begin{aligned}
\left|\max_{a\in\mathcal{A}} \widehat{Q}_i(s,a) - \max_{a\in\mathcal{A}} \widehat{Q}'_i(s,a)\right| &\leq \left|\max_{a\in\mathcal{A}} \widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right| \\
&\leq \max_{a\in\mathcal{A}} \left|\widehat{Q}_i(s,a) - \widehat{Q}'_i(s,a)\right| \ .
\end{aligned}$$

Therefore, GVI with the max operator converges, and does so to the unique fixed-point of the following Bellman equation:

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \; \max_{a' \in \mathcal{A}} Q^*(s',a') \; . \tag{2.10}$$

If we use operators $\bigotimes$ other than max, then GVI would still converge so long as property (2.9) is satisfied, but the fixed point would be different than the optimal value function and needs to be further characterized.

Note that property (2.9) is sometimes referred to as non-expansion. That is, any operator $\bigotimes$ that satisfies (2.9) is a non-expansion and yields a convergent behavior when used in conjunction with GVI. When this property is lacking for an operator, we refer to the operator as an expansion. An expansion may be prone to misbehavior as I show later, but note that having a non-expansion is a sufficient property for GVI to converge. It is not necessary to have a non-expansion to ensure convergence, but then proving convergence could be difficult, if at all possible, and requires carefully inspecting the operator in question.

2.3.2 Q-learning and SARSA

So far my focus was on GVI, which is a planning algorithm. But what if a model of the environment was unavailable? Can we still learn the Q function? It turns out that it is still possible to find the fixed point of Belmman equation even when a model is unavailable.

In this case, a class of RL algorithms directly solve for the fixed point of the Bellman equation using environmental interactions. Q-learning [116], a notable example of these so-called model-free algorithms, learns an approximation of Q^*, denoted by $\widehat{Q}$. Assume that at a particular timestep t, the agent took the action a_t in state s_t and was moved to the state s_{t+1} having received the reward r_{t+1}. Q-learning then updates its estimate of the value function as follows:

$$\widehat{Q}(s_t,a_t) \leftarrow \alpha\big(r_{t+1} + \gamma \max_{a'} \widehat{Q}(s_{t+1},a')\big) + (1-\alpha)\widehat{Q}(s_t,a_t)$$

This update is intuitively plausible: the new value is a convex combination of the old value and the quantity $\big(r_{t+1} + \gamma \max_{a'} \widehat{Q}(s_{t+1},a')\big)$. This quantity could be thought of as an estimate of the discounted sum of the future rewards for the greedy policy with respect to $\widehat{Q}$. Taking some trivial steps, I rewrite the update as:

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\big(r_{t+1} + \gamma \max_{a'} \widehat{Q}(s_{t+1},a') - \widehat{Q}(s_t,a_t)\big) \; . \tag{2.11}$$

Intuitively, learning is over when this error is zero on average indicating that we have reached a fixed point.

In terms of action selection, Q-learning allows for any exploration policy to be utilized. Examples include epsilon greedy action-selection, soft argmax, and uncertainty-based policies. In this sense, Q-learning is identified as an off-policy algorithm, meaning that the action-selection strategy can in general be different than the operator (policy) used to learn based on the Q function of the next state s_{t+1}, i.e. $\max_{a'\in\mathcal{A}}$.

Another popular model-free algorithm is the SARSA algorithm. This algorithm proceeds with the following update rule in each timestep:

$$\widehat{Q}(s_t,a_t) \leftarrow \widehat{Q}(s_t,a_t) + \alpha\big(r_{t+1} + \gamma\widehat{Q}(s_{t+1},a_{t+1}) - \widehat{Q}(s_t,a_t)\big) \ ,$$

where the action $a_{t+1} \sim \pi(\cdot \mid s_{t+1})$ is the action that the agent took at timestep $t+1$.

Note that the update looks quite similar to the Q-learning algorithm presented above, but in contrast to Q-learning, SARSA is an on-policy algorithm. What I mean by on-policy is that the algorithm uses the same operation for action selection and learning. This stands in contrast with off-policy algorithms, such as Q-learning, in which two different policies may be employed for the purpose of learning and action selection. While this may seem like a tiny distinction, it is actually a subtle one in the sense that convergence guarantees are usually easier to attain for on-policy algorithms relative to off-policy ones.

Algorithm 1 Q-learning

```
Input: initial $\widehat{Q}(s,a)\ \forall s\in\mathcal{S}\ \forall a\in\mathcal{A}$, $\alpha$, and policy $\pi$
for each episode do
    Initialize $s$
    repeat
        $a \sim \pi(\cdot \mid s)$
        Take $a$, observe $r, s'$
        $\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a)$
            $+\ \alpha\Big[r + \gamma \max_{a'} \widehat{Q}(s',a') - \widehat{Q}(s,a)\Big]$
        $s \leftarrow s^{'}$
    until $s$ is terminal
end for
```

Algorithm 2 SARSA

```
Input: initial $\widehat{Q}(s,a)\ \forall s\in\mathcal{S}\ \forall a\in\mathcal{A}$, $\alpha$, and policy $\pi$
for each episode do
    Initialize $s$
    $a \sim \pi(\cdot \mid s)$
    repeat
        Take $a$, observe $r, s'; a^{'} \sim \pi(\cdot \mid s)$
        $\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a)$
            $+\ \alpha\Big[r + \gamma\widehat{Q}(s',a') - \widehat{Q}(s,a)\Big]$
        $s \leftarrow s^{'}, a \leftarrow a^{'}$
    until $s$ is terminal
end for
```

2.4 RL with Neural Networks

While it is valuable to study RL theory in the most clear and simple setting, i.e. problems with small and finite states and actions, in practice the number of states and actions of RL problems are usually so large that it is unrealistic to use a lookup table to learn a separate $\widehat{Q}$ for every state-action pair. This issue arises not just for learning the value function, but also for learning other key ingredients of RL such as models and policies.

Function approximation is a standard technique that can enable us to apply some of the existing RL algorithms to large settings. More specifically, our desire is to produce RL agents that can harness powerful function approximators such as neural networks to learn key ingredients of RL. In doing so, our goal is to develop agents that can effectively generalize experience from observed situations to unseen ones. Not every tabular RL algorithm has a clear extension to the function approximation case, but fortunately there exist some examples that integrate nicely with function approximation as I discuss next.

Consider the Q-learning algorithm with the update rule presented in (2.11). There exists an extension of Q-learning to the function approximation case [117]. Let $\widehat{Q}(\cdot,\cdot;\theta)$ be the neural network representing the Q function, where θ denotes the parameters of the neural network $\widehat{Q}$. The update rule of Q-learning with function approximation will be as follows:

$$\theta \leftarrow \theta + \alpha \, \big(r_t + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s_{t+1}, a'; \theta) - \widehat{Q}(s_t, a_t; \theta)\big) \, \nabla_\theta \widehat{Q}(s_t, a_t; \theta) \ , \tag{2.12}$$

where ∇_θ indicates the gradient with respect to parameters θ, and therefore, $\nabla_\theta \widehat{Q}(s_t, a_t; \theta)$ is a vector whose size equals the number of parameters of the function approximator (for example, the neural net).

Note that Q-learning's update rule (2.11) is agnostic to the choice of function class, and so in principle any differentiable function class could be used in conjunction with the above update rule to learn θ parameters. For example, Sutton used linear function approximation [101], and Konidaris et al., used Fourier basis functions [59]. One of the contributions of my thesis is to introduce a new function approximator that is conducive to problems with large actions. I will introduce this approximator in chapter 4.

In 2015, Mnih et al., [67] published breakthrough results where they showed that the simple update rule written above, when used in conjunction with deep convolutional neural networks, and a few other algorithmic tricks, can surpass human-level performance when learning to play Atari games from images.

This result became highly celebrated not just because solving some of these games were in and of itself impressive, but because this was perhaps the most convincing example so far showing that RL agents are able to learn the right kind of feature representations when endowed by powerful function approximators. This finding was significant because it obviated the burdensome practice of using human knowledge and domain expertise to find the right kinds of features and representations on a per-problem basis.

Another important contribution of Mnih et al., was to introduce (or at least re-popularize) algorithmic tricks that could hedge against problems that arise when doing RL with non-linear function approximation. Chief among these tricks was the notion of using a replay buffer and conducting experience replay, which, they argued, serves as a tool to a) reduce the highly correlated updates present when doing *online* updates (i.e. to update the Q function using the most recent sample and then discarding the sample.) and b) reduce the variance of gradient updates by averaging over multiple examples.

When using experience replay, the agent adds each observed experience $\langle s, a, r, s' \rangle$ to a replay buffer $\mathcal{B}$. In order to update the Q function, the agent samples a batch of these tuples, for example by defining a uniform distribution over examples in $\mathcal{B}$, and updates the parameters of the Q function θ by averaging updates over this batch of tuples.

$$\theta \leftarrow \theta + \alpha \sum_{\langle s,a,r,s' \rangle} \big(r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s', a'; \theta) - \widehat{Q}(s, a; \theta)\big) \ \nabla_\theta \widehat{Q}(s, a; \theta) \ . \tag{2.13}$$

The replay buffer $\mathcal{B}$ is bounded, so the agent discards tuples in the order received, akin to a deque, when the buffer reaches its capacity. The maximum size of the buffer, as well as the size of the sampled batch, could be thought of as hyper-parameters.

Mnih et al., [67] introduced another idea, namely a separate target network. Observe that the update rule (2.13) is akin to stochastic gradient descent (SGD), but there is a notable distinction. In particular, in SGD a label is provided to the agent and remains fixed across learning. However, in (2.13) the agent updates θ, and in doing so it also changes the target by changing $\max_{a' \in \mathcal{A}} \widehat{Q}(s, a'; \theta)$. This, Mnih et al., argued could engender extreme instability that needs to be addressed for effective learning.

To this end, Mnih et al., used a second set of weights θ^- that are different than θ, and are only updated periodically to match θ every C gradient updates, where C could be thought of as a hyper-parameter. The update thus becomes:

$$\begin{aligned} &\theta \leftarrow \theta + \alpha \sum_{\langle s,a,r,s' \rangle} \big(r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s', a'; \theta^-) - \widehat{Q}(s, a; \theta)\big) \ \nabla_\theta \widehat{Q}(s, a; \theta) \ . \\ &\theta^- \leftarrow \theta \quad \text{(after } C \text{ updates of } \theta) \end{aligned} \tag{2.14}$$

As I show later in the thesis, performance can heavily depend on choosing a good value of C.

Lilicrap et al., [61] proposed a different scheduling for updating θ^- parameters. Referred to as exponentially-weighted moving average, they proposed to perform one update for θ^- after one update for θ. They way in which they update θ^- is as follows:

$$\theta^- \leftarrow (1 - \alpha)\theta^- + \alpha\theta \ .$$

One way of thinking about this scheduling is as a way of ensuring that θ^- always equals a weighted average of all the previous θ values. The update constitutes a weighted average that puts exponentially larger weights behind more recent settings of θ. To see why this is the case, note that after i

updates of θ^- we get:

$$\begin{aligned}
\theta_i^- &= (1-\alpha)\theta_{i-1}^- + \alpha\theta_i \\
&= (1-\alpha)\big((1-\alpha)\theta_{i-2}^- + \alpha\theta_{i-1} + \alpha\theta_i \\
&= (1-\alpha)^2\theta_{i-2}^- + (1-\alpha)\alpha\theta_{i-1} + \alpha\theta_i \\
&\vdots \\
&= (1-\alpha)^i\theta_0 + \alpha\sum_{j=0}^{i-1}(1-\alpha)^j\theta_{i-j}\ ,
\end{aligned}$$

So the weight of θ belonging to each previous timestep is multiplied by a factor of $1-\alpha$ as we go backwards in time.

When using deep neural networks, another question becomes relevant, namely what kind of network architecture to use in conjunction with the RL algorithm? For the Q function, it is common to use a network architecture that takes the state vector as input, and outputs a vector whose size equals the number of actions. This is depicted in Figure 2.2. This is a sound and useful architecture in settings where the number of actions is small and finite. However, when the number of actions is large, or infinite, it is clearly impossible to utilize this network architecture.

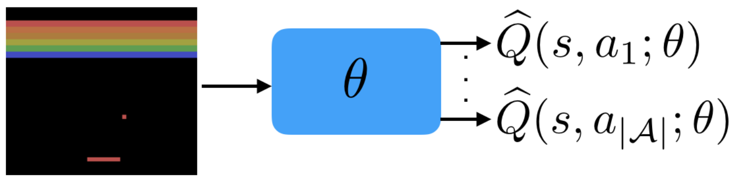

θ
$\widehat{Q}(s, a_1; \theta)$
$\widehat{Q}(s, a_{|\mathcal{A}|}; \theta)$

Figure 2.2: An illustration of the neural-network architecture proposed by Mnih et. al., for learning the Q function. This architecture became standard in RL solutions for discrete and finite action spaces.

Two standard alternatives co-exists. First, provided that the action space is bounded, and that we know the boundaries, we can always discretize the continuous action space and use the same architecture depicted above. Stated differently, we can reduce the continuous-action problem to a discrete one for which the above architecture is apt. However, this solution best underscores the presence of curse of dimensionality in RL, because to uniformly cover the action space via discretization, we need a network output whose size grows exponentially with the number of action dimensions.

Another common solution is to use a network architecture that takes as input, not just the state of the agent, but also its action (See Figure 2.3.) where the action input could also be thought of as a vector akin to the state input. This architecture, however, also comes at its own cost, namely that now computing the quantity, $\max_{a\in\mathcal{A}} \widehat{Q}(s,a)$ becomes difficult. This is significant because applying many RL solutions, such as Q-learning, hinges on the ability to efficiently and accurately solving this optimization problem. To see why solving this problem is hard, recall that a neural network can in general be a non-concave function of its inputs, therefore it is not obvious how to solve this kind of optimization problem using gradient-based techniques. I come back to this problem in chapter 4, and introduce a principled solution.

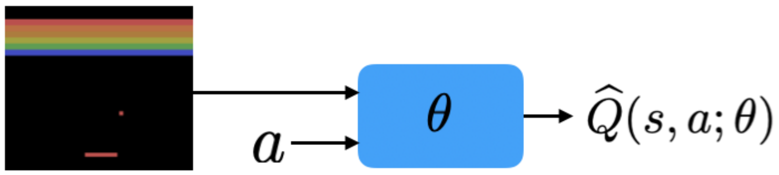


Figure 2.3: An illustration of a second architecture for learning the Q function using neural nets. The network takes a state and an action as input.

2.5 Smoothness

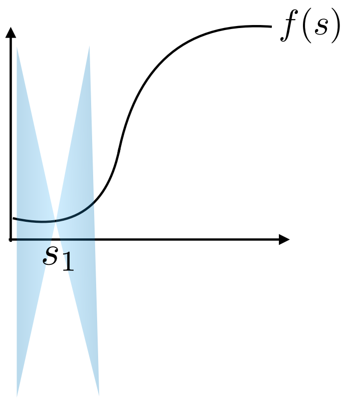


Figure 2.4: An illustration of Lipschitz continuity. Pictorially, knowing that f is K-Lipschitz means that f lies in between the two affine functions (colored in blue) with slope K. Since this is a one-dimensional function, the slope K is equivalent to the maximum derivative of the function on its domain.

In this thesis I leverage the "smoothness" of various ingredients of RL (operators, value functions, and models). I formulate this notion of smoothness using the mathematical tool of Lipschitz continuity defined momentarily.

Before presenting the definition, I provide an intuition for smoothness as characterized by Lipschitz continuity. I understand a smooth function to have the following property: For any two distinct points on the domain of the function, the line that connects the surface of the function at the two points should have a finite slope. The Lipschitz constant of the function is defined as the maximum value of this slope for any pair of points. A function with a Lipschitz constant K is said to be K-Lipschitz. I present an illustration in Figure 2.4 for a one-dimensional problem.

I now define a smooth (Lipschitz) function formally:

Definition 2.5.1. *Given two metric spaces (M_1, d_1) and (M_2, d_2) consisting of a space and a distance metric, a function $f : M_1 \to M_2$ is* Lipschitz continuous *(sometimes simply Lipschitz) if the Lipschitz constant, defined as*

$$K_{d_1,d_2}(f) := \sup_{s_1 \in M_1, s_2 \in M_1} \frac{d_2\big(f(s_1), f(s_2)\big)}{d_1(s_1, s_2)} \ , \tag{2.15}$$

is finite.

Equivalently, for a Lipschitz f,

$$\forall s_1, \forall s_2 \quad d_2\big(f(s_1), f(s_2)\big) \leq K_{d_1,d_2}(f)\ d_1(s_1, s_2) \ \ .$$

The concept of Lipschitz continuity is visualized in Figure 2.4.

A Lipschitz function f is called a *non-expansion* when $K_{d_1,d_2}(f) \leq 1$ and a *contraction* when $K_{d_1,d_2}(f) < 1$. Lipschitz continuity, in one form or another, has been a key tool in the theory of RL [25, 26, 63, 70, 41, 49, 84, 106, 77, 80, 79, 24, 21] and bandits [56, 32].

Below, I also define Lipschitz continuity over a subset of inputs.

Definition 2.5.2. *A function $f : M_1 \times \mathcal{A} \to M_2$ is* uniformly Lipschitz continuous *in $\mathcal{A}$ if*

$$K^{\mathcal{A}}_{d_1,d_2}(f) := \sup_{a \in \mathcal{A}} \sup_{s_1, s_2} \frac{d_2\big(f(s_1, a), f(s_2, a)\big)}{d_1(s_1, s_2)} \ , \tag{2.16}$$

is finite.

It is useful to know some properties of Lipschitz functions. For example, it is easy to show that for a scalar c, we have $K_{d_1,d_2}(cf) = cK_{d_1,d_2}(f)$. Also $K_{d_1,d_2}(f + g) \leq K_{d_1,d_2}(f) + K_{d_1,d_2}(g)$.

The following result relates Lipschitz continuity and compositions.

Lemma 2.5.1. *Define three metric spaces (M_1, d_1), (M_2, d_2), and (M_3, d_3). Define Lipschitz functions $f : M_2 \to M_3$ and $g : M_1 \to M_2$ with constants $K_{d_2,d_3}(f)$ and $K_{d_1,d_2}(g)$. Then, $h : f \circ g : M_1 \to M_3$ is Lipschitz with constant $K_{d_1,d_3}(h) \leq K_{d_2,d_3}(f) K_{d_1,d_2}(g)$.*

Proof.

$$\begin{aligned}
K_{d_1,d_3}(h) &= \sup_{s_1,s_2} \frac{d_3\Big(f\big(g(s_1)\big), f\big(g(s_2)\big)\Big)}{d_1(s_1, s_2)} \\
&= \sup_{s_1,s_2} \frac{d_3\Big(f\big(g(s_1)\big), f\big(g(s_2)\big)\Big)}{d_2\big(g(s_1), g(s_2)\big)} \frac{d_2\big(g(s_1), g(s_2)\big)}{d_1(s_1, s_2)} \\
&\leq \sup_{s_1,s_2} \frac{d_3\Big(f\big(g(s_1)\big), f\big(g(s_2)\big)\Big)}{d_2\big(g(s_1), g(s_2)\big)} \sup_{s_1,s_2} \frac{d_2\big(g(s_1), g(s_2)\big)}{d_1(s_1, s_2)} \\
&\leq \sup_{x_1,x_2} \frac{d_3\big(f(x_1), f(x_2)\big)}{d_2(x_1, x_2)} \sup_{s_1,s_2} \frac{d_2\big(g(s_1), g(s_2)\big)}{d_1(s_1, s_2)} \\
&= K_{d_2,d_3}(f) K_{d_1,d_2}(g)
\end{aligned}$$

□

2.5.1 Smoothness in Neural Networks

We can show that common operations of modern neural networks are Lipschitz. Notice that neural networks are generally composed of a couple of layers, each with a kernel and an activation function. We know from lemma 2.5.1 that composition of Lipschitz layers will be Lipschitz, so we just need to show that each layer is itself Lipschitz, and the Lipschitzness of the entire network will follow regardless of the number of layers and/or operation types.

Below, we derive the Lipschitz constant for various functions.
ReLu non-linearity We show that $\text{ReLu} : \mathbb{R}^n \to \mathbb{R}^n$ has Lipschitz constant 1 for p.

$$\begin{aligned}
K_{\|.\|_p,\|.\|_p}(\text{ReLu}) &= \sup_{x_1,x_2} \frac{\|\text{ReLu}(x_1) - \text{ReLu}(x_2)\|_p}{\|x_1 - x_2\|_p} \\
&= \sup_{x_1,x_2} \frac{(\sum_i |\text{ReLu}(x_1)_i - \text{ReLu}(x_2)_i|^p)^{\frac{1}{p}}}{\|x_1 - x_2\|_p} \\
&\quad \text{(We can show that } |\text{ReLu}(x_1)_i - \text{ReLu}(x_2)_i| \leq |x_{1,i} - x_{2,i}| \text{ and so) :} \\
&\leq \sup_{x_1,x_2} \frac{(\sum_i |x_{1,i} - x_{2,i}|^p)^{\frac{1}{p}}}{\|x_1 - x_2\|_p} \\
&= \sup_{x_1,x_2} \frac{\|x_1 - x_2\|_p}{\|x_1 - x_2\|_p} = 1
\end{aligned}$$

Matrix multiplication Let $W \in \mathbb{R}^{n\times m}$. We derive the Lipschitz continuity for the function $\times W(x) = Wx$.
For $p = \infty$ we have:

$$\begin{aligned}
K_{\|\|_\infty,\|\|_\infty}\big(\times W(x_1)\big) &= \sup_{x_1,x_2} \frac{\|\times W(x_1) - \times W(x_2)\|_\infty}{\|x_1 - x_2\|_\infty} \\
&= \sup_{x_1,x_2} \frac{\|Wx_1 - Wx_2\|_\infty}{\|x_1 - x_2\|_\infty} \\
&= \sup_{x_1,x_2} \frac{\|W(x_1 - x_2)\|_\infty}{\|x_1 - x_2\|_\infty} \\
&= \sup_{x_1,x_2} \frac{\sup_j |W_j(x_1 - x_2)|}{\|x_1 - x_2\|_\infty} \\
&\leq \sup_{x_1,x_2} \frac{\sup_j \|W_j\| \, \|x_1 - x_2\|_\infty}{\|x_1 - x_2\|_\infty} \quad \text{(Hölder's inequality)} \\
&= \sup_j \|W_j\|_1 \ ,
\end{aligned}$$

where W_j refers to jth row of the weight matrix W.

Similarly, for $p = 1$ we have:

$$\begin{aligned}
K_{\|\|_1,\|\|_1}\big(\times W(x_1)\big) &= \sup_{x_1,x_2} \frac{\|{\times W(x_1)} - {\times W(x_2)}\|_1}{\|x_1-x_2\|_1}\\
&= \sup_{x_1,x_2} \frac{\|Wx_1 - Wx_2\|_1}{\|x_1-x_2\|_1}\\
&= \sup_{x_1,x_2} \frac{\|W(x_1-x_2)\|_1}{\|x_1-x_2\|_1}\\
&= \sup_{x_1,x_2} \frac{\sum_j |W_j(x_1-x_2)|}{\|x_1-x_2\|_1}\\
&\leq \sup_{x_1,x_2} \frac{\sum_j \|W_j\|_\infty \|x_1-x_2\|_1}{\|x_1-x_2\|_1}\\
&= \sum_j \|W_j\|_\infty \ ,
\end{aligned}$$

and finally for $p = 2$:

$$\begin{aligned}
K_{\|\|_2,\|\|_2}\big(\times W(x_1)\big) &= \sup_{x_1,x_2} \frac{\|{\times W(x_1)} - {\times W(x_2)}\|_2}{\|x_1-x_2\|_2}\\
&= \sup_{x_1,x_2} \frac{\|Wx_1 - Wx_2\|_2}{\|x_1-x_2\|_2}\\
&= \sup_{x_1,x_2} \frac{\|W(x_1-x_2)\|_2}{\|x_1-x_2\|_2}\\
&= \sup_{x_1,x_2} \frac{\sqrt{\sum_j |W_j(x_1-x_2)|^2}}{\|x_1-x_2\|_2}\\
&\leq \sup_{x_1,x_2} \frac{\sqrt{\sum_j \|W_j\|_2^2 \|x_1-x_2\|_2^2}}{\|x_1-x_2\|_2}\\
&= \sqrt{\sum_j \|W_j\|_2^2} \ .
\end{aligned}$$

Vector addition We show that $+b : \mathbb{R}^n \to \mathbb{R}^n$ has Lipschitz constant 1 for $p = 0, 1, \infty$ for all $b \in \mathbb{R}^n$.

$$\begin{aligned}
K_{\|\cdot\|_p,\|\cdot\|_p}(\text{ReLu}) &= \sup_{x_1,x_2} \frac{\| + b(x_1) - +b(x_2)\|_p}{\|x_1-x_2\|_p}\\
&= \sup_{x_1,x_2} \frac{\|(x_1+b) - (x_2+b)\|_p}{\|x_1-x_2\|_p}\\
&= \frac{\|x_1-x_2\|_p}{\|x_1-x_2\|_p} = 1
\end{aligned}$$

Any neural network that is comprised of the above operations is therefore Lipschitz. Controlling the Lipschitz constant of neural networks has become a popular topic in the neural-network literature. Example of the kinds of applications of this idea include regularization [74], and learning generative

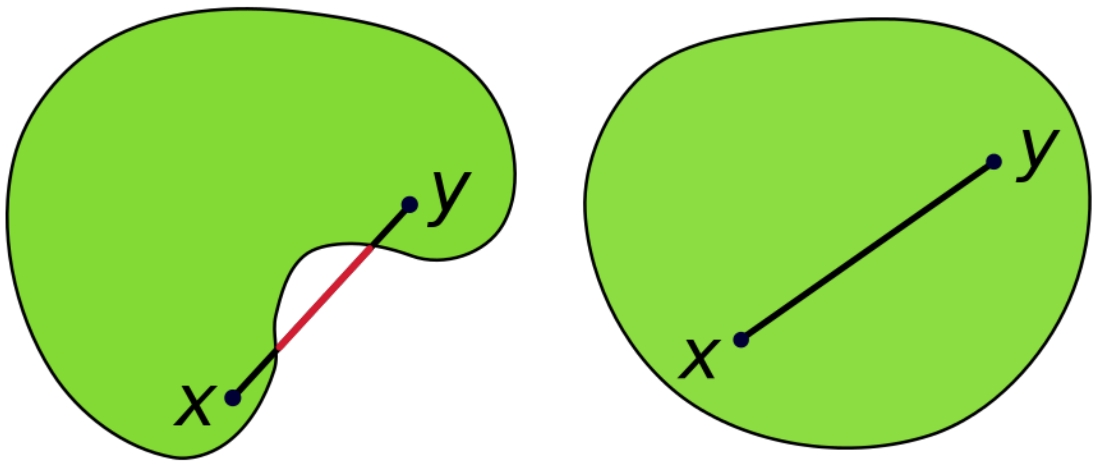


Figure 2.5: An example of a non-convex (left) and a convex (right) set.

adversarial networks [5]. In chapter 5 I will show that controlling Lipschitz continuity of approximate models can be conducive to long-horizon planning in model-based RL.

2.6 Convexity

This thesis leans on the rich theory of convex functions and convex optimization techniques. In this section I provide a brief introduction to some of the important ideas in convex optimization, specially those that pertain to RL, but for a thorough treatment of this subject see Boyd and Vandenberghe [27].

2.6.1 Convex Sets

Consider a set of points $\mathcal{X}$. The set $\mathcal{X}$ is said to be convex if for any distinct pair of points $x \in \mathcal{X}$ and $y \in \mathcal{X}$, and a scalar $c \in (0, 1)$:

$$cx + (1 - c)y \in \mathcal{X} \ .$$

In Figure 2.5 I provide two examples to visualize the difference between convex and non-convex sets. The examples are taken from Wikipedia.com. Next I define convex functions.

2.6.2 Zero-Order Convexity

Define a function $f : \mathcal{X} \to \mathcal{R}$. We say that f is convex if a) its domain $\mathcal{X}$ is a convex set, and b) for any two points $x \in \mathcal{X}$ and $y \in \mathcal{X}$ and a scalar $c \in (0, 1)$ we have:

$$f\big(cx + (1 - c)y\big) \leq cf(x) + (1 - c)f(y) \ .$$

Note that a convex function may not necessarily be differentiable. As a concrete example, we can show that the non-differentiable function $f(x) = \max_i x_i$, also known as the ∞-norm, is in fact a convex function:

$$\begin{aligned} f\big(cx + (1-c)y\big) &= \max_i cx_i + (1-c)y_i \\ &\leq \max_i cx_i + \max_i (1-c)y_i \\ &= c\max_i x_i + (1-c)\max_i y_i \\ &= cf(x) + (1-c)f(y). \end{aligned}$$

More generally, every p-norm, $f(x) = \|x\|$, is convex:

$$\begin{aligned} f\big((1-c)x + cy\big) &= \|(1-c)x + cy\| \\ &\leq \|(1-c)x\| + \|cy\| \quad \text{(due to triangle inequality of norms)} \\ &= (1-c)\,\|x\| + c\,\|y\| \\ &= (1-c)f(x) + cf(y) \end{aligned}$$

Conversely, a function is said to be concave if a) its domain $\mathcal{X}$ is a convex set, and b) for any two points $x \in \mathcal{X}$ and $y \in \mathcal{X}$ and a scalar $c \in (0,1)$ we have:

$$f\big(cx + (1-c)y\big) \geq cf(x) + (1-c)f(y)$$

A function is both concave and convex if and only if it is affine.

2.6.3 First-Order Convexity

It can be shown that, for a differentiable function, satisfying the zero-order condition is equivalent to satisfying the following condition:

$$\forall x, y \quad f(y) \geq \nabla f(x)^\top (y - x) + f(x) \ ,$$

i.e. the first-order Taylor expansion of f at any point x provides a lower bound for the value of the function $f(y)$ for any point y within the domain of the function.

As an immediate consequence of this condition, note that for a differentiable and convex function, finding the global minimum of the function is equivalent to finding a point x with $\nabla f(x) = 0$, since $\forall y \ f(y) \geq f(x)$ in light of the above inequality.

2.6.4 Second-Order Convexity

It can also be shown that the zero and first-order conditions above are equivalent to the following second-order condition:

$$\forall x \in \mathcal{X} \quad \nabla^2 f(x) \geq 0 \ .$$

Similarly the function is identified as concave if $\forall x \in \mathcal{X} \quad \nabla^2 f(x) \geq 0$. The quantity $\nabla^2 f(x)$ is often referred to as the Hessian of f. Pictorially, this means that the function has a positive curvature at all points within its domain.

The second-order condition often serves a pretty useful test for proving that a function is convex. As a relevant example, consider the following function:

$$f(x) = \frac{\log \frac{1}{n} \sum_i e^{\omega x_i}}{\omega} \ ,$$

where ω is a constant. This function has various applications in electrical engineering and optimization, and is considered to be a differentiable alternative for maximum.

It is straightforward to show that the Hessian of this function is:

$$\nabla^2 f(x) = \frac{\omega}{(1^\top z)^2}(1^\top z \ \mathrm{diag}(z) - zz^\top) \ ,$$

where $z = e^{wx_i}$.

To show that $\nabla^2 f(x) \geq 0$, we must show that for each vector v we have:

$$v^\top \nabla^2 f(x) v = \frac{\omega}{(1^\top z)^2}\Big((\sum_{i=1}^{n} z_i)(\sum_{i=1}^{n} v_i^2 z_i) - (\sum_{i=1}^{n} v_i z_i)^2 \Big) \geq 0,$$

which follows from Cauchy-Schwartz inequality [27] so long as $\omega \geq 0$. By a similar argument, the function will be concave if $\omega \leq 0$.

2.6.5 Jansen's Inequality

An important property of convex functions is that the expected value of a function is larger than the function applied to the expected value, i.e.:

$$\mathbf{E}_x[f(x)] \geq f(\mathbf{E}[x]) \quad \text{if } f \text{ is convex.}$$

Similarly, for a concave function we have:

$$\mathbf{E}_x[f(x)] \leq f(\mathbf{E}[x]) \quad \text{if } f \text{ is concave.}$$

2.6.6 Convex Conjugate

Let $f : \mathcal{X} \to \mathcal{R}$ be an arbitrary function. We define $f^* : \mathcal{R}^n \to \mathcal{R}$, known as the conjugate of f, as follows:

$$f^*(y) = \max_x \left(y^\top x - f(x) \right) \ .$$

The conjugate is always convex, even if f is not convex, because the conjugate is defined as the maximum over a bunch of linear functions, which is known to be convex.

It is sometimes possible to analytically derive the convex conjugate of a function, specially if the original function $f(x)$ is itself convex. As a concrete example, consider the log-sum-exp function:

$$f(x) = \log \sum_{i=1}^{n} e^{x_i} \ .$$

Using the above definition, the convex conjugate of this function is defined as follows:

$$f^*(y) = \max_x \left(y^\top x - \log \sum_{i=1}^{n} e^{x_i} \right) \ . \tag{2.17}$$

A sound approach to finding the solution involves computing and setting the derivative of the function to zero with respect to each x_i. The reason this works is that a) $y^\top x$ is affine (therefore concave), b) $\log \sum_{i=1}^{n} x_i$ is convex and so its negative is concave and c) the sum of two concave functions is also concave, so zero derivative ensures globally maximizing the functions as I explained when introducing the first-order convexity condition.

Setting the derivative to zero, we get:

$$\frac{\partial y^\top x - \log \sum_{i=1}^{n} x_i}{\partial x_i} = y_i - \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} = 0 \implies y_i = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}$$

It is clear now that a) $\sum_{i=1}^{n} y_i = 1$ and b) $\forall i \ 0 \leq y_i \leq 1$.

Taking log from both sides,

$$\log y_i = \log \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} \ ,$$

and so:

$$x_i = \log y_i - \log \sum_{i=1}^{n} e^{x_i} \ .$$

Plugging x_i into (2.17), we get:

$$\begin{aligned}
f^*(y) &= \sum_{i=1}^{n} y_i (\log y_i - \log \sum_{i=1}^{n} e^{x_i}) - \log \sum_{i=1}^{n} e^{x_i} \\
&= \sum_{i=1}^{n} y_i \log y_i - \sum_{i=1}^{n} y_i \log \sum_{i=1}^{n} e^{x_i} - \log \sum_{i=1}^{n} e^{x_i} \\
&= \sum_{i=1}^{n} y_i \log y_i - \log \sum_{i=1}^{n} e^{x_i} \underbrace{\sum_{i=1}^{n} y_i}_{=1} - \log \sum_{i=1}^{n} e^{x_i} \\
&= \sum_{i=1}^{n} y_i \log y_i - \log \sum_{i=1}^{n} e^{x_i} - \log \sum_{i=1}^{n} e^{x_i} \\
&= \sum_{i=1}^{n} y_i \log y_i
\end{aligned}$$

The conjugate of log-sum-exp, is therefore, the entropy function. The conjugate is only defined on the probability simplex (which is a convex set), and equals ∞ otherwise. It is also easy to show that entropy is a convex function. This was to be expected because the conjugate is always convex as argued above.

2.6.7 Convexity and Composition

Suppose that we have two functions $g : \mathcal{R} \rightarrow \mathcal{R}$ and a second function $h = f : \mathcal{R} \rightarrow \mathcal{R}$. We are interested to know under which conditions the composition $f \circ g$ is also convex. For simplicity, we assume that both functions are twice differentiable, and relax this assumption later.

Computing the second derivative with respect to x, we get:

$$g^{''}(x)f^{'}\big(g(x)\big) + (g^{'}(x))^2 f^{''}\big(g(x)\big)$$

Recall that our desire is to ensure that the second derivative is positive. We can obtain this property in two ways:

- both g and f are convex, and f is monotonically non-decreasing.
- g is concave, and f is convex and monotonically non-increasing.

It is also easy to show that under the following conditions, the composition will be concave:

- both g and f are concave, and f is monotonically non-decreasing.
- g is convex, and f is concave and monotonically non-decreasing.

In fact, as Boyd and Vandenberghe show, it is not necessary to assume that the domain is just $\mathcal{R}$ or even that the two functions are differentiable. We can get similar conditions in the absence of these assumptions.

2.6.8 Convexity in Neural Networks

Recall that neural networks are comprised of compositions of many operations. In light of the convexity conditions in the previous section, it would be natural to ask if we can restrict the parameters of a neural network in a way that ensures convexity (or concavity).

Amos et. al., answer this question in the affirmative by providing very simple constraints that ensure that a neural network remains convex [4]. Consider a single layer of a neural network with weights $\langle W, b \rangle$, input x, and activation function f, and output y. Then the operation of the layer could be formulated as follows:

$$y = f(Wx + b)$$

Imagine that the weight matrix W is only comprised of positive weights. It is clear that $Wx + b$ will be a monotonically non-decreasing, and also an affine (therefore convex) function. If we use a convex and non-decreasing activation function f, then the entire layer will be both convex and monotonically non-decreasing. Many common activation functions in neural-network literature, such as Rectified Linear Units (ReLU) can be shown to be convex and non-decreasing. It is then easy to generalize the argument to networks with multiple layers since each individual layer is convex and non-decreasing. See Amos et. al., [4] for slightly more general conditions.

Chapter 3

Smoothness for Convergent Reinforcement Learning

It is imperative that RL agents can maintain a balance between exploration and exploitation. This tensions becomes increasingly more challenging to address in domains with large states and actions. One scalable approach to this problem is to use soft approximations of maximum that hedge against problems that arise from putting all of one's weight behind a single decision. These so called *soft-max* operators applied to a finite set of values act somewhat like the maximization function and somewhat like an average. The Boltzmann operator is the most commonly used softmax operator in this setting, but in this chapter I show that this operator is prone to misbehavior because it is not sufficiently smooth. I study an alternative softmax operator that, among other properties, has a Lipschitz constant that is less than or equal to 1, ensuring a convergent behavior in learning and planning. I introduce variants of SARSA and Q-learning that utilize the new operator, and by doing so,v perform competitively when integrated with deep neural networks.

This chapter is mainly based on a paper written by Michael Littman and myself [7], and a second paper written by Seungchan Kim, George Konidaris, Michael Littman, and myself [55].

3.1 Introduction

There is a fundamental tension in decision making between choosing the action that has highest expected utility and avoiding "starving" the other actions. The issue arises in the context of the exploration–exploitation dilemma [109], non-stationary decision problems [99], and when interpreting observed decisions [17]. Standard approaches exist that can nicely deal with this problem [98], but these approaches are often ill-suited for problems with large or infinite state and action spaces.

A heuristic yet effective and commonly-used approach to addressing the tension is the use of *softmax* operators for value-function optimization, and softmax policies for action selection. Examples of the kinds of algorithms that utilizie softmax include value-based methods such as Q-learning [117], SARSA [86] or expected SARSA [102, 114], and policy-search methods such as REINFORCE [119] and actor critic [19].

While the term softmax operator is quite self-explanatory, I understand an ideal softmax operator as a parameterized set of operators that:

1. has parameter settings that allow it to approximate maximization arbitrarily accurately to perform reward-seeking behavior;

2. is sufficiently smooth ($K_{\|\cdot\|_\infty,|\cdot|}$(operator) ≤ 1 where K denotes Lipschitz constant of the softmax operator) ensuring convergence to a unique fixed point as I indicated in the Background section. An operator with such property is identified as a non-expansion;

3. is differentiable to make it possible to improve via gradient-based optimization; and

4. avoids the starvation of non-maximizing actions.

Let $\mathbf{X} = x_1, \ldots, x_n$ be a vector of values. We define the following operators:

- $\max(\mathbf{X}) = \max_{i \in \{1,\ldots,n\}} x_i$,
- $\text{mean}(\mathbf{X}) = \frac{1}{n} \sum_{i=1}^{n} x_i$,
- $\text{eps}_\epsilon(\mathbf{X}) = \epsilon \, \text{mean}(\mathbf{X}) + (1-\epsilon) \max(\mathbf{X})$,
- $\text{boltz}_\beta(\mathbf{X}) = \frac{\sum_{i=1}^{n} x_i \; e^{\beta x_i}}{\sum_{i=1}^{n} e^{\beta x_i}}$.

The first operator, $\max(\mathbf{X})$, is a non-expansion [63] as I showed in the background section. However, it is non-differentiable (Property 3), and ignores non-maximizing selections (Property 4). Obviouly it also lacks any parameter setting to allow more flexible interpretations.

The next operator, $\text{mean}(\mathbf{X})$, computes the average of its inputs. It is differentiable and, like any operator that takes a fixed convex combination of its inputs, is a non-expansion. However, it does not allow for maximization (Property 1).

The third operator $\text{eps}_\epsilon(\mathbf{X})$, commonly referred to as epsilon greedy [102], interpolates between max and mean. The operator is a non-expansion, because it is a fixed and convex combination of two non-expansions. But it is non-differentiable (Property 3).

The Boltzmann operator $\text{boltz}_\beta(\mathbf{X})$ is differentiable. It also approximates max as $\beta \to \infty$, and mean as $\beta \to 0$. However, it is not a non-expansion (Property 2), and therefore, prone to misbehavior as I will show later. Nevertheless, this operator has been widely common in the RL literature both for value-function optimization and action selection.

In the following sections, I provide a simple example illustrating why the non-expansion property is important, especially in the context of planning and on-policy learning. I then present a new softmax operator that is a non-expansion in contrast to the Boltzmann softmax. I prove several critical properties of this new operator, introduce an information-theoretic policy, and present empirical results for settings where we combine the new operator with powerful neural network function approximators. Results indicate that the new operator could serve as a competent and a theoretically more plausible alternative for Boltzmann softmax.

3.2 Boltzmann Misbehaves

I first show that boltz_β can lead to problematic behavior. To this end, I ran SARSA with Boltzmann softmax policy (Algorithm 3) on the MDP shown in Figure 3.1. The edges are labeled with a transition probability (unsigned) and a reward number (signed). Also, state s_2 is a terminal state, so I only consider two action values, namely $\widehat{Q}(s_1, a)$ and $\widehat{Q}(s_2, b)$. Recall that the Boltzmann softmax policy assigns the following probability to each action: $\pi(a \mid s) = \frac{e^{\beta \widehat{Q}(s,a)}}{\sum_a e^{\beta \widehat{Q}(s,a)}}$.

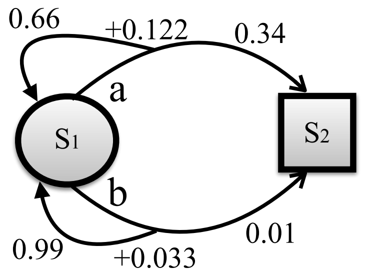


Figure 3.1: A simple MDP with two states, two actions, and $\gamma = 0.98$. The use of a Boltzmann softmax policy is not sound in this simple domain, and can lead into undesirable behavior.

Algorithm 3 SARSA with Boltzmann softmax policy

```
**Input:** initial $\widehat{Q}(s,a)\ \forall s \in \mathcal{S}\ \forall a \in \mathcal{A}$, $\alpha$, and $\beta$
**for** each episode **do**
    Initialize $s$
    $a \sim$ Boltzmann with parameter $\beta$
    **repeat**
        Take action $a$, observe $r, s'$
        $a' \sim$ Boltzmann with parameter $\beta$
        $\widehat{Q}(s,a) \leftarrow \widehat{Q}(s,a) + \alpha\left[r + \gamma\widehat{Q}(s',a') - \widehat{Q}(s,a)\right]$
        $s \leftarrow s', a \leftarrow a'$
    **until** $s$ is terminal
**end for**
```

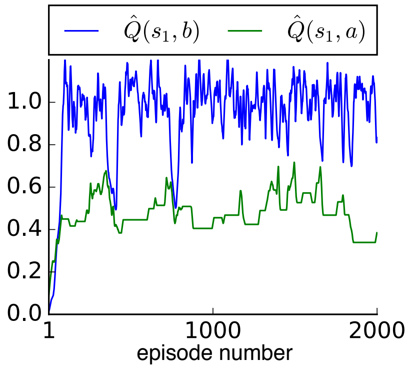


Figure 3.2: Values estimated by SARSA with Boltzmann softmax (Algorithm 3). The algorithm never achieves stable values.

In Figure 3.2, I plot state–action value estimates at the end of each episode of a single run (smoothed by averaging over ten consecutive points). I set $\alpha = .1$ and $\beta = 16.55$. Estimates remain unstable.

SARSA is known to converge in the tabular setting using ϵ-greedy exploration [63], under decreasing exploration [95], and to a region in the function-approximation setting [45]. However, this example is the first, to our knowledge, to show that SARSA fails to converge in the tabular setting with Boltzmann policy. As I argue next, Boltzmann exhibits this undesirable behavior because the operator is not sufficiently smooth. The next section provides background for our analysis of this example.

3.3 Boltzmann Has Multiple Fixed Points

Although it has been known that the Boltzmann operator is not a non-expansion [64], in our paper we showed the first published example of an MDP for which two distinct fixed points exist [7].

Shown in Figure 3.1, is an MDP for which we get two distinct fixed points if we run GVI under boltz_β. This is depicted in Figure 3.3.

I also show, in Figure 3.4, a vector field visualizing GVI updates under $\text{boltz}_{\beta=16.55}$. The updates can move the current estimates farther from the fixed points, which can only occur if the operator is not a non-exapansion.

The behavior of SARSA (Figure 3.2) results from the algorithm stochastically bouncing back and forth between the two fixed points. When the learning algorithm performs a sequence of noisy updates, it moves from a fixed point to the other. As I will show later, planning will also progress extremely slowly near the fixed points. The lack of the non-expansion property leads to multiple fixed points and ultimately a misbehavior in learning and planning.

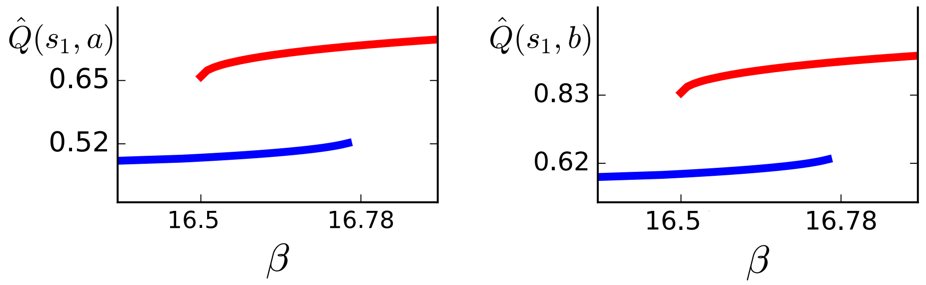


Figure 3.3: Fixed points of GVI under boltz$_\beta$ for varying β. Two distinct fixed points (red and blue) co-exist for a range of β.

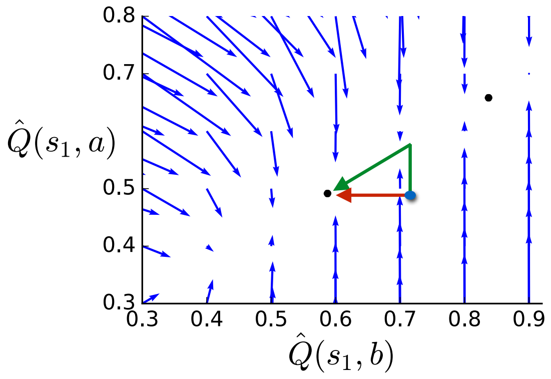


Figure 3.4: A vector field showing GVI updates under boltz$_{\beta=16.55}$. Fixed points are marked in black. For some points, such as the large blue point, updates can move the current estimates farther from the fixed points. Also, for points that lie in between the two fixed-points, progress is extremely slow.

3.4 Mellowmax and Its Properties

I advocate for an alternative softmax operator defined as follows:

$$\mathrm{mm}_\omega(\mathbf{X}) = \frac{\log(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i})}{\omega} \ ,$$

which can be viewed as a particular instantiation of the quasi-arithmetic mean [20]. It can also be derived from information theoretical principles as a way to prevent policies with high entropy [111, 85, 42].

I prove that mm_ω is a non-expansion (Property 2), and therefore, GVI and SARSA under mm_ω are guaranteed to converge to a unique fixed point.

Let $\mathbf{X} = x_1, \ldots, x_n$ and $\mathbf{Y} = y_1, \ldots, y_n$ be two vectors of values. Let $\Delta_i = |x_i - y_i|$ for $i \in \{1, \ldots, n\}$ be the difference of the ith components of the two vectors. Also, let i^* be the index with the maximum component-wise difference, $i^* = \text{argmax}_i\, \Delta_i$. For simplicity, we assume that i^* is unique and $\omega > 0$. Also, without loss of generality, we assume that $x_{i^*} - y_{i^*} \geq 0$. It follows that:

$$\begin{aligned}
\big|\text{mm}_\omega(\mathbf{X}) - \text{mm}_\omega(\mathbf{Y})\big| &= \big|\log(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i})/\omega \;\; - \log(\frac{1}{n}\sum_{i=1}^{n} e^{\omega y_i})/\omega\;\big| \\
&= \big|\log\frac{\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i}}{\frac{1}{n}\sum_{i=1}^{n} e^{\omega y_i}}/\omega\;\big| \\
&= \big|\log\frac{\sum_{i=1}^{n} e^{\omega\big(y_i+\Delta_i\big)}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega\;\big| \\
&\leq \big|\log\frac{\sum_{i=1}^{n} e^{\omega\big(y_i+\Delta_{i^*}\big)}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega\;\big| \\
&= \big|\log\frac{e^{\omega\Delta_{i^*}}\sum_{i=1}^{n} e^{\omega y_i}}{\sum_{i=1}^{n} e^{\omega y_i}}/\omega\;\big| \\
&= \big|\log(e^{\omega\Delta_{i^*}})/\omega\big| = \big|\Delta_{i^*}\big| = \max_i \big|x_i - y_i\big|\;,
\end{aligned}$$

allowing us to conclude that mellowmax is a non-expansion, i.e. $\forall\omega\; K_{\|\cdot\|_\infty,|\cdot|},(mm_\omega) \leq 1$.

3.4.1 Maximization

Mellowmax includes parameter settings that allow for maximization (Property 1) as well as for minimization. In particular, as ω goes to infinity, mm_ω acts like max.

Let $m = \max(\mathbf{X})$ and let $W = |\{x_i = m | i \in \{1,\ldots,n\}\}|$. Note that $W \geq 1$ is the number of maximum values ("winners") in $\mathbf{X}$. Then:

$$\begin{aligned}
\lim_{\omega\to\infty}\text{mm}_\omega(\mathbf{X}) &= \lim_{\omega\to\infty}\frac{\log(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i})}{\omega} \\
&= \lim_{\omega\to\infty}\frac{\log(\frac{1}{n}e^{\omega m}\sum_{i=1}^{n} e^{\omega(x_i-m)})}{\omega} \\
&= \lim_{\omega\to\infty}\frac{\log(\frac{1}{n}e^{\omega m}W)}{\omega} \\
&= \lim_{\omega\to\infty}\frac{\log(e^{\omega m}) - \log(n) + \log(W)}{\omega} \\
&= m + \lim_{\omega\to\infty}\frac{-\log(n) + \log(W)}{\omega} \\
&= m = \max(\mathbf{X})\;.
\end{aligned}$$

That is, the operator acts more and more like pure maximization as the value of ω is increased. Conversely, as ω goes to $-\infty$, the operator approaches the minimum.

3.4.2 Derivatives

We can take the derivative of mellowmax with respect to each one of the arguments x_i and for any non-zero ω:

$$\frac{\partial\text{mm}_\omega(\mathbf{X})}{\partial x_i} = \frac{e^{\omega x_i}}{\sum_{i=1}^{n} e^{\omega x_i}} \geq 0\;.$$

Note that the operator is non-decreasing in each component of $\mathbf{X}$.

3.4.3 Averaging

Because of the division by ω in the definition of mm_ω, the parameter ω cannot be set to zero. However, we can examine the behavior of mm_ω as ω approaches zero and show that the operator computes an average in the limit.

Since both the numerator and denominator go to zero as ω goes to zero, we will use L'Hôpital's rule and the derivative given in the previous section to derive the value in the limit:

$$\begin{aligned}
\lim_{\omega\to 0} \mathrm{mm}_\omega(\mathbf{X}) &= \lim_{\omega\to 0} \frac{\log(\frac{1}{n}\sum_{i=1}^n e^{\omega x_i})}{\omega} \\
&\overset{\text{L'Hôpital}}{=} \lim_{\omega\to 0} \frac{\frac{1}{n}\sum_{i=1}^n x_i e^{\omega x_i}}{\frac{1}{n}\sum_{i=1}^n e^{\omega x_i}} \\
&= \frac{1}{n}\sum_{i=1}^n x_i \\
&= \mathrm{mean}(\mathbf{X})\ .
\end{aligned}$$

That is, as ω gets closer to zero, $\mathrm{mm}_\omega(\mathbf{X})$ approaches the mean of the values in $\mathbf{X}$.

3.4.4 Monotonic non-Decrease

For any $\omega \geq 0$ and any x, $mm_\omega(x)$ is non-decreasing with respect to ω.

Proof: Let $\omega_2 > \omega_1 > 0$.
We want to show that $mm_{\omega_2}(x) \geq mm_{\omega_1}(x)$:

$$\begin{aligned}
mm_{\omega_2}(x) &= \frac{\log \frac{1}{n}\sum_i e^{\omega_2 x_i}}{\omega_2} \\
&= \frac{\log \frac{1}{n}\sum_i e^{\omega_1 x_i \frac{\omega_2}{\omega_1}}}{\omega_2} \\
&= \frac{\log \frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}}}{\omega_2}.
\end{aligned}$$

We now utilize the convexity of the function $f(z) = z^p$ for $z > 0$ and $p > 1$. For any such convex function, Jensen's inequality holds:

$$\frac{1}{n}\sum_i f(y_i) \geq f(\frac{1}{n}\sum_i y_i).$$

Substituting f with $y_i = e^{\omega_1 x_i}$ and $p = \frac{\omega_2}{\omega_1}$ we get:

$$\frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}} \geq (\frac{1}{n}\sum_i e^{(\omega_1 x_i)})^{(\frac{\omega_2}{\omega_1})}.$$

Using the above inequality, we finally get:

$$\begin{aligned} mm_{\omega_2}(x) &= \frac{\log \frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}}}{\omega_2} \\ &\geq \frac{\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})^{(\frac{\omega_2}{\omega_1})}}{\omega_2} \\ &= \frac{(\frac{\omega_2}{\omega_1})\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})}{\omega_2} \\ &= \frac{\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})}{\omega_1} = mm_{\omega_1}(x), \end{aligned}$$

allowing us to conclude that $mm_\omega(x)$ is a non-decreasing function of ω.

3.5 Mellowmax as Entropy Regularization

In this section I show that using mellowmax in Bellman equation could be thought of as a way penalizing the agent for choosing policies that are highly deterministic.

Recall that the original Bellman equation presented in the background section was:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a)\ \max_{a' \in \mathcal{A}} Q(s',a')\ .$$

This could equivalently be written as:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a)\ \big(\max_{\pi} \sum_{a'} \pi(a' \mid s') Q(s',a')\big)\ .$$

Now suppose that instead of doing $\big(\max_\pi \sum_{a'} \pi(a' \mid s')Q(s',a')\big)$, we desire to penalize the agent for putting all or most of its weight behind just one action. In other words, we desire to have policies that not just attain high values of $\sum_{a'} \pi(a' \mid s')Q(s',a')\big)$, but also are stochastic enough.

One way of achieving this would be add an entropy term $-\sum_a \pi(a \mid s)\log\pi(a \mid s)$. that would be lowest if the agent chooses to put all of its weight behind a single action, and would be at its peak if the agent chooses actions uniformly at random. A middle ground would be two consider both reward maximization and entropy minimization at the same time with a parameter that governs the trade-off:

$$f(Q,\omega) = \max_\pi \sum_a \pi(a \mid s)Q(s,a) - \frac{1}{\omega}\sum_a \pi(a \mid s)\log\pi(a \mid s)\ .$$

We additionally add a third term $\log|\mathcal{A}|$ that does not change the optimal policy, but allows us to get the functional form that we require:

$$\max_\pi \sum_a \pi(a \mid s)Q(s,a) - \frac{1}{\omega}\sum_a \pi(a \mid s)\log\pi(a \mid s) - \frac{1}{\omega}\log|\mathcal{A}|\ .$$

We now have to solve the above optimization problem under the constraint that $\sum_a \pi(a|s) = 1$. To this end, we can utilize the Lagrangian:

$$L(\pi, \lambda, Q) := \sum_a \pi(a \mid s) Q(s,a) - \frac{1}{\omega} \sum_a \pi(a \mid s) \log \pi(a \mid s) - \frac{1}{\omega} \log |\mathcal{A}| - \lambda(\sum_a \pi(a|s) - 1)$$

First note that:

$$\frac{\partial L}{\partial \lambda} = (\sum_a \pi(a|s) - 1) = 0 \implies \sum_a \pi(a|s) = 1 \tag{3.1}$$

Second:

$$\frac{\partial L}{\partial \pi(a|s)} = Q(s,a) - \frac{1}{\omega} \log \pi(a|s) - \frac{1}{\omega} - \lambda = 0$$

This implies that:

$$\pi(a|s) = \frac{e^{\omega Q(s,a)}}{e^{1+\omega\lambda}} \tag{3.2}$$

Combining (3.1) and (3.2), we can write:

$$\frac{\sum e^{\omega Q(s,a)}}{e^{1+\omega\lambda}} = 1 \qquad \implies \qquad e^{1+\omega\lambda} = \sum_a e^{\omega Q(s,a)}$$

And so we get:

$$\pi(a|s) = \frac{e^{\omega Q(s,a)}}{\sum_a e^{\omega Q(s,a)}} \ .$$

Plugging the derived policy, we get:

$$\begin{aligned} f(Q,\omega) &= \frac{\sum_a Q(s,a) e^{\omega Q(s,a)}}{\sum_a e^{\omega Q(s,a)}} - \frac{1}{\omega} \frac{\sum_a e^{\omega Q(s,a)}(\omega Q(s,a) - \log \sum_a e^{\omega Q(s,a)})}{\sum_a e^{\omega Q(s,a)}} - \frac{1}{\omega} \log |\mathcal{A}| \\ &= \frac{\sum_a Q(s,a) e^{\omega Q(s,a)}}{\sum_a e^{\omega Q(s,a)}} - \frac{\omega}{\omega} \frac{\sum_a e^{\omega Q(s,a)} Q(s,a)}{\sum_a e^{\omega Q(s,a)}} + \frac{1}{\omega} \log \sum_a e^{\omega Q(s,a)}) - \frac{1}{\omega} \log |\mathcal{A}| \\ &= \frac{1}{\omega} \log \frac{\sum_a e^{\omega Q(s,a)}}{|\mathcal{A}|} \\ &= \mathrm{mm}_\omega\big(Q(s,\cdot)\big) \end{aligned}$$

Thus we can conclude by solving for the bellman equation

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s' \mid s,a) \, \big(\mathrm{mm}_\omega Q(s',\cdot)\big) \ ,$$

we are in fact solving for the policy that maximize a combination of reward and entropy as governed by the ω parameter.

3.6 Maximum Entropy Mellowmax Policy

As described, mm_ω computes a value for a list of numbers somewhere between its minimum and maximum. However, it is often useful to actually provide a probability distribution over the actions such that (1) a non-zero probability mass is assigned to each action, and (2) the resulting expected value equals the computed value. Such a probability distribution can then be used for action selection in algorithms such as SARSA.

In this section, I address the problem of identifying such a probability distribution as a maximum entropy problem—over all distributions that satisfy the properties above, pick the one that maximizes information entropy [36, 78]. I formally define the maximum entropy mellowmax policy of a state s as:

$$\pi_{\text{mm}}(s) = \operatorname*{argmin}_{\pi} \sum_{a\in\mathcal{A}} \pi(a|s)\log\big(\pi(a|s)\big) \tag{3.3}$$

$$\text{subject to} \begin{cases} \sum_{a\in\mathcal{A}} \pi(a|s)\widehat{Q}(s,a) = \text{mm}_{\omega}(\widehat{Q}(s,.)) \\ \pi(a|s) \geq 0 \\ \sum_{a\in\mathcal{A}} \pi(a|s) = 1\ . \end{cases}$$

Note that this optimization problem is convex and can be solved reliably using any numerical convex optimization library.

One way of finding the solution is to use the method of Lagrange multipliers. Here, the Lagrangian is:

$$L(\pi,\lambda_1,\lambda_2) = \sum_{a\in\mathcal{A}} \pi(a|s)\log\big(\pi(a|s)\big) - \lambda_1\big(\sum_{a\in\mathcal{A}} \pi(a|s) - 1\big) - \lambda_2\Big(\sum_{a\in\mathcal{A}} \pi(a|s)\widehat{Q}(s,a) - \text{mm}_{\omega}\big(\widehat{Q}(s,.)\big)\Big)\ .$$

Taking the partial derivative of the Lagrangian with respect to each $\pi(a|s)$ and setting them to zero, we obtain:

$$\frac{\partial L}{\partial \pi(a|s)} = \log\big(\pi(a|s)\big) + 1 - \lambda_1 - \lambda_2\widehat{Q}(s,a) = 0 \quad \forall\ a\in\mathcal{A}\ .$$

These $|\mathcal{A}|$ equations, together with the two linear constraints in (3.3), form $|\mathcal{A}|+2$ equations to constrain the $|\mathcal{A}|+2$ variables $\pi(a|s)\ \forall a\in\mathcal{A}$ and the two Lagrangian multipliers λ_1 and λ_2.

Solving this system of equations, the probability of taking an action under the maximum entropy mellowmax policy has the form:

$$\pi_{mm}(a|s) = \frac{e^{\beta\widehat{Q}(s,a)}}{\sum_{a\in\mathcal{A}} e^{\beta\widehat{Q}(s,a)}} \quad \forall a\in\mathcal{A}\ ,$$

where β is a value for which:

$$\sum_{a\in\mathcal{A}} e^{\beta\big(\widehat{Q}(s,a)-\text{mm}_{\omega}\widehat{Q}(s,.)\big)}\big(\widehat{Q}(s,a) - \text{mm}_{\omega}\widehat{Q}(s,.)\big) = 0\ .$$

The argument for the existence of a unique root is simple. As $\beta\to\infty$ the term corresponding to the best action dominates, and so, the function is positive. Conversely, as $\beta\to-\infty$ the term corresponding to the action with lowest utility dominates, and so the function is negative. Finally, by taking the derivative with respect to β, it is clear that the function is monotonically increasing, allowing us to conclude that there exists only a single root. Therefore, we can find β easily using any root-finding algorithm. In particular, I use Brent's method [29] available in the Numpy library of Python.

This policy has the same form as Boltzmann softmax, but with a parameter β whose value depends indirectly on ω. This mathematical form arose not from the structure of mm_ω, but from maximizing the entropy. One way to view the use of the mellowmax operator, then, is as a form of Boltzmann policy with a temperature parameter chosen adaptively in each state to ensure that the operator is sufficiently smooth.

Finally, note that the SARSA update under the maximum entropy mellowmax policy could be thought of as a stochastic implementation of the GVI update under the mm_ω operator:

$$\mathbf{E}_{\pi_{mm}}\big[r+\gamma\widehat{Q}(s',a')\big|s,a\big]=\sum_{s'\in\mathcal{S}}\mathcal{R}(s,a,s')+\gamma\mathcal{P}(s,a,s')\underbrace{\sum_{a'\in\mathcal{A}}\pi_{mm}(a'|s')\widehat{Q}(s',a')\big]}_{\mathrm{mm}_\omega\left(\widehat{Q}(s',.)\right)}$$

due to the first constraint of the convex optimization problem (3.3). Because mellowmax is a non-expansion, SARSA with the maximum entropy mellowmax policy is guaranteed to converge to a unique fixed point. Note also that, similar to other variants of SARSA, the algorithm simply bootstraps using the value of the next state while implementing the new policy.

3.7 Experiments in the Tabular Setting

Before presenting experiments, I note that in practice computing mellowmax can yield overflow if the exponentiated values are large. In this case, we can safely shift the values by a constant before exponentiating them due to the following equality:

$$\frac{\log(\frac{1}{n}\sum_{i=1}^{n}e^{\omega x_i})}{\omega}=c+\frac{\log(\frac{1}{n}\sum_{i=1}^{n}e^{\omega(x_i-c)})}{\omega}\ .$$

A value of $c=\max_i x_i$ usually avoids overflow.

3.7.1 Two-state MDP

I repeat the experiment from Figure 3.4 for mellowmax with $\omega = 16.55$ to get a vector field. The result, presented in Figure 3.5, shows a rapid and steady convergence towards the unique fixed point. As a result, GVI under mm_ω can terminate significantly faster than GVI under boltz_β, as illustrated in Figure 3.6.

3.7.2 Random MDPs

The example in Figure 3.1 was contrived. It is interesting to know whether such examples are likely to be encountered naturally. To this end, I constructed 200 MDPs as follows: I sampled $|\mathcal{S}|$ from

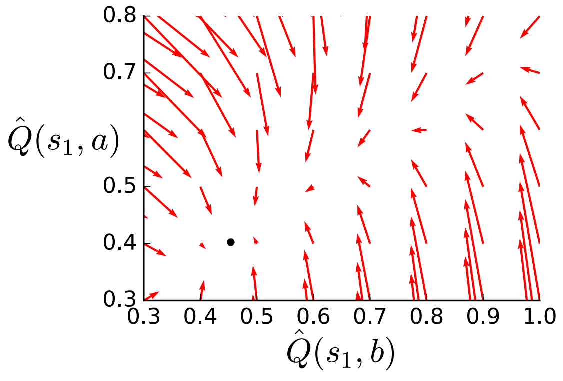


Figure 3.5: GVI updates under $\text{mm}_{\omega=16.55}$. The fixed point is unique, and all updates move quickly toward the fixed point.

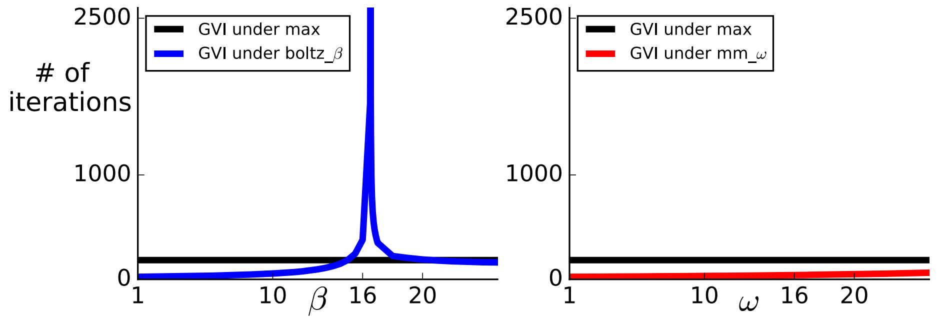


Figure 3.6: Number of iterations before termination of GVI on the example MDP. GVI under mm_ω outperforms the alternatives.

$\{2, 3, ..., 10\}$ and $|\mathcal{A}|$ from $\{2, 3, 4, 5\}$ uniformly at random. I initialized the transition probabilities by sampling uniformly from $[0, .01]$. I then added to each entry, with probability 0.5, Gaussian noise with mean 1 and variance 0.1. I next added, with probability 0.1, Gaussian noise with mean 100 and variance 1. Finally, I normalized the raw values to ensure that I get a transition matrix. I did a similar process for rewards, with the difference that I divided each entry by the maximum entry and multiplied by 0.5 to ensure that $R_{\max} = 0.5$.

I measured the failure rate of GVI under boltz_β and mm_ω by stopping GVI when it did not terminate in 1000 iterations. I also computed the average number of iterations needed before termination. A summary of results is presented in the table below. Mellowmax outperforms Boltzmann based on the three measures provided below.

	MDPs, no terminate	MDPs, > 1 fixed points	average iterations
boltz_β	8 of 200	3 of 200	231.65
mm_ω	**0**	**0**	**201.32**

Table 3.1: A comparison between Mellowmax and Boltzmann in terms of convergence to a unique fixed point.

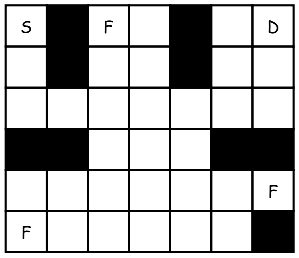


Figure 3.7: Multi-passenger taxi domain. The discount rate γ is 0.99. Reward is +1 for delivering one passenger, +3 for two passengers, and +15 for three passengers. Reward is zero for all the other transitions. Here F, S, and D denote passengers, start state, and destination respectively.

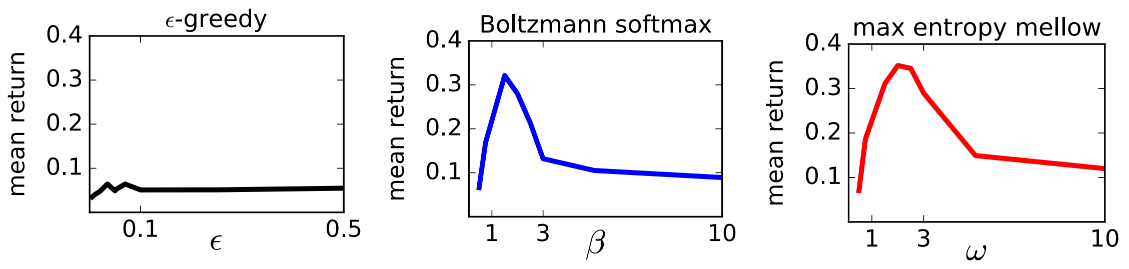


Figure 3.8: Comparison on the multi-passenger taxi domain. Results are shown for different values of ϵ, β, and ω. For each setting, the learning rate is optimized. Results are averaged over 25 independent runs, each consisting of 300000 time steps.

3.7.3 Multi-passenger Taxi Domain

I evaluated SARSA on the multi-passenger taxi domain introduced by Dearden et al. [37]. (See Figure 3.7.)

One challenging aspect of this domain is that it admits many locally optimal policies. Exploration needs to be set carefully to avoid either over-exploring or under-exploring the state space. Note also that Boltzmann softmax performs remarkably well on this domain, outperforming sophisticated Bayesian reinforcement-learning algorithms [37]. As shown in Figure 3.8, SARSA with the epsilon-greedy policy performs poorly. In fact, in our experiment, the algorithm rarely was able to deliver all the passengers. However, SARSA with Boltzmann softmax and SARSA with the maximum entropy mellowmax policy achieved significantly higher average reward. Maximum entropy mellowmax policy is no worse than Boltzmann softmax, here, suggesting that the greater stability does not come at the expense of less effective exploration.

3.8 Experiments in the Function Approximation Setting

Mellowmax is demonstrably better than Boltzmann softmax in contrived examples as I showed above, but can it also be useful in large-scale settings? To answer this question in the affirmative, I now move to the function approximation case where Kim, myself, Littman, and Konidaris [55] used mellowmax as the bootstrapping operator in Deep Q Networks (DQN) [66], hence the name

Parameters	Acrobot	Lunar Lander	Atari
learning rate	10^{-3}	10^{-4}	0.00025
neural network	MLP	MLP	CNN
layers	3	3	4
neurons per layer	300	500	–
update frequency	100	200	10000
number of runs	100	50	5
processing unit	CPU	GPU	GPU

Table 3.2: Experimental details for evaluating DeepMellow for each domain.

DeepMellow. We simply changed the bootstrapping operator of DQN from max to mellowmax.

We first tested DeepMellow in two control domains (Acrobot, Lunar Lander) for which a compact, albeit continuous, representation of the state is presented to the agent. We also tested DeepMellow on two Atari games (Breakout, Seaquest), with a network architecture and hyper parameters akin to that of the original DQN paper [66]. The parameters and neural network architectures for each domain are summarized in Table 3.2.

Target network update frequency is iportant knob in our experiments. In DQN, while the real action-value function is updated every iteration, the target network is only updated every C iterations—we call C the target network update frequency. When $C > 1$, the target network is updated with a delay, while setting $C = 1$ ensures the target network is copied from the real action-value function after every single update. Stated differently, $C = 1$ would obviate the separate target network.

Choice of Temperature Parameter ω

It is important to tune the ω parameter of DeepMellow. If ω is set to an extremely large value, Mellowmax behaves like max, so no benefit over max would be expected. With an ω close to zero, Mellowmax behaves like averaging, a behavior not conducive to reward maximization. To find the optimal ranges of the ω parameter for each domain, we used a grid search method, as we did for other hyperparameters. We empirically found that simpler domains (Acrobot, Lunar Lander) require relatively smaller ω values while large-scale Atari domains require larger values. For Acrobot and Lunar Lander, our parameter search set was $\omega \in \{1, 2, 5, 10\}$. For Breakout and Seaquest, we tested $\omega \in \{100, 250, 1000, 2000, 3000, 5000\}$ and $\omega \in \{10, 20, 30, 40, 50, 100, 200\}$, respectively.

3.8.1 DeepMellow vs DQN without a Target Network

We first compared DeepMellow and DQN in the absence of a target network (or target network update frequency $C = 1$). The control domain results are shown in the Figure 3.9 (left). In Acrobot, DeepMellow achieves more stable learning than DQN—without a target network, the learning curve of DQN goes upward fast, but soon starts fluctuating and fails to improve towards the end. By contrast, DeepMellow (especially with temperature parameter $\omega = 1$) succeeds. Similar results are

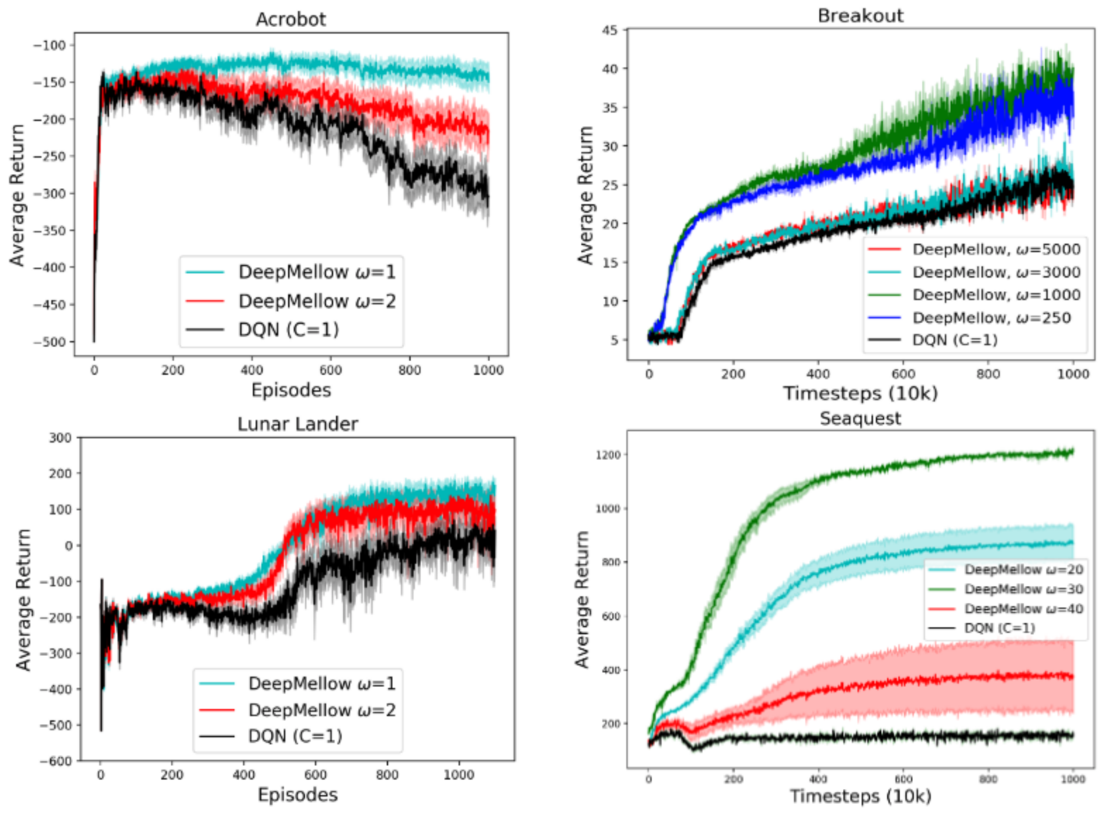


Figure 3.9: The performance of DeepMellow (no target network) and DQN (no target network) in control domains (left) and Atari games (right). DeepMellow outperforms DQN in all domains, in the absence of target network. Note that the best performing temperature ω values vary across domains.

observed in Lunar Lander. Overall, DeepMellow ($\omega \in \{1, 2\}$) achieves more stable learning and higher average returns than DQN by virtue of avoiding the pitfalls of pure maximization and also by using a sufficiently smooth operator.

We also compared the performances of DeepMellow and DQN in two Atari games, Breakout and Seaquest, to ensure that the observed benefits are generalizable to larger environments. We chose these two domains because the effects of having a target network are known to be different in each domain [66]. In Breakout, the performance of DQN does not differ significantly with and without a target network. On the other hand, Seaquest is a domain that shows a significant performance drop when the target network is absent. Thus, these two domains are two contrasting examples for us to see whether DeepMellow obviates target network.

Figure 3.9 (right) shows the performances of DeepMellow and DQN in these games. DeepMellow performed better than DQN without a target network in both Breakout and Seaquest; especially in

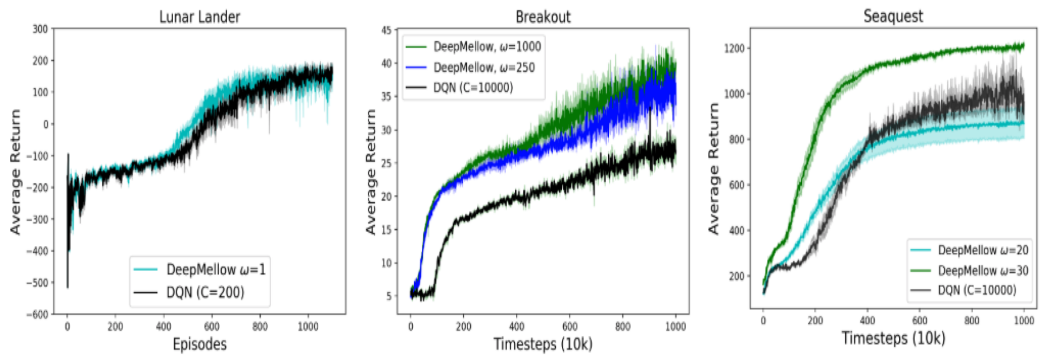


Figure 3.10: Performances of DeepMellow (no target network) and DQN (with a target network). If tuned with an optimal temperature parameter ω value, DeepMellow learns faster than DQN with a target network.

Seaquest, the performance gap was substantial. Also, note that there are intermediate ω values that yield best performances of DeepMellow in each domain.

3.8.2 DeepMellow vs DQN with a Target Network

In the previous section, I reported that DeepMellow outperforms DQN without a target network. The next question that naturally arises is whether DeepMellow without a target network performs even better than DQN *with* a target network. To this end, we compared their performances, focusing on their learning speed.

As shown in Figure 3.10, DeepMellow *does* learn faster than DQN in Lunar Lander, Breakout, and Seaquest domains. In Acrobot (not shown), there was no significant difference, because both algorithms learned very quickly. Together, these results compliment the theoretical benefits of using mellowmax.

3.9 Conclusion

In this chapter, I proposed and evaluated the mellowmax operator as an appealing alternative for a smooth approximation of max in RL. I showed that mellowmax has several desirable properties and that it works favorably in practice, including on large Atari games. Arguably, mellowmax could be used in place of Boltzmann throughout RL research.

Chapter 4

Smoothness in Continuous Control

In this chapter I take a deep dive into solving RL problems with continuous action spaces. As I showed before, a core operation in RL is to perform maximization with respect to the Q function, i.e. $\max_{a\in\mathcal{A}} \widehat{Q}(s,a)$. When $\mathcal{A}$ is discrete and finite, the operation is trivially performed by taking the maximum over a finite set of numbers. In contrast, when $\mathcal{A}$ is continuous we need to rethink the way in which we represent the Q function so that performing the operation becomes tractable. To this end, I introduce deep radial-basis value functions (RBVFs): value functions learned using a deep neural network with a radial-basis function (RBF) output layer equipped with a smoothing parameter. I show that the maximum action-value with respect to a deep RBVF can be found tractably. Moreover, deep RBVFs can represent any true value function owing to their support for universal function approximation. I introduce a value-function-only algorithm that utilizes Deep RBVFs, and with the right degree of smoothness, achieves state-of-the-art performance in continuous-action RL problems.

This chapter is based on a paper written by Neev Parikh, Ronald Parr, George Konidaris, Michael Littman, and myself [10].

4.1 Introduction

Value (Q) function is a core ingredient of RL, and it quantifies the expected return for taking an action a in a state s. Numerous RL algorithms learn an approximation of Q either directly from environmental interactions or indirectly using a learned model (See next chapter). When using function approximation to learn the Q function, the agent has a parameterized function class, and seeks for parameter setting θ for $\widehat{Q}(s, a; \theta)$ that accurately represents the true Q function:

$$\widehat{Q}(s, a; \theta) \approx Q^{\pi}(s, a) \ .$$

A core operation here is finding an optimal action with respect to the learned Q function, specifically $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$. Action selection is the first context in which this operation is relevant: it is clear that successfully performing greedy action selection hinges on solving this maximization problem accurately.

The need to perform this operation arises in another important context as we have seen before, namely when learning the Q function from data. As a concrete example, I showed that Q-learning proceeds with the following update rule:

$$\theta \leftarrow \theta + \alpha \ \big(r_t + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s_{t+1}, a'; \theta) - \widehat{Q}(s_t, a_t; \theta)\big) \ \nabla_{\theta} \widehat{Q}(s_t, a_t; \theta)$$

and so for successfully applying Q-learning, as well as many other RL algorithms, it is imperative that the maximization problem can be solved efficiently and accurately.

A simple approach to performing this maximization in continuous actions is to partition the action space into a finite number of subsets, thus reducing the original continuous problem to a discrete one for which maximization is trivial to do. While this approach can be effective in low-dimensional settings, to uniformly cover the action space, we need a partition whose size grows exponentially with the number of action dimensions. Pazis and Parr attempt to combat this curse of dimensionality via approximate linear programming [76], but scaling their approach to larger domains remains open to the best of my knowledge.

Another line of work has shown the benefits of using function classes that are conducive to efficient action maximization. For example, Gu et al., explored function classes that can capture an arbitrary dependence on the state, but only a quadratic dependence on the action [47]. Given a quadratic action dependence, Gu et al., showed how to compute $\arg\max_{a \in \mathcal{A}} \widehat{Q}(s, a; \theta)$ quickly.

A more general idea is to use input–convex neural networks [4] that restrict $\widehat{Q}(s, a; \theta)$ to convex (or concave) functions with respect to a, so that the maximization problem can be solved efficiently using convex-optimization techniques [28]. We can obtain neural networks that are convex by ensuring that all weights applied to the action input of the network is positive, and that the activation function is also convex and non-decreasing. See Amos et. al, [4] for more details.

Restricting the function class in these ways, however, comes at its own cost, namely that these approaches are unable to support universal function approximation [50], and may lead into inaccurate value functions regardless of the amount of experience provided to the agent. This is significant because our desire is to apply RL algorithms to any problem including to those whose value function is highly complicated.

Together with Parikh, Parr, Konidaris, and Littman [10], we introduced deep radial-basis value functions (RBVFs): Q functions approximated by a standard deep neural network augmented with an RBF output layer. I show that deep RBVFs enable us to efficiently and accurately identify an approximately optimal action without impeding universal function approximation. A function approximator that enjoys both properties was absent in the RL literature prior to our work.

Moreover, I present two sets of experimental results revolving around RBVFs. I first equip DQN, a standard deep RL algorithm originally proposed for discrete actions [66], with a deep RBVF and produce a new continuous-control algorithm called RBF-DQN. I evaluate RBF-DQN to demonstrate its superior performance relative to value-function-only baselines, and to show its competitiveness with state-of-the-art actor-critic RL. I also show that a deep RBVF could serve as the critic in standard actor-critic algorithms such as DDPG [92, 61].

An important piece of a deep RBVF is the smoothing parameter which enables us to adjust the smoothness of RBVFs. I show that often an intermediate degree of smoothness yields best results, and that carefully tuning this parameter is crucial for obtaining state-of-the-art performance in continuous control.

4.2 Deep Radial-Basis Value Functions

Deep Radial-Basis Value Functions (RBVFs) combine the practical advantages of deep networks [44] with the theoretical benefits of radial-basis functions (RBFs) [81]. A deep RBVF is comprised of a number of arbitrary hidden layers, followed by an *RBF output layer*, defined next. The RBF output layer [31] is sometimes used as a standalone single-layer function approximator, and is referred to as a (shallow) RBF network. It is also a core ingredient of the kernel trick in Support Vector Machines [35]. We use an RBF network as the final, or output, layer of a deep network.

For a given input a, the RBF layer $f(a)$ is defined as:

$$f(a) := \sum_{i=1}^{N} g(a - a_i)\ v_i\ , \tag{4.1}$$

where each a_i represents a *centroid* location, v_i is the value of the centroid a_i, N is the number of centroids, and g is an RBF. A commonly used RBF is the negative exponential:

$$g(a - a_i) := e^{-\beta\|a-a_i\|} \ , \tag{4.2}$$

equipped with an inverse smoothing parameter $\beta \geq 0$. Formulation (4.1) could be thought of as an interpolation based on the value and the weights of all centroids, where the weight of each centroid is determined by its proximity to the input. Proximity here is quantified by the RBF g, in this case the negative exponential (4.2).

It is theoretically useful to normalize centroid weights to ensure that they sum to 1 so that f implements a weighted average. This weighted average is sometimes referred to as a normalized RBF layer [69, 33]:

$$f_\beta(a) := \frac{\sum_{i=1}^{N} e^{-\beta\|a-a_i\|} \ v_i}{\sum_{i=1}^{N} e^{-\beta\|a-a_i\|}} \ . \tag{4.3}$$

As the inverse smoothing parameter $\beta \to \infty$, the function implements a winner-take-all case where the value of the function at a given input is determined only by the value of the closest centroid location, nearest-neighbor style. This limiting case is sometimes referred to as a *Voronoi decomposition* [13]. Conversely, f converges to the mean of centroid values regardless of the input a as β gets close to 0; that is, $\forall a \ \lim_{\beta\to 0} f_\beta(a) \ = \ \frac{\sum_{i=1}^{N} v_i}{N}$.

Since an RBF layer is differentiable, it could be used in conjunction with gradient-based optimization techniques and Backprop to learn the centroid locations and their values by optimizing for a loss function.

Note that formulation (4.3) is different than the Boltzmann softmax operator studied in the previous chapter. With a Boltzmann operator the weights of individual actions are determined not by an RBF, but by the action values.

Finally, to represent the Q function for RL, I use the following formulation:

$$\widehat{Q}_\beta(s, a; \theta) := \frac{\sum_{i=1}^{N} e^{-\beta\|a-a_i(s;\theta)\|} \ v_i(s;\theta)}{\sum_{i=1}^{N} e^{-\beta\|a-a_i(s;\theta)\|}} \ . \tag{4.4}$$

From equation (4.4) a deep RBVF learns two mappings: state-dependent centroid locations $a_i(s;\theta)$ and state-dependent centroid values $v_i(s;\theta)$. The role of the output layer is to compute the output of the entire deep RBVF. I illustrate the architecture of a deep RBVF in Figure 4.1. In the experimental section, I demonstrate how to learn parameters θ.

I now show that deep RBVFs have a highly desirable property for value-function-based RL, namely that they enable easy action maximization.

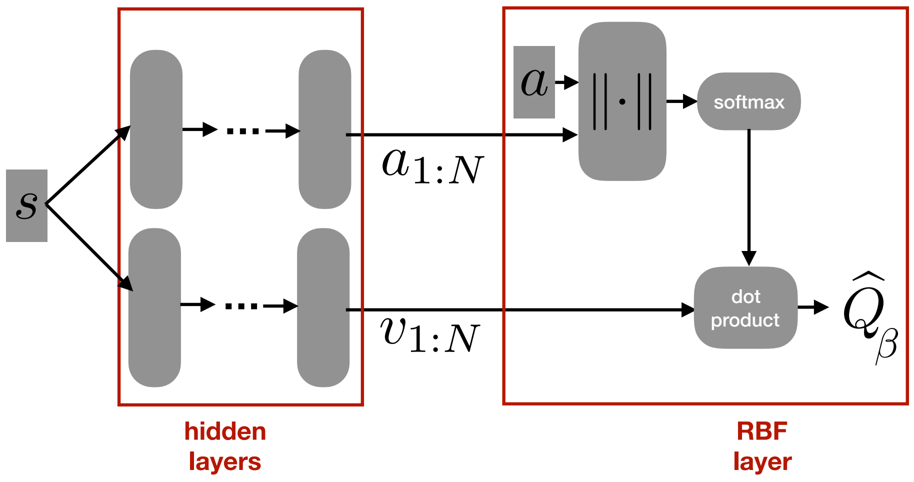


Figure 4.1: Architecture of a deep RBVF, which could be thought of as an RBF output layer added to an otherwise standard deep Q function. All operations of the final RBF layer are differentiable, so the parameters of hidden layers θ, which represent the mappings $a_i(s;\theta)$ and $v_i(s;\theta)$, can be learned using gradient-based optimization techniques.

First, it is easy to find the output of a deep RBVF at each centroid location a_i, that is, to compute $\widehat{Q}_\beta(s, a_i;\theta)$. Note that $\widehat{Q}_\beta(s, a_i;\theta) \neq v_i(s;\theta)$ in general for a finite β, because the other centroids a_j $\forall j \in \{1, .., N\} - i$ may have non-zero weights at a_i. To compute $\widehat{Q}_\beta(s, a_i;\theta)$, we access the centroid location using $a_i(s;\theta)$, then input a_i to get $\widehat{Q}(s, a_i;\theta)$. Once we have $\widehat{Q}(s, a_i;\theta)$ $\forall a_i$, we can trivially find the centroid with highest value: $\max_{i\in[1,N]} \widehat{Q}_\beta(s, a_i;\theta)$.

Recall that our goal is to compute $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s, a;\theta)$, but so far I have shown how to compute $\max_{i\in[1,N]} \widehat{Q}_\beta(s, a_i;\theta)$. I now show that these two quantities are equivalent in one-dimensional action spaces. More importantly, Theorem 4.2.1 shows that with arbitrary number of dimensions, there may be a gap, but that this gap gets exponentially small with increasing the inverse smoothing parameter β.

Theorem 4.2.1. *Let $\widehat{Q}_\beta$ be a normalized negative-exponential RBVF.*

1. *For $\mathcal{A} \in \mathcal{R}$:*
   $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s, a;\theta) = \max_{i\in[1,N]} \widehat{Q}_\beta(s, a_i;\theta)$.
2. *For $\mathcal{A} \in \mathcal{R}^d \quad \forall d > 1$:*
   $\max_{a\in\mathcal{A}}\widehat{Q}_\beta(s, a;\theta) - \max_{i\in[1,N]}\widehat{Q}_\beta(s, a_i;\theta) \leq \mathcal{O}(e^{-\beta})$.

Proof. I begin by proving the first result. For an arbitrary action a, we can write

$$\widehat{Q}_\beta(s, a;\theta) = w_1 v_1(s;\theta) + ... + w_N v_N(s;\theta) \ ,$$

Without loss of generality, we sort centroids so that $\forall i \in [1, N-1]$, $a_i \leq a_{i+1}$. Take two neighboring centroids a_L and a_R and notice that:

$$\forall i < L, \quad \frac{w_L}{w_i} = \frac{e^{-|a-a_L|}}{e^{-|a-a_i|}} = \frac{e^{-a+a_L}}{e^{-a+a_i}} = e^{a_L - a_i} \overset{\text{def}}{=} \frac{1}{c_i} \implies w_i = w_L c_i \ .$$

In the above, we used the fact that all a_i are to the left of a and a_L. Similarly, we can argue that $\forall i > R \quad W_i = W_R c_i$. Intuitively, for actions between a_L and a_R, we will have a constant ratio between the weight of a centroid to the left of a_L, over the weight of a_L itself. The same holds for the centroids to the right of a_R.

In light of the above result, by some changes of variables we can now write:

$$\begin{aligned}
\widehat{Q}_\beta(s,a;\theta) &= w_1 v_1(s;\theta) + ... + w_L v_L(s;\theta) + w_R v_R(s;\theta) + ... + w_K v_K(s;\theta) \\
&= w_L c_1 v_1(s;\theta) + ... + w_L v_L(s;\theta) + w_R v_R(s;\theta) + ... + w_R c_K v_K(s;\theta) \\
&= w_L\big(c_1 v_1(s;\theta) + ... + v_L(s;\theta)\big) + w_R\big(v_R(s;\theta) + ... + c_K v_K(s;\theta)\big) \ .
\end{aligned}$$

Moreover, note that the weights sum to 1: $w_L(c_1 + ... + 1) + w_R(1 + ... + c_K) = 1$, and w_L is at its peak when we choose $a = a_L$ and at its smallest value when we choose $a = a_R$. A converse statement is true about w_R. Moreover, the weights monotonically increase and decrease as we move the input a. We call the endpoints of the range w_{min} and w_{max}. As such, the problem $\max_{a \in [a_L, a_R]} \ \widehat{Q}_\beta(s,a;\theta)$ could be written as this linear program:

$$\begin{aligned}
\max_{w_L, w_R} \quad & w_L\big(c_1 v_1(s;\theta) + ... + v_L(s;\theta)\big) + w_R\big(v_R(s;\theta) + ... + c_K v_K(s;\theta)\big) \\
s.t. \quad & w_L(c_1 + ... + 1) + w_R(1 + ... + c_K) = 1 \\
& w_L, w_R \geq W_{min} \\
& w_L, w_R \leq W_{max}
\end{aligned}$$

A standard result in linear programming is that every linear program has an extreme point that is an optimal solution [28]. Therefore, at least one of the points $(w_L = w_{min}, w_R = w_{max})$ or $(w_L = w_{max}, w_R = w_{min})$ is an optimal solution. It is easy to see that there is a one-to-one mapping between a and W_L, W_R in light of the monotonic property.

As a result, the first point corresponds to the unique value of $a = a_R(s)$, and the second corresponds to unique value of $a = a_L(s)$. Since no point in between two centroids can be bigger than the surrounding centroids, at least one of the centroids is a globally optimal solution in the range $[a_1(s), a_N(s)]$, that is

$$\max_{a \in \mathcal{A}} \widehat{Q}_\beta(s,a;\theta) = \max_{i \in [1,N]} \widehat{Q}_\beta(s,a_i;\theta) \ .$$

To finish the proof, we can show that $\forall a < a_1 \ \widehat{Q}_\beta(s,a;\theta) = \widehat{Q}_\beta(s,a_1;\theta)$. The proof for $\forall a >$

a_N $\widehat{Q}_\beta(s,a;\theta)=\widehat{Q}_\beta(s,a_N;\theta)$ follows similar steps:

$$\begin{aligned}
\forall a < a_1\ \widehat{Q}_\beta(s,a;\theta) &= \frac{\sum_{i=1}^N e^{-\beta|a-a_i|}v_i(s)}{\sum_{i=1}^N e^{-\beta|a-a_i|}} \\
&= \frac{\sum_{i=1}^N e^{-\beta|a_1-c-a_i|}v_i(s)}{\sum_{i=1}^N e^{-\beta|a_1-c-a_i|}} \quad (a = a_1 - c \text{ for some } c>0) \\
&= \frac{\sum_{i=1}^N e^{\beta(a_1-c-a_i)}v_i(s)}{\sum_{i=1}^N e^{\beta(a_1-c-a_i)}} \quad (a_1 - c < a_1 \le a_i) \\
&= \frac{e^{-c}\sum_{i=1}^N e^{\beta(a_1-a_i)}v_i(s)}{e^{-c}\sum_{i=1}^N e^{\beta(a_1-a_i)}} \\
&= \frac{\sum_{i=1}^N e^{\beta(a_1-a_i)}v_i(s)}{\sum_{i=1}^N e^{\beta(a_1-a_i)}} = \widehat{Q}_\beta(s,a_1;\theta)\ ,
\end{aligned}$$

We now prove the second part of the Theorem. First, note that:

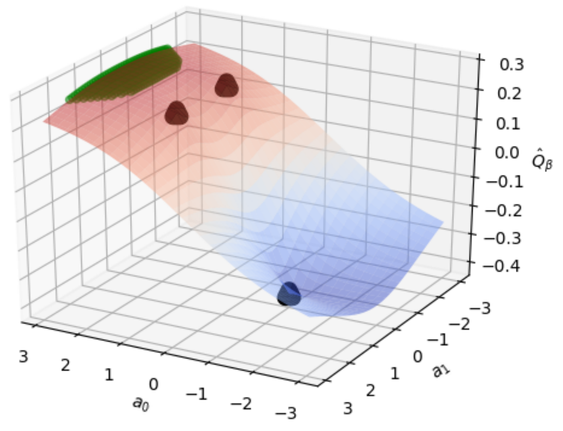
$$\begin{aligned}
\max_a \widehat{Q}_\beta(s,a;\theta) - \max_{i\in[1:N]} \widehat{Q}(s,a_i;\theta) &\le v_{max}(s;\theta) - \max_{i\in[1:N]} \widehat{Q}(s,a_i;\theta) \\
&= v_{max}(s;\theta) - \widehat{Q}_\beta(s,a_{max};\theta)\ .
\end{aligned}$$

Without loss of generality, we assume that the first centroid is the one with the highest v, that is $v_1(s;\theta) = \arg\max_{v_i} v_i(s;\theta)$:

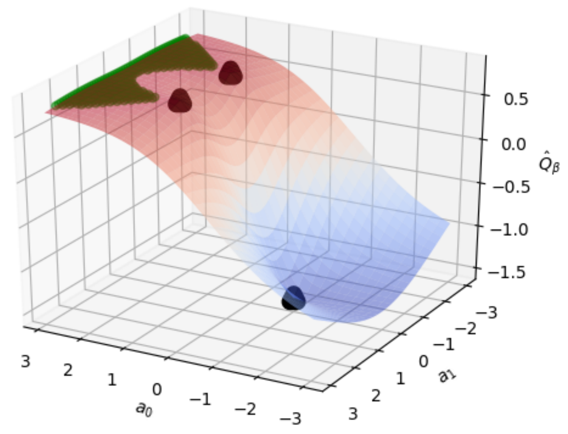
$$\begin{aligned}
v_{max}(s) - \widehat{Q}_\beta(s,a_{max};\theta) &= v_1 - \frac{\sum_{i=1}^N e^{-\beta\|a_1-a_i\|}v_i(s)}{\sum_{i=1}^N e^{-\beta\|a_1-a_i\|}} \\
&= \frac{\sum_{i=1}^N e^{-\beta\|a_1-a_i\|}\big(v_1(s)-v_i(s)\big)}{\sum_{i=1}^N e^{-\beta\|a_1-a_i\|}} \\
&= \frac{\sum_{i=2}^N e^{-\beta\|a_1-a_i\|}\big(v_1(s)-v_i(s)\big)}{1+\sum_{k=2}^K e^{-\beta\|a_1-a_i\|}} \\
&\le \Delta_q \frac{\sum_{i=2}^N e^{-\beta\|a_1-a_i\|}}{1+\sum_{i=2}^N e^{-\beta\|a_1-a_i\|}} \quad \text{(See [96].)} \\
&\le \Delta_q \sum_{i=2}^N \frac{e^{-\beta\|a_1-a_i\|}}{1+e^{-\beta\|a_1-a_i\|}} \\
&= \Delta_q \sum_{i=2}^N \frac{1}{1+e^{\beta\|a_1-a_i\|}} = \mathcal{O}(e^{-\beta}).
\end{aligned}$$

□

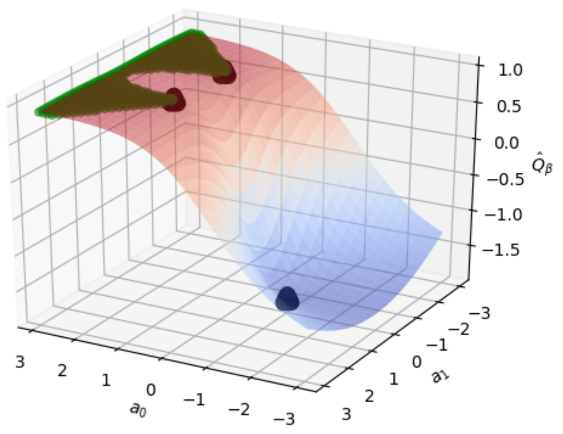
In light of Theorem 4.2.1, to approximate $\max_{a\in\mathcal{A}} \widehat{Q}(s,a;\theta)$ we simply compute $\max_{i\in[1,N]} \widehat{Q}(s,a_i;\theta)$. If the goal is to ensure that the approximation is sufficiently accurate, one can always increase the inverse smoothing parameter to quickly get the desired accuracy. Notice that this result holds for normalized negative-exponential RBVFs, but not necessarily for the unnormalized case or for RBFs other than negative exponential.



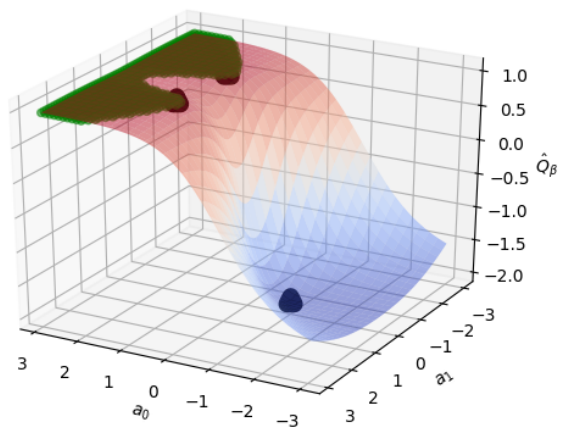
(a) $\beta = 0.25$



(b) $\beta = 1$



(c) $\beta = 1.5$



(d) $\beta = 2$

Figure 4.2: The output of an RBVF with 3 fixed centroid locations and values, but different settings of the inverse smoothing parameter β on a 2-dimensional action space. The regions in dark green highlight the set of actions a for which a is extremely close to the global maximum, $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta)$. Observe the reduction of the gap between $\max_{a\in\mathcal{A}} \widehat{Q}_\beta(s,a;\theta)$ and $\max_{i\in[1,N]} \widehat{Q}_\beta(s,a_i;\theta)$ by increasing β, as guaranteed by Theorem 4.2.1.

Theorem 4.2.2. *Consider any state–action value function $Q^\pi(s,a)$ defined on a closed action space $\mathcal{A}$. Assume that $Q^\pi(s,a)$ is a continuous function. For a fixed state s and for any $\epsilon > 0$, there exists a deep RBF value function $\widehat{Q}_\beta(s,a;\theta)$ and a setting of the inverse smoothing parameter β_0 for which:*

$$\forall a \in \mathcal{A} \quad \forall \beta \geq \beta_0 \quad |Q^\pi(s,a) - \widehat{Q}_\beta(s,a;\theta)| \leq \epsilon \ .$$

Proof. Since Q^π is continuous, we leverage the fact that it is Lipschitz with a Lipschitz constant L:

$$\forall a_0,\ a_1 \quad |f(a_1) - f(a_0)| \leq L\,\|a_1 - a_0\|$$

As such, assuming that $\|a_1 - a_0\| \leq \frac{\epsilon}{4L}$, we have that $|f(a_1) - f(a_0)| \leq \frac{\epsilon}{4}$ Consider a set of centroids $\{c_1, c_2, ..., c_N\}$, define the $\mathit{cell}(j)$ as:

$$\mathit{cell}(j) = \{a \in \mathcal{A}|\ \ \|a - c_j\| = \min_z \|a - c_z\|\} \ ,$$

and the radius $\mathit{Rad}(j, \mathcal{A})$ as:

$$\mathit{Rad}(j, \mathcal{A}) := \sup_{x \in \mathit{cell}(j)} \|x - c_j\| \ \ .$$

Assuming that $\mathcal{A}$ is a closed set, there always exists a set of centroids $\{c_1, c_2, ..., c_N\}$ for which $\mathit{Rad}(c, \mathcal{A}) \leq \frac{\epsilon}{4L}$. Now consider the following functional form:

$$\begin{aligned} \widehat{Q}_\beta(s,a) \quad &:= \quad \sum_{j=1}^{N} Q^\pi(s, c_j) w_j \ , \\ & \qquad \text{where} \quad w_j = \frac{e^{-\beta\|a - c_j\|}}{\sum_{z=1}^{N} e^{-\beta\|a - c_z\|}} \ . \end{aligned}$$

Now suppose a lies in a subset of cells, called the *central* cells $\mathcal{C}$:

$$\mathcal{C} := \{j | a \in \mathit{cell}(j)\} \ ,$$

We define a second *neighboring* set of cells:

$$\mathcal{N} := \{j | \mathit{cell}(j) \cap \big(\cup_{i \in \mathcal{C}} \mathit{cell}(i)\big) \neq \emptyset\} - \mathcal{C} \ ,$$

and a third set of *far* cells:

$$\mathcal{F} := \{j | j \notin \mathcal{C} \ \& \ j \notin \mathcal{N}\} \ ,$$

We now have:

$$\begin{aligned} |Q^\pi(s,a) - \widehat{Q}_\beta(s,a;\theta)| \quad &= \quad |\sum_{j=1}^{N} \big(Q^\pi(s,a) - Q^\pi(s,c_j)\big) w_j| \leq \sum_{j=1}^{N} \big|Q^\pi(s,a) - Q^\pi(s,c_j)\big| w_j \\ &= \quad \sum_{j \in \mathcal{C}} \big|Q^\pi(s,a) - Q^\pi(s,c_j)\big| w_j + \sum_{j \in \mathcal{N}} \big|Q^\pi(s,a) - Q^\pi(s,c_j)\big| w_j \\ & \qquad + \sum_{j \in \mathcal{F}} \big|Q^\pi(s,a) - Q^\pi(s,c_j)\big| w_j \end{aligned}$$

We now bound each of the three sums above. Starting from the first sum, it is easy to see that $\big|Q^\pi(s,a)-Q^\pi(s,c_j)\big| \le \frac{\epsilon}{4}$, simply because $a \in \mathit{cell}(j)$. As for the second sum, since c_j is the centroid of a neighboring cell, using a central cell i, we can write:

$$\|a-c_j\| = \|a-c_i+c_i-c_j\| \le \|a-c_i\| + \|c_i-c_j\| \le \frac{\epsilon}{4L}+\frac{\epsilon}{4L}=\frac{\epsilon}{2L}\ ,$$

and so in this case $\big|Q^\pi(s,a)-\widehat{Q}_\beta(s,c_j)\big| \le \frac{\epsilon}{2}$. In the third case with the set of far cells $\mathcal{F}$, observe that for a far cell j and a central cell i we have:

$$\frac{w_j}{w_i}=\frac{e^{-\beta\|a-c_j\|}}{e^{-\beta\|a-c_i\|}} \to w_j = w_i e^{-\beta(\|a-c_j\|-\|a-c_i\|)} \le w_i e^{-\beta\mu} \le e^{-\beta\mu},$$

For some $\mu > 0$. In the above, we used the fact that $\|a-c_j\|-\|a-c_i\| > 0$ is always true. Then:

$$\begin{aligned}
& |Q^\pi(s,a)-\widehat{Q}_\beta(s,a)| \\
= & \sum_{j\in\mathcal{C}} \underbrace{\big|Q^\pi(s,a)-Q^\pi(s,c_j)\big|}_{\le\frac{\epsilon}{4}} \underbrace{w_j}_{\le 1} + \sum_{j\in\mathcal{N}} \underbrace{\big|Q^\pi(s,a)-Q^\pi(s,c_j)\big|}_{\le\frac{\epsilon}{2}} \underbrace{w_j}_{1} \\
& + \sum_{j\in\mathcal{F}} \big|Q^\pi(s,a)-Q^\pi(s,c_j)\big| \underbrace{w_j}_{e^{-\beta\mu}} \\
\le & \frac{\epsilon}{4}+\frac{\epsilon}{2}+\sum_{j\in\mathcal{F}}\big|Q^\pi(s,a)-Q^\pi(s,c_j)\big|e^{-\beta\mu} \\
\le & \frac{\epsilon}{4}+\frac{\epsilon}{2}+2N\sup_a|Q^\pi(s,a)|e^{-\beta\mu}
\end{aligned}$$

In order to have $2N\sup_a|Q^\pi(s,a)|e^{-\beta\mu} \le \frac{\epsilon}{4}$, it suffices to have $\beta \ge \frac{-1}{\mu}\log(\frac{\epsilon}{8N\sup_a|Q^\pi(s,a)|}) := \beta_0$. To conclude the proof:

$$|Q^\pi(s,a)-\widehat{Q}_\beta(s,a;\theta)| \le \epsilon \quad \forall\ \beta \ge \beta_0\ .$$

□

Collectively, Theorems 4.2.1 and 4.2.2 guarantee that deep RBVFs ensure accurate and efficient action maximization without impeding universal function approximation. This combination of properties stands in contrast with prior work that used function classes that enable easy action maximization but lack UFA [47, 4], as well as prior work that preserved the UFA property but did not guarantee arbitrarily high accuracy when performing the maximization step [62, 88].

In terms of scalability, note that the RBVF formulation scales naturally owing to its freedom to determine centroids that best minimize the loss function. As a thought experiment, suppose that some region of the action space has a high value, so an agent with greedy action selection frequently chooses actions from that region. A deep RBVF would then move more centroids to the region, because the region heavily contributes to the loss function. Since the centroids are state-dependent, the network can learn to move the centroids to more rewarding regions on a per-state basis. It is unnecessary to initialize centorid locations carefully, or to uniformly cover the action space *a priori*. In this sense, learning RBVFs could be thought of as a form of adaptive and soft discretization learned by gradient descent. Adaptive discretization techniques have proven fruitful in terms of sample-complexity guarantees in bandit problems [57], as well as in RL [94, 112, 39].

4.3 RBVFs for Regression

To demonstrate the operation of an RBF network in the clearest setting, I present experiments on a single-input continuous optimization problem where the agent lacks access to the true reward function but can sample input–output pairs $\langle a, r \rangle$. This setting is akin to the action maximization step in stateless RL.

I use the reward function: $r(a) = \|a\|_2 \frac{\sin(a_0)+\sin(a_1)}{2}$. Figure 4.3 (left) shows the surface of this function. It is clearly non-convex and includes several local maxima (and minima). We are interested in two cases, first the problem where the goal is to find $\max_{a \in \mathcal{A}} r(a)$, and the converse problem where we desire to find $\min_{a \in \mathcal{A}} r(a)$.

Here, my focus is not to find the most effective exploration policy, but to evaluate different approaches based on their effectiveness for representing and optimizing a learned reward function $\widehat{r}(a;\theta)$. I therefore adopt the same random action-selection strategy for all approaches, in the interest of fairness: we sampled 500 actions uniformly randomly from $[-3, 3]^2$ to obtain a dataset for training. When learning ended, I computed the action that maximized (or minimized) the learned $\widehat{r}(a;\theta)$. Details of the function classes used in each case, as well as how to perform $\max_{a \in \mathcal{A}} \widehat{r}(a;\theta)$ and $\min_{a \in \mathcal{A}} \widehat{r}(a;\theta)$, are presented below:

Discretization: For our first baseline, we discretized each action dimension to 7 bins, resulting in 49 bins that uniformly covered the two dimensions of the input space. The choice of the number of bins per dimension reflected the need to use roughly the same number of bins for discretization, and the number of centroids in an RBVF. For each bin, we averaged the rewards over $\langle a, r \rangle$ pairs for which the sampled action a belonged to that bin. Once we had a learned $\widehat{r}(a;\theta)$, which in this case was just a 7×7 table, we performed $\max_a \widehat{r}(a;\theta)$ and $\min_a \widehat{r}(a;\theta)$ by a simple table lookup.

Input-Convex Neural Network: Our second baseline used the input-convex neural network architecture [4], where the neural network is constrained so that the learned reward function $\widehat{r}(a;\theta)$ is convex. Learning was performed by RMSProp optimization, with mean-squared loss. Once $\widehat{r}(a;\theta)$ was learned, we used gradient ascent for finding the maximum, and gradient descent for finding the minimum. Note that this input-convex approach subsumes the excluded quadratic case proposed by Gu et al. [47], because quadratic functions are just a special case of convex functions, but the converse in not necessarily true [28].

Wire Fitting: Our next baseline was the wire-fitting method proposed by Baird and Klopf [16]. Their function approximator operates similarly to RBVFs in that it also learns a set of centroids. Again, we used the RMSprop optimizer and mean-squared loss, and finally returned the centroids with lowest (or highest) values according to the learned $\widehat{r}(a;\theta)$.

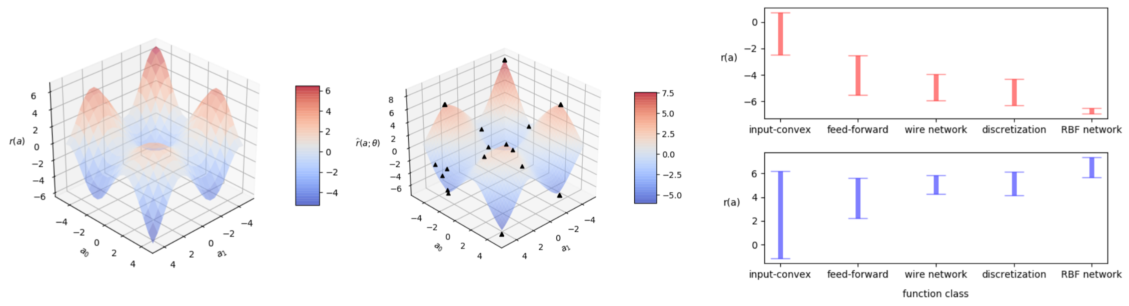


Figure 4.3: Left: True reward function. Center: The reward learned by the RBF reward network. Black dots represent the centroids. Right: Means (averaged over 30 runs) and standard errors on the continuous optimization task. Top: Results for action minimization. (Lower is better.) Bottom: Results for action maximization. (Higher is better.)

Feed Forward: As the last baseline, we used a standard feed-forward neural network architecture with two hidden layers. It is well–known that this function class is capable of UFA [50] and so can accurately learn the reward function in principle. However, once learning ends, we face a non–convex optimization problem for action maximization (or minimization) $\max_a \widehat{r}(a;\theta)$. We simply initialized gradient descent (ascent) to a point chosen uniformly randomly, and followed the corresponding direction until no further progress was made.

To learn a radial-basis reward function, I used $N = 50$ centroids and $\beta = 0.5$. I used RMSprop and mean-squared loss minimization. Recall that Theorem 4.2.1 showed that with an RBVF the following approximations are well-justified: $\max_{a\in\mathcal{A}} \widehat{r}(a) \approx \max_{i\in[1,50]} \widehat{r}(a_i)$ and $\min_{a\in\mathcal{A}} \widehat{r}(a) \approx \min_{i\in[1,50]} \widehat{r}(a_i)$. As such, when the learning of $\widehat{r}(a;\theta)$ ends, we output the centroid values with highest and lowest reward.

For each individual case, I ran the corresponding experimental pipeline for 30 different random seeds. The solution found by each learner was fed to the true reward function to evaluate the quality of the found solution.

I report the average reward achieved by each function class in Figure 4.3 (right). The RBVF learner outperforms all baselines on both the maximization and the minimization problem. I further show the learned RBVF in a run (Figure 4.3 center), which is highly accurate.

4.4 RBVFs for Value-Function-Only Deep RL

I now use deep RBVFs for solving continuous-action RL problems. To this end, we learn a deep RBVF using a learning algorithm akin to that of DQN [66], but extended to the continuous-action

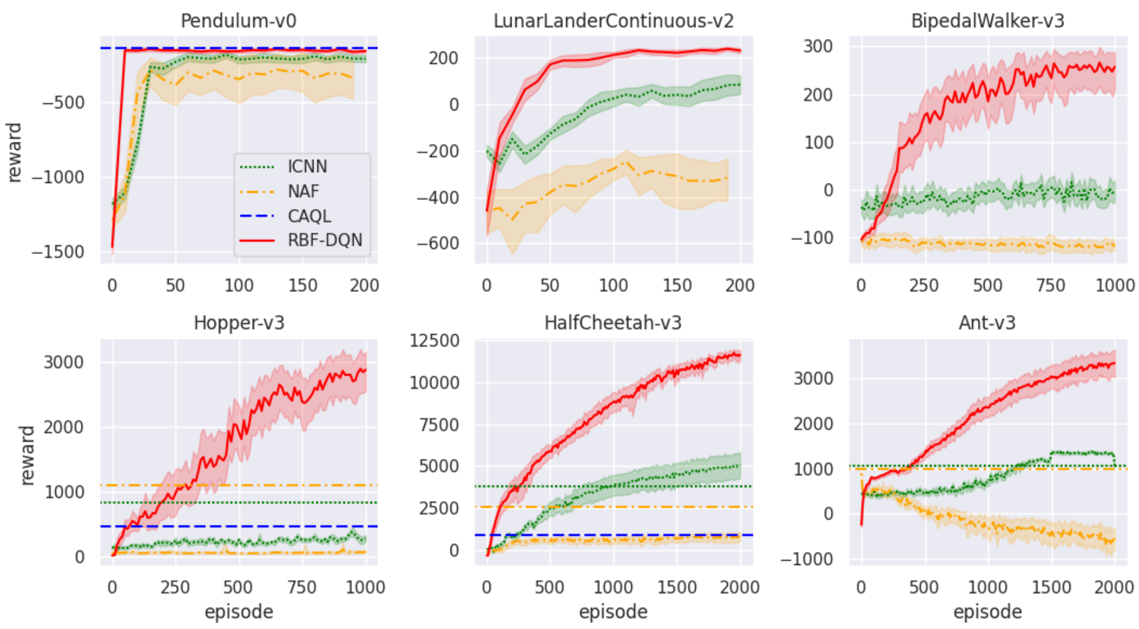


Figure 4.4: A comparison between RBF-DQN and value-function-only deep RL baselines on 6 Open AI Gym environments. For ICNN and NAF, Ihave included two results: Dashed lines indicate the level of performance reported by [4], and the learning curves show the performance of the baselines obtained by running publicly-available implementations. All runs for this figure and all following figures are averaged over 20 trials with different random seeds.

case. DQN uses the following loss function for learning the value function:

$$L(\theta) := \mathbf{E}_{s,a,r,s'}\Big[\Big(r + \gamma \max_{a' \in \mathcal{A}} \widehat{Q}(s', a'; \theta^-) - \widehat{Q}(s, a; \theta)\Big)^2\Big].$$

DQN adds tuples of experience $\langle s, a, r, s' \rangle$ to a buffer, and later samples a minibatch of tuples to compute $\nabla_\theta L(\theta)$. DQN maintains a second network parameterized by weights θ^-. This second network, denoted $\widehat{Q}(\cdot, \cdot, \theta^-)$ and referred to as the *target network*, is periodically synchronized with the online network $\widehat{Q}(\cdot, \cdot, \theta)$. RBF-DQN uses the same loss function, but modifies the function class of DQN. Concretely, DQN learns a deep network with one output per action, exploiting the discrete and finite nature of the action space. By contrast, RBF-DQN takes a state and an action vector, and outputs a single scalar using a deep RBVF. The pseudo-code for RBF-DQN is presented in Algorithm 1.

I now evaluate RBF-DQN against state-of-the-art value-function-only deep RL baselines. Iunderstand NAF [47] and ICNN [4] as two of the best continuous-action extensions of DQN in the RL literature, so Iused them as our baselines. For RBF-DQN, as well as each of the two baselines, we performed 1000 updates per episodes when each episode ends. The authors of ICNN have released an official code base [1], which we used to obtain the ICNN learning curves. To the best of our knowledge, [47] did not release code, but we have used a public implementation of NAF [2]. Compared to the

[1]github.com/locuslab/icnn

[2]github.com/ikostrikov/pytorch-ddpg-naf

Algorithm 4 Pseudo-code for RBF-DQN

```
Initialize deep RBVF with $N, \beta, \theta$
Initialize replay buffer $\mathcal{B}, \epsilon, \gamma, \alpha, \alpha^-, \theta^-$
for $E$ episodes do
    Initialize $s$
    while not done do
        $a \leftarrow \epsilon\text{-greedy}\big(\widehat{Q}_\beta(s, \cdot; \theta), \epsilon\big)$
        $s', r, \text{done} \leftarrow \text{env.step}(s, a)$
        add $\langle s, a, r, s', \text{done}\rangle$ to $\mathcal{B}$; $s \leftarrow s'$
    end while
    for $M$ minibatches sampled from $\mathcal{B}$ do
        for $\langle s, a, r, s', \text{done}\rangle$ in minibatch do
            $\Delta = \big(r - \widehat{Q}_\beta(s, a; \theta)\big)\nabla_\theta \widehat{Q}(s, a; \theta)$
            if not done then
                get centroids $a_i(s'; \theta^-), i \in [1, N]$
                $\Delta \mathrel{+}= \gamma \max_i \widehat{Q}_\beta\big(s', a_i\big(s'; \theta^-\big); \theta^-\big)$
            end if
            $\theta \leftarrow \theta + \alpha\Delta \cdot \ \nabla_\theta \widehat{Q}_\beta(s, a; \theta)$
        end for
        $\theta^- \leftarrow (1 - \alpha^-)\theta^- + \alpha^-\theta$
    end for
end for
```

reported results in the ICNN paper, we were able to roughly achieve the same performance for NAF and ICNN. However, for completeness, I show, via dashed horizontal lines, levels of performance for NAF and ICNN reported by the ICNN paper whenever one was present.

Another relevant baseline is CAQL [88], for which we could neither find the authors' code nor a public implementation. Since the original CAQL paper used environments with constrained action spaces, we only compare with the best reported results for Hopper-v3, HalfCheetah-v3, and Pendulum-v0, as these environments were used with the full action space.

Table 4.5 lists the common hyper-parameters used for deep RBVFs in RBF-DQN, and Table 4.6 lists domain-dependent hyper parameters.

RBF-DQN is clearly outperforming the two baselines, even when considering the results reported by the ICNN authors [4].

In light of the above results, I can claim that RBF-DQN is demonstrably a state-of-the-art value-function-only deep RL algorithm, but how does RBF-DQN compare to state-of-the-art actor-critic deep RL? To answer this question, we compared RBF-DQN to state-of-the-art actor critic deep RL, namely DDPG [92, 61], TD3 [43], and SAC [48]. For TD3 and DDPG, we used the official code released by [43][3], which is also used by numerous other papers. For SAC, we used a public

[3]github.com/sfujim/TD3

hyper-parameter	value
number of hidden layers for centroid values	3
number of hidden layers for centroid locations	1
number of nodes in all hidden layers	512
target network learning rate (exponential moving average)	0.005
replay buffer size	5×10^5
discount-rate γ	0.99
size of mini-batch	256
number of centroids N	100
optimizer	RMSProp
number of updates per episode	10^3

Figure 4.5: Common hyper-parameters used for deep RBVFs in RBF-DQN and RBF-DDPG.

domain	step size (locations)	step size (values)	smoothing
Pendulum-v0	25×10^{-5}	25×10^{-6}	1
LunarLanderContinuous-v2	25×10^{-5}	25×10^{-6}	2
BipedalWalker-v3	10^{-5}	5×10^{-6}	2
Hopper-v3	5×10^{-5}	10^{-5}	1.5
HalfCheetah-v3	10^{-5}	75×10^{-6}	.25
Ant-v3	10^{-5}	5×10^{-6}	.1

Figure 4.6: Tuned RBF-DQN hyper-parameters for each domain.

implementation.[4] In Figure 4.7, we show, not just the learning curves we obtained by running the implementations, but also results reported in the TD3 and SAC papers.

I note that the authors of TD3, DDPG, and SAC report their results in steps. In our experiments, we report results in episodes and have modified all implementations to have 1000 gradient updates per episode, while continuing to record steps. We plot horizontal lines at the level of reward obtained by the authors (and reported in their papers) at the average steps reached by the publicly available implementations when run for the specified number of episodes. For example, in Ant-v3, SAC reaches 1,076,890 steps on average when run for 2000 episodes. Therefore, we estimate the reward from the authors' graph at around 1.07×10^6 steps to be roughly 4000.

From Figure 4.7, RBF-DQN is performing better than or is competitive with state-of-the-art actor critic deep RL baselines on all domains. RBF-DQN can compete with these baselines despite the fact that it only uses 2 neural networks (an online value function and a target value function), while SAC and TD3, for instance, use 5 and 6 networks, respectively.

In our experiments, TD3 and SAC perform 1000 gradient updates on two critics and one actor per each episode, while RBF-DQN only updates a deep RBVF 1000 times. Therefore, TD3 and SAC

[4]github.com/pranz24/pytorch-soft-actor-critic

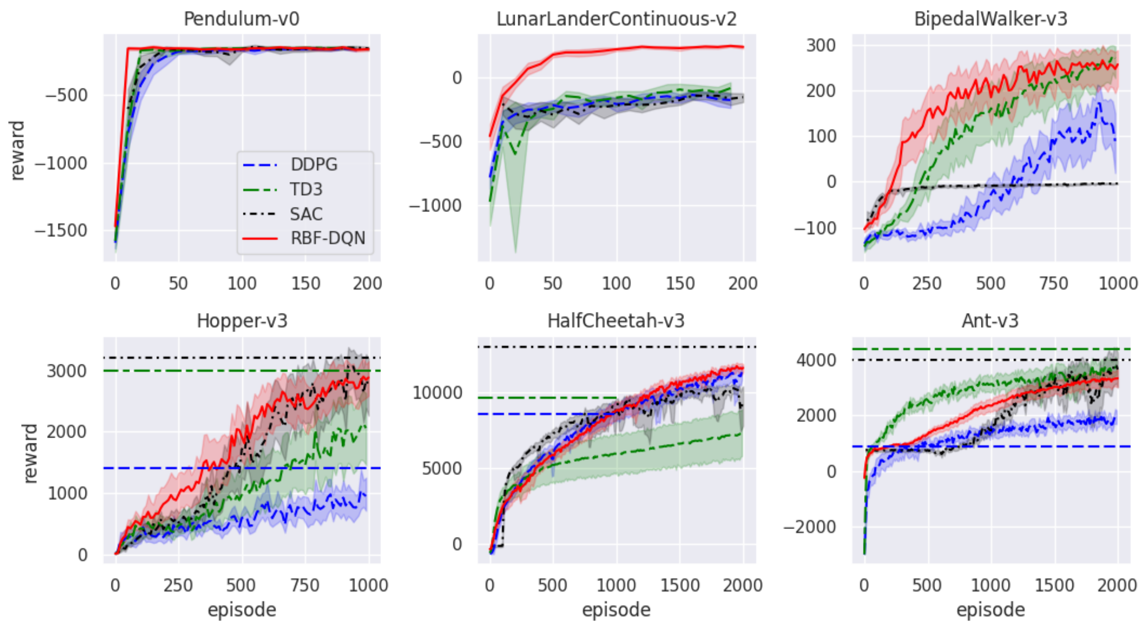


Figure 4.7: A comparison between RBF-DQN and state-of-the-art actor-critic deep RL.

are also performing 3 times as many updates as RBF-DQN. Lastly, some of the ideas leveraged by TD3 and SAC, such as value clipping in TD3 and SAC, and entropy regularization in SAC, can be integrated into RBF-DQN. I believe these combinations are promising, and leave the investigation of these combinations for future work.

4.5 RBVFs for Actor-Critic Deep RL

So far, I have applied RBVFs to value-function-only RL, but can RBVFs be useful for other RL algorithms? To answer this question affirmatively, I now use RBVFs for actor-critic deep RL, in particular as the critic in the DDPG algorithm [92, 61].

Recall that, in contrast with value-function-only RL, actor-critic algorithms learn two separate networks: a value function (or the critic), and a policy (or the actor) that is mainly used for action selection. [92] introduces the deterministic policy-gradient actor critic, but makes the observation that, in continuous control, computing the greedy action with respect to the learned critic is not tractable (see their Subsection 3.1), leading them to instead use a SARSA update for the critic. I showed that a deep RBVF approximately solves this maximization problem tractably. Leveraging this insight, we can modify the DDPG algorithm to use an RBVF critic. Given a tuple $\langle s, a, r, s' \rangle$, we can learn, via the critic, the value function with a Q-learning or SARSA update:

$$\begin{aligned} \theta \leftarrow \theta + \alpha \ \delta_{\text{RBF-DDPG}} \ \nabla_\theta \widehat{Q}(s, a; \theta) \ , \\ \delta_{\text{Q-learning}} := r + \gamma \max_i \widehat{Q}(s', a_i(s'); \theta) - \widehat{Q}(s, a; \theta) \\ \delta_{\text{SARSA}} := r + \gamma \widehat{Q}(s', \pi(s'; \omega); \theta) - \widehat{Q}(s, a; \theta) \ . \end{aligned} \tag{4.5}$$

The original DDPG algorithm used the SARSA update, obviating the action-maximization step.

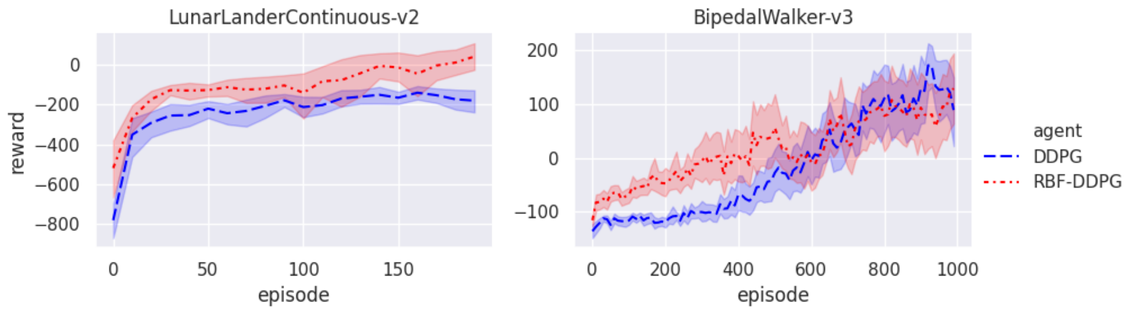


Figure 4.8: A comparison between DDPG and RBF-DDPG.

Using a deep RBVF, RBF-DDPG performs the Q-learning update shown above to update the critic. RBF-DDPG is otherwise analogous to DDPG.

I compare RBF-DDPG and DDPG in Figure 4.8. It is clear that the use of RBVFs benefits DDPG. Although preliminary evaluations of RBF-DDPG do not exceed state-of-the-art performance, further investigation is required to see if the addition of other algorithmic ideas, such as those presented in [43] and [48], can further improve the performance of RBF-DDPG.

4.6 Conclusion

I proposed, analyzed, and exhibited the strengths of deep RBF value functions in continuous control. These value functions facilitate easy action maximization, support universal function approximation, and scale to large continuous action spaces. Deep RBF value functions are thus an appealing choice for value function approximation in continuous control. Controlling the inverse smoothing parameter of deep RBVFs played a critical role in achieving state-of-the-art performance on continuous-action problems.

Chapter 5

Smoothness in Model-based Reinforcement Learning

When is a learned model effective for long-horizon planning? In this chapter I answer this question by examining the impact of learning Lipschitz continuous models. I make the case for a new loss function for model-based RL based on the Wasserstein metric. By formalizing a good one-step model as a Lipschitz model with bounded one-step Wasserstein error, I provide a novel bound on multi-step prediction error of Lipschitz models, as well as on value-estimation error of the one-step model. I conclude with empirical results that show the benefits of controlling the Lipschitz constant of transition models when they are represented using neural networks.

This chapter is based on three papers: a paper written by Dipendra Misra, Michael Littman, and myself [9], a second paper written by Evan Cater, Dipendra Misra, Michael Littman, and myself [6], and a third paper written by Seungchan Kim, Dipendra Misra, Michael Littman and myself [8].

5.1 Introduction

The model-based approach to reinforcement learning (RL) focuses on predicting the dynamics of the environment to plan and make high-quality decisions [52, 102, 12]. Although the behavior of model-based algorithms in tabular environments is well understood and can be effective [102], scaling up to the approximate setting can cause instabilities. Even small model errors can be magnified by the planning process resulting in poor performance [107].

In this chapter, I study model-based RL through the lens of Lipschitz continuity. I show that the ability of a model to make accurate multi-step predictions hinges on not just the model's one-step accuracy, but also the magnitude of the Lipschitz constant (smoothness) of the model. I further show that the dependence on the Lipschitz constant carries over to the value-prediction problem, ultimately influencing the quality of the policy found by planning.

I consider a setting with continuous state spaces and stochastic transitions where I quantify the distance between distributions using the Wasserstein metric. I introduce a novel characterization of models, referred to as a Lipschitz model class, that represents stochastic dynamics using a set of component deterministic functions. This allows us to study any stochastic dynamic using the Lipschitz continuity of its component deterministic functions. To learn a Lipschitz model class in continuous state spaces, I provide an Expectation-Maximization algorithm [38].

One promising direction for mitigating the effects of inaccurate models is the idea of limiting the complexity of the learned models or reducing the horizon of planning [51]. Doing so can sometimes make models more useful, much as regularization in supervised learning can improve generalization performance [110]. I examine a type of regularization that comes from controlling the Lipschitz constant of models. This regularization technique can be applied efficiently, as I will show, when we represent the transition model by neural networks.

Finally, I move beyond one-step models, and introduce a novel multi-step model that can mitigate the compounding error problem in long-horizon planning by removing one source of error. I show that multi-step models can provide more accurate multi-step predictions, and ultimately yield more effective planning in model-based RL.

5.2 Lipschitz Model Class

We introduce a novel representation of stochastic MDP transitions in terms of a distribution over a set of deterministic components.

Definition 5.2.1. *Given a metric state space $(\mathcal{S}, d_{\mathcal{S}})$ and an action space $\mathcal{A}$, we define F_g as a collection of functions: $F_g = \{f : \mathcal{S} \to \mathcal{S}\}$ distributed according to $g(f \mid a)$ where $a \in \mathcal{A}$. We say*

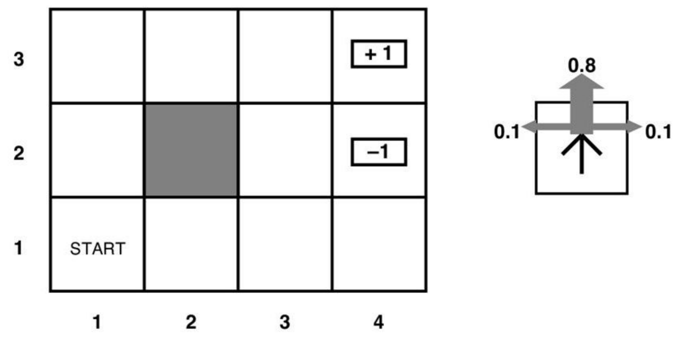


Figure 5.1: An example of a Lipschitz model class in a gridworld environment [87]. The dynamics are such that any action choice results in an attempted transition in the corresponding direction with probability 0.8 and in the neighboring directions with probabilities 0.1 and 0.1. We can define $F_g = \{f^{up}, f^{right}, f^{down}, f^{left}\}$ where each f outputs a deterministic next position in the grid (factoring in obstacles). For $a = up$, we have: $g(f^{up} \mid a = up) = 0.8,\ g(f^{right} \mid a = up) = g(f^{left} \mid a = up) = 0.1$, and $g(f^{down} \mid a = up) = 0$. Defining distances between states as their Manhattan distance in the grid, then $\forall f \sup_{s_1,s_2} \big(d(f(s_1), f(s_2))/d(s_1, s_2)\big) = 2$, and so $K_F = 2$. So, the four functions and g comprise a Lipschitz model class.

that F_g *is* a Lipschitz model class *if*

$$K_F := \sup_{f \in F_g} \ K_{d_\mathcal{S}, d_\mathcal{S}}(f) \ ,$$

is finite.

Our definition captures a subset of stochastic transitions, namely ones that can be represented as a state-independent distribution over deterministic transitions. An example is provided in Figure 5.1.

Associated with a Lipschitz model class is a transition function given by:

$$\widehat{T}(s' \mid s, a) = \sum_f \mathbb{1}\big(f(s) = s'\big) \ g(f \mid a) \ .$$

Given a state distribution $\mu(s)$, I also define a generalized notion of transition function $\widehat{T}_\mathcal{G}(\cdot \mid \mu, a)$ given by:

$$\widehat{T}_\mathcal{G}(s' \mid \mu, a) = \int_s \underbrace{\sum_f \mathbb{1}\big(f(s) = s'\big) \ g(f \mid a)}_{\widehat{T}(s' \mid s, a)} \mu(s) ds \ .$$

We are primarily interested in $K^\mathcal{A}_{d,d}(\widehat{T}_\mathcal{G})$, the Lipschitz constant of $\widehat{T}_\mathcal{G}$. However, since $\widehat{T}_\mathcal{G}$ takes as input a probability distribution and also outputs a probability distribution, we require a notion of distance between two distributions. This notion is quantified using Wasserstein and is justified in the next section.

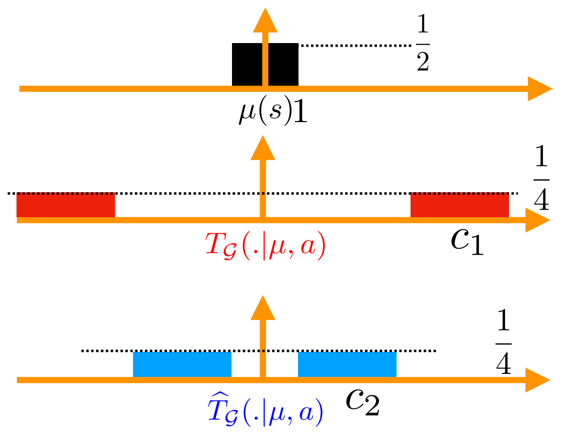

$\frac{1}{2}$
$\mu(s)1$
$\frac{1}{4}$
$T_{\mathcal{G}}(.|\mu,a)$
c_1
$\frac{1}{4}$
$\widehat{T}_{\mathcal{G}}(.|\mu,a)$
c_2

Figure 5.2: A state distribution $\mu(s)$ (top), a stochastic environment that randomly adds or subtracts c_1 (middle), and an approximate transition model that randomly adds or subtracts a second scalar c_2 (bottom).

5.3 On the Choice of Probability Metric

I consider the stochastic model-based setting and show through an example that the Wasserstein metric is a reasonable choice compared to other common options.

Consider a uniform distribution over states $\mu(s)$ as shown in black in Figure 5.2 (top). Take a transition function $T_{\mathcal{G}}$ in the environment that, given an action a, uniformly randomly adds or subtracts a scalar c_1. The distribution of states after one transition is shown in red in Figure 5.2 (middle). Now, consider a transition model $\widehat{T}_{\mathcal{G}}$ that approximates $T_{\mathcal{G}}$ by uniformly randomly adding or subtracting the scalar c_2. The distribution over states after one transition using this imperfect model is shown in blue in Figure 5.2 (bottom). We desire a metric that captures the similarity between the output of the two transition functions. I first consider Kullback-Leibler (KL) divergence and observe that:

$$\begin{aligned}&KL\big(T_{\mathcal{G}}(\cdot \mid \mu,a),\widehat{T}_{\mathcal{G}}(\cdot \mid \mu,a)\big)\\ &:= \int T_{\mathcal{G}}(s' \mid \mu,a)\log\frac{T_{\mathcal{G}}(s' \mid \mu,a)}{\widehat{T}_{\mathcal{G}}(s' \mid \mu,a)}ds' = \infty\ ,\end{aligned}$$

unless the two constants are exactly the same.

The next possible choice is Total Variation (TV) defined as:

$$\begin{aligned}&TV\big(T_{\mathcal{G}}(\cdot \mid \mu,a),\widehat{T}_{\mathcal{G}}(\cdot \mid \mu,a)\big)\\ &:= \frac{1}{2}\int \big|T_{\mathcal{G}}(s' \mid \mu,a)-\widehat{T}_{\mathcal{G}}(s' \mid \mu,a)\big|ds' = 1\ ,\end{aligned}$$

if the two distributions have disjoint supports regardless of how far the supports are from each other.

In contrast, Wasserstein is sensitive to how far the constants are as:

$$W\big(T_{\mathcal{G}}(\cdot \mid \mu,a),\widehat{T}_{\mathcal{G}}(\cdot \mid \mu,a)\big) = |c_1-c_2|\ .$$

It is clear that, of the three, Wasserstein corresponds best to the intuitive sense of how closely $T_{\mathcal{G}}$ approximates $\widehat{T}_{\mathcal{G}}$. This is particularly important in high-dimensional spaces where the true distribution is known to usually lie in low-dimensional manifolds [72].

In the next section, I present a theoretical argument for the choice of Wasserstein.

5.3.1 Value-Aware Model Learning (VAML) Loss

In model-based RL it is very common to have a model-learning process that is agnostic to the specific planing process. In such cases, the usefulness of the model for the specific planning procedure comes as an afterthought. In contrast, the basic idea behind VAML [40] is to learn a model tailored to the planning algorithm that intends to use it. To illustrate this idea, consider Bellman equations [23] which are at the core of many planning and RL algorithms [102]:

$$Q(s,a) = R(s,a) + \gamma \int T(s'|s,a) f\big(Q(s',.)\big) ds' \ ,$$

where f can generally be any arbitrary operator [63] such as max. We also define:

$$v(s') := f\big(Q(s',.)\big) \ .$$

A good model $\widehat{T}$ could then be thought of as the one that minimizes the error:

$$\begin{aligned} l(T,\widehat{T})(s,a) &= R(s,a) + \gamma \int T(s'|s,a) v(s') ds' \\ &- R(s,a) - \gamma \int \widehat{T}(s'|s,a) v(s') ds' \\ &= \gamma \int \big(T(s'|s,a) - \widehat{T}(s'|s,a)\big) v(s') ds' \end{aligned}$$

Note that minimizing this objective requires access to the value function in the first place, but we can obviate this need by leveraging Holder's inequality:

$$\begin{aligned} l(\widehat{T},T)(s,a) &= \gamma \int \big(T(s'|s,a) - \widehat{T}(s'|s,a)\big) v(s') ds' \\ &\leq \gamma \Big\| T(s'|s,a) - \widehat{T}(s'|s,a) \Big\|_1 \|v\|_\infty \end{aligned}$$

Further, we can use Pinsker's inequality to write:

$$\Big\| T(\cdot|s,a) - \widehat{T}(\cdot|s,a) \Big\|_1 \leq \sqrt{2KL\big(T(\cdot|s,a) || \widehat{T}(\cdot|s,a)\big)} \ .$$

This justifies the use of maximum likelihood estimation for model learning, a common practice in model-based RL [15, 1, 2], since maximum likelihood estimation is equivalent to empirical KL minimization.

However, there exists a major drawback with the KL objective, namely that it ignores the structure of the value function during model learning.

As a simple example, if the value function is constant through the state-space, any randomly chosen model $\widehat{T}$ will, in fact, yield zero Bellman error. However, a model learning algorithm that ignores the structure of value function can potentially require many samples to provide any guarantee about the performance of the learned policy.

Consider the objective function $l(T, \widehat{T})$, and notice again that v itself is not known so we cannot directly optimize for this objective. Farahmand et al. [40] proposed to search for a model that results in lowest error given all possible value functions belonging to a specific class:

$$L(T,\hat{T})(s,a) = \sup_{v\in\mathcal{F}} \Big|\int \big(T(s' \mid s,a) - \widehat{T}(s' \mid s,a)\big) v(s')ds'\Big|^2 \tag{5.1}$$

Note that minimizing this objective is shown to be tractable if, for example, $\mathcal{F}$ is restricted to the class of exponential functions. Observe that the VAML objective (5.1) is similar to the dual of Wasserstein:

$$W(\mu_1,\mu_2) = \sup_{f:K_{d,d_{\mathbb{R}}}(f)\leq 1} \int \big(\mu_1(s) - \mu_2(s)\big) f(s)ds \ ,$$

but the difference is in the space of value functions. In the next section I show that the space of value functions are the same under certain conditions.

5.3.2 Lipschitz Generalized Value Iteration

I show that solving for a class of Bellman equations yields a Lipschitz value function. Our proof is in the context of GVI [63], which defines Value Iteration [23] with arbitrary backup operators. We make use of the following lemmas.

Lemma 5.3.1. *Given a non-expansion $f : \mathcal{S} \to \mathbb{R}$:*

$$K^{\mathcal{A}}_{d_{\mathcal{S}},d_{\mathbb{R}}}\Big(\int T(s'|s,a)f(s')ds'\Big) \leq K^{\mathcal{A}}_{d_{\mathcal{S}},W}(T) \ .$$

Proof. Starting from the definition, we write:

$$\begin{aligned} K^{\mathcal{A}}_{d_{\mathcal{S}},d_{\mathbb{R}}}\Big(\int T(s'|s,a)f(s')ds'\Big) &= \sup_a \sup_{s_1,s_2} \frac{\Big|\int \big(T(s'|s_1,a) - T(s'|s_2,a)\big) f(s')ds'\Big|}{d(s_1,s_2)} \\ &\leq \sup_a \sup_{s_1,s_2} \frac{\Big|\sup_g \int \big(T(s'|s_1,a) - T(s'|s_2,a)\big) g(s')ds'\Big|}{d(s_1,s_2)} \\ &\quad \text{(where } K_{d_{\mathcal{S}},d_{\mathbb{R}}}(g) \leq 1) \\ &= \sup_a \sup_{s_1,s_2} \frac{\sup_g \int \big(T(s'|s_1,a) - T(s'|s_2,a)\big) g(s')ds'}{d(s_1,s_2)} \\ &= \sup_a \sup_{s_1,s_2} \frac{W\big(T(\cdot|s_1,a), T(\cdot|s_2,a)\big)}{d(s_1,s_2)} = K^{\mathcal{A}}_{d_{\mathcal{S}},W}(T) \ . \end{aligned}$$

□

Lemma 5.3.2. *The following operators are non-expansion ($K_{\|\cdot\|_\infty, d_R}(\cdot) = 1$):*

1. *max*(x), *mean*(x)
2. ϵ-*greedy*$(x) := \epsilon$ *mean*$(x) + (1-\epsilon)$*max*(x)
3. $mm_\beta(x) := \frac{\log \frac{\sum_i e^{\beta x_i}}{n}}{\beta}$

Proof. 1 is proven by Littman & Szepsevari [63]. 2 follows from 1: (metrics not shown for brevity)

$$\begin{aligned} K(\epsilon\text{-greedy(x)}) &= K\big(\epsilon \text{ mean}(x) + (1-\epsilon)\text{max}(x)\big) \\ &\leq \epsilon K\big(\text{mean}(x)\big) + (1-\epsilon)K\big(\text{max}(x)\big) \\ &= 1 \end{aligned}$$

Finally, 3 is proven multiple times in the literature. [7, 71, 73] □

I now present the main result of the section.

Theorem 5.3.3. *For any choice of backup operator f outlined in Lemma 5.3.2, GVI computes a value function with a Lipschitz constant bounded by $\frac{K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(R)}{1-\gamma K_{d_\mathcal{S}, W}(T)}$ if $\gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T) < 1$.*

Proof. In the nth round of GVI updates we have:

$$\widehat{Q}_{n+1}(s,a) \leftarrow R(s,a) + \gamma \int T(s' \mid s,a) f\big(\widehat{Q}_n(s', \cdot)\big) ds'.$$

First observe that:

$$\begin{aligned} &K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(\widehat{Q}_{n+1}) \\ &\leq K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(R) + \gamma K^{\mathcal{A}}_{d_\mathcal{S}, d_\mathbb{R}}\Big(\int T(s' \mid s,a) f\big(\widehat{Q}_n(s', \cdot)\big) ds'\Big) \\ &\big(\text{due to Lemma (5.3.1)}\big) \\ &\leq K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(R) + \gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T)\ K_{d_\mathcal{S}, \mathbb{R}}\Big(f\big(\widehat{Q}_n(s, \cdot)\big)\Big) \\ &\big(\text{due to Composition Lemma }\big) \\ &\leq K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(R) + \gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T) K_{\|\cdot\|_\infty, d_\mathbb{R}}(f) K^{\mathcal{A}}_{d_\mathcal{S}, d_\mathbb{R}}(\widehat{Q}_n) \\ &\big(\text{due to Lemma (5.3.2), the non-expansion property of } f\big) \\ &= K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(R) + \gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T) K^{\mathcal{A}}_{d_\mathcal{S}, d_\mathbb{R}}(\widehat{Q}_n) \end{aligned}$$

Equivalently:

$$\begin{aligned} K^{\mathcal{A}}_{d_\mathcal{S}, d_R}(\widehat{Q}_{n+1}) \quad &\leq \quad K^{\mathcal{A}}_{d_\mathcal{S}, d_\mathbb{R}}(R) \sum_{i=0}^{n} \big(\gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T)\big)^i \\ &+ \quad \big(\gamma K^{\mathcal{A}}_{d_\mathcal{S}, W}(T)\big)^n\ K^{\mathcal{A}}_{d_\mathcal{S}, d_\mathbb{R}}(\widehat{Q}_0)\ . \end{aligned}$$

By computing the limit of both sides, we get:

$$\begin{aligned}
\lim_{n\to\infty} K^{\mathcal{A}}_{d_\mathcal{S},d_\mathbb{R}}(\widehat{Q}_n) &\leq \lim_{n\to\infty} K^{\mathcal{A}}_{d_\mathcal{S},d_\mathbb{R}}(R)\sum_{i=0}^{n}\big(\gamma K^{\mathcal{A}}_{d_\mathcal{S},W}(T)\big)^i \\
&+ \lim_{n\to\infty}\big(\gamma K^{\mathcal{A}}_{d_\mathcal{S},W}(T)\big)^n\ K^{\mathcal{A}}_{d_\mathcal{S},d_\mathbb{R}}(\widehat{Q}_0) \\
&= \frac{K^{\mathcal{A}}_{d_\mathcal{S},d_R}(R)}{1-\gamma K_{d_\mathcal{S},W}(T)} + 0 \ ,
\end{aligned}$$

where we used the fact that

$$\lim_{n\to\infty}\big(\gamma K^{\mathcal{A}}_{d_\mathcal{S},W}(T)\big)^n = 0 \ .$$

This concludes the proof. □

Now notice that as defined earlier:

$$\widehat{V}_n(s) := f\big(\widehat{Q}_n(s,\cdot)\big) \ ,$$

so as a relevant corollary of our theorem we get:

$$\begin{aligned}
K_{d_\mathcal{S},d_\mathbb{R}}\big(v(s)\big) &= \lim_{n\to\infty} K_{d_\mathcal{S},d_\mathbb{R}}(\widehat{V}_n) \\
&= \lim_{n\to\infty} K_{d_\mathcal{S},d_\mathbb{R}}\Big(f\big(\widehat{Q}_n(s,\cdot)\big)\Big) \\
&\leq \lim_{n\to\infty} K^{\mathcal{A}}_{d_\mathcal{S},d_\mathbb{R}}(\widehat{Q}_n) \\
&\leq \frac{K^{\mathcal{A}}_{d_\mathcal{S},d_R}(R)}{1-\gamma K_{d_\mathcal{S},W}(T)} \ .
\end{aligned}$$

That is, solving for the fixed point of this general class of Bellman equations results in a Lipschitz state-value function.

5.3.3 Equivalence Between VAML and Wasserstein

I now show that minimizing for the VAML objective is the same as minimizing the Wasserstein metric.

Consider again the VAML objective:

$$L(T,\hat{T})(s,a) = \sup_{v\in\mathcal{F}}\left|\int\big(T(s'\mid s,a)-\widehat{T}(s'\mid s,a)\big)v(s')ds'\right|^2$$

where $\mathcal{F}$ can generally be any class of functions. From our theorem, however, the space of value functions $\mathcal{F}$ should be restricted to Lipschitz functions. Moreover, it is easy to design an MDP and a policy such that a desired Lipschitz value function is attained.

This space $\mathcal{L}_C$ can then be defined as follows:

$$\mathcal{L}_C = \{f : K_{d_\mathcal{S},d_\mathbb{R}}(f) \leq C\} \ ,$$

where

$$C = \frac{K^{\mathcal{A}}_{d_{\mathcal{S}},d_R}(R)}{1-\gamma K_{d_{\mathcal{S}},W}(T)} \ .$$

So we can rewrite the VAML objective L as follows:

$$\begin{aligned} L\big(T,\hat{T}\big)(s,a) &= \sup_{f\in\mathcal{L}_C}\Big|\int f(s)\big(T(s' \mid s,a)-\widehat{T}(s' \mid s,a)\big)ds'\Big|^2 \\ &= \sup_{f\in\mathcal{L}_C}\Big|\int C\frac{f(s)}{C}\big(T(s' \mid s,a)-\widehat{T}(s' \mid s,a)\big)\ ds'\Big|^2 \\ &= C^2\sup_{g\in\mathcal{L}_1}\Big|\int g(s)\big(T(s' \mid s,a)-\widehat{T}(s' \mid s,a)\big)\ ds'\Big|^2 \ . \end{aligned}$$

It is clear that a function g that maximizes the Kantorovich-Rubinstein dual form:

$$\sup_{g\in\mathcal{L}_1}\int g(s)\big(T(s' \mid s,a)-\widehat{T}(s' \mid s,a)\big)\ ds' := W(T(\cdot|s,a),\widehat{T}(\cdot|s,a)) \ ,$$

will also maximize:

$$L\big(T,\hat{T}\big)(s,a) = \Big|\int g(s)\big(T(s' \mid s,a)-\widehat{T}(s' \mid s,a)\big)\ ds'\Big|^2 \ .$$

This is due to the fact that $\forall g\in\mathcal{L}_1 \Rightarrow -g\in\mathcal{L}_1$ and so computing absolute value or squaring the term will not change arg max in this case.
As a result:

$$L\big(T,\widehat{T}\big)(s,a) = \Big(C\ W\big(T(\cdot|s,a),\widehat{T}(\cdot|s,a)\big)\Big)^2 \ .$$

This highlights a nice property of Wasserstein, namely that minimizing this metric yields a value-aware model. Therefore, the strong theoretical properties shown for value-aware loss [40] further justifies our choice of Wasserstein assuming that the Lipschitz assumption holds. I will use the Wasserstein metric in the remainder of the chapter.

5.4 Understanding the Compounding Error Phenomenon

To extract a prediction with a horizon $n > 1$, model-based algorithms typically apply the model for n steps by taking the state input in step t to be the state output from the step $t-1$. Previous work has shown that model error can result in poor long-horizon predictions and ineffective planning [107, 108]. Observed even beyond reinforcement learning [65, 115], this is referred to as the *compounding error phenomenon.* The goal of this section is to provide a bound on multi-step prediction error of a model. In light of the previous section, I formalize the notion of model accuracy below:

Definition 5.4.1. *Given an MDP with a transition function T, we identify a Lipschitz model F_g as Δ-accurate if its induced $\widehat{T}$ satisfies:*

$$\forall s\ \forall a \quad W\big(\widehat{T}(\cdot \mid s,a),T(\cdot \mid s,a)\big) \le \Delta \ .$$

We want to express the multi-step Wasserstein error in terms of the single-step Wasserstein error and the Lipschitz constant of the transition function $\widehat{T}_{\mathcal{G}}$. I provide a bound on the Lipschitz constant of $\widehat{T}_{\mathcal{G}}$ using the following lemma:

Lemma 5.4.1. *A generalized transition function $\widehat{T}_{\mathcal{G}}$ induced by a Lipschitz model class F_g is Lipschitz with a constant:*

$$K^{\mathcal{A}}_{W,W}(\widehat{T}_{\mathcal{G}}) := \sup_a \sup_{\mu_1,\mu_2} \frac{W\big(\widehat{T}_{\mathcal{G}}(\cdot|\mu_1,a),\widehat{T}_{\mathcal{G}}(\cdot|\mu_2,a)\big)}{W(\mu_1,\mu_2)} \leq K_F$$

Proof.

$$\begin{aligned}
&W\big(\widehat{T}(\cdot \mid \mu_1,a),\widehat{T}(\cdot \mid \mu_2,a)\big)\\
&:= \inf_j \int_{s_1'}\int_{s_2'} j(s_1',s_2')d(s_1',s_2')ds_1'ds_2'\\
&= \inf_j \int_{s_1}\int_{s_2}\int_{s_1'}\int_{s_2'} \sum_f \mathbb{1}\big(f(s_1)=s_1' \wedge f(s_2)=s_2'\big)j(s_1,s_2,f)d(s_1',s_2')ds_1'ds_2'ds_1ds_2\\
&= \inf_j \int_{s_1}\int_{s_2} \sum_f j(s_1,s_2,f)d\big(f(s_1),f(s_2)\big)ds_1ds_2\\
&\leq K_F \inf_j \int_{s_1}\int_{s_2} \sum_f g(f|a)j(s_1,s_2)d(s_1,s_2)ds_1ds_2\\
&= K_F \sum_f g(f|a) \inf_j \int_{s_1}\int_{s_2} j(s_1,s_2)d(s_1,s_2)ds_1ds_2\\
&= K_F \sum_f g(f|a)W(\mu_1,\mu_2) = K_F W(\mu_1,\mu_2)
\end{aligned}$$

□

Intuitively, Lemma 5.4.1 states that, if the two input distributions are similar, then for any action the output distributions given by $\widehat{T}_{\mathcal{G}}$ are also similar up to a K_F factor.

Given the one-step error (Definition 5.4.1), a start state distribution μ and a fixed sequence of actions $a_0,...,a_{n-1}$, we desire a bound on n-step error:

$$\delta(n) := W\big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), T^n_{\mathcal{G}}(\cdot \mid \mu)\big) \ ,$$

where $\widehat{T}^n_{\mathcal{G}}(\cdot|\mu) := \underbrace{\widehat{T}_{\mathcal{G}}(\cdot|\widehat{T}_{\mathcal{G}}(\cdot|...\widehat{T}_{\mathcal{G}}(\cdot|\mu,a_0)...,a_{n-2}),a_{n-1})}_{n \text{ recursive calls}}$ and $T^n_{\mathcal{G}}(\cdot \mid \mu)$ is defined similarly. I provide a useful lemma followed by the theorem.

Lemma 5.4.2. *(Composition Lemma) Define three metric spaces (M_1,d_1), (M_2,d_2), and (M_3,d_3). Define Lipschitz functions $f: M_2 \mapsto M_3$ and $g: M_1 \mapsto M_2$ with constants $K_{d_2,d_3}(f)$ and $K_{d_1,d_2}(g)$. Then, $h: f\circ g: M_1 \mapsto M_3$ is Lipschitz with constant $K_{d_1,d_3}(h) \leq K_{d_2,d_3}(f)K_{d_1,d_2}(g)$.*

Proof.

$$\begin{aligned}
K_{d_1,d_3}(h) &= \sup_{s1,s_2} \frac{d_3\Big(f\big(g(s_1)\big), f\big(g(s_2)\big)\Big)}{d_1(s_1,s_2)} \\
&= \sup_{s_1,s_2} \frac{d_2\big(g(s_1), g(s_2)\big)}{d_1(s_1,s_2)} \frac{d_3\Big(f\big(g(s_1)\big), f\big(g(s_2)\big)\Big)}{d_2\big(g(s_1), g(s_2)\big)} \\
&\leq \sup_{s_1,s_2} \frac{d_2\big(g(s_1), g(s_2)\big)}{d_1(s_1,s_2)} \sup_{s_1,s_2} \frac{d_3\big(f(s_1), f(s_2)\big)}{d_2(s_1,s_2)} \\
&= K_{d_1,d_2}(g) K_{d_2,d_3}(f).
\end{aligned}$$

□

Similar to composition, we can show that summation preserves Lipschitz continuity with a constant bounded by the sum of the Lipschitz constants of the two functions.

Theorem 5.4.3. *Define a Δ-accurate $\widehat{T}_{\mathcal{G}}$ with the Lipschitz constant K_F and an MDP with a Lipschitz transition function $T_{\mathcal{G}}$ with constant K_T. Let $\bar{K} = \min\{K_F, K_T\}$. Then $\forall n \geq 1$:*

$$\delta(n) := W\big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), T^n_{\mathcal{G}}(\cdot \mid \mu)\big) \leq \Delta \sum_{i=0}^{n-1} (\bar{K})^i \ .$$

Proof. We construct a proof by induction. Using Kantarovich-Rubinstein duality (Lipschitz property of f not shown for brevity) we first prove the base of induction:

$$\begin{aligned}
\delta(1) &:= W\big(\widehat{T}_{\mathcal{G}}(\cdot \mid \mu, a_0), T_{\mathcal{G}}(\cdot \mid \mu, a_0)\big) \\
&:= \sup_f \int\int \big(\widehat{T}(s' \mid s, a_0) - T(s' \mid s, a_0)\big) f(s') \mu(s) \ ds \ ds' \\
&\leq \int \underbrace{\sup_f \int \big(\widehat{T}(s'|s, a_0) - T(s'|s, a_0)\big) f(s') \ ds'}_{= W\big(\widehat{T}(\cdot|s,a_0), T(\cdot|s,a_0)\big) \text{due to duality (5.3.1)}} \mu(s) \ ds \\
&= \int \underbrace{W\big(\widehat{T}(\cdot \mid s, a_0), T(\cdot \mid s, a_0)\big)}_{\leq \Delta \text{ due to Definition 5.4.1}} \mu(s) \ ds \\
&\leq \int \Delta \ \mu(s) \ ds = \Delta \ .
\end{aligned}$$

We now prove the inductive step. Assuming $\delta(n-1) := W\big(\widehat{T}^{n-1}_{\mathcal{G}}(\cdot \mid \mu), T^{n-1}_{\mathcal{G}}(\cdot \mid \mu)\big) \leq \Delta \sum_{i=0}^{n-2} (K_F)^i$ we can write:

$$\begin{aligned}
\delta(n) &:= W\big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), T^n_{\mathcal{G}}(\cdot \mid \mu)\big) \\
&\leq W\Big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), \widehat{T}_{\mathcal{G}}\big(\cdot \mid T^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}\big)\Big) \\
&+ W\Big(\widehat{T}_{\mathcal{G}}\big(\cdot \mid T^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}\big), T^n_{\mathcal{G}}(\cdot \mid \mu)\Big) \text{ (Triangle ineq)} \\
&= W\Big(\widehat{T}_{\mathcal{G}}(\cdot \mid \widehat{T}^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}), \widehat{T}_{\mathcal{G}}\big(\cdot \mid T^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}\big)\Big) \\
&\quad + W\Big(\widehat{T}_{\mathcal{G}}\big(\cdot \mid T^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}\big), T_{\mathcal{G}}\big(\cdot \mid T^{n-1}_{\mathcal{G}}(\cdot \mid \mu), a_{n-1}\big)\Big)
\end{aligned}$$

We now use Lemma 5.4.1 and Definition 5.4.1 to upper bound the first and the second term of the last line respectively.

$$\begin{aligned}
\delta(n) &\leq K_F\ W\big(\widehat{T}^{n-1}_{\mathcal{G}}(\cdot \mid \mu), T^{n-1}_{\mathcal{G}}(\cdot \mid \mu)\big) + \Delta \\
&= K_F\ \delta(n-1) + \Delta \leq \Delta \sum_{i=0}^{n-1} (K_F)^i\ . \qquad (5.2)
\end{aligned}$$

Note that in the triangle inequality, we may replace $\widehat{T}_{\mathcal{G}}\big(\cdot \mid\ T^{n-1}_{\mathcal{G}}(\cdot \mid \mu)\big)$ with $T_{\mathcal{G}}\big(\cdot \mid \widehat{T}^{n-1}_{\mathcal{G}}(\cdot \mid \mu)\big)$ and follow the same basic steps to get:

$$W\big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), T^n_{\mathcal{G}}(\cdot \mid \mu)\big) \leq \Delta \sum_{i=0}^{n-1} (K_T)^i\ . \qquad (5.3)$$

Combining (5.2) and (5.3) allows us to write:

$$\begin{aligned}
\delta(n) &= W\big(\widehat{T}^n_{\mathcal{G}}(\cdot \mid \mu), T^n_{\mathcal{G}}(\cdot \mid \mu)\big) \\
&\leq \min\left\{\Delta \sum_{i=0}^{n-1} (K_T)^i, \Delta \sum_{i=0}^{n-1} (K_F)^i\right\} \\
&= \Delta \sum_{i=0}^{n-1} (\bar{K})^i\ ,
\end{aligned}$$

which concludes the proof. □

There exist similar results in the literature relating one-step transition error to multi-step transition error and sub-optimality bounds for planning with an approximate model. The Simulation Lemma [54, 97] is for discrete state MDPs and relates error in the one-step model to the value obtained by using it for planning. A related result for continuous state-spaces [53] bounds the error in estimating the probability of a trajectory using total variation. A second related result [115] provides a slightly looser bound for prediction error in the deterministic case—this result can be thought of as a generalization of their result to the probabilistic case.

5.5 Value Error with Lipschitz Models

We next investigate the error in the state-value function induced by a Lipschitz model class. To answer this question, we consider an MRP M_1 denoted by $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$ and a second MRP M_2

that only differs from the first in its transition function $\langle \mathcal{S}, \mathcal{A}, \widehat{T}, R, \gamma\rangle$. Let $\mathcal{A} = \{a\}$ be the action set with a single action a. We further assume that the reward function is only dependent upon state. We first express the state-value function for a start state s with respect to the two transition functions. By δ_s below, we mean a Dirac delta function denoting a distribution with probability 1 at state s.

$$V_T(s) := \sum_{n=0}^{\infty} \gamma^n \int T^n_{\mathcal{G}}(s'|\delta_s) R(s')\ ds' \ ,$$

$$V_{\widehat{T}}(s) := \sum_{n=0}^{\infty} \gamma^n \int \widehat{T}^n_{\mathcal{G}}(s'|\delta_s) R(s')\ ds' \ .$$

Next we derive a bound on $\big|V_T(s) - V_{\widehat{T}}(s)\big|\ \forall s$.

Theorem 5.5.1. *Assume a Lipschitz model class F_g with a Δ-accurate $\widehat{T}$ with $\bar{K} = \min\{K_F, K_T\}$. Further, assume a Lipschitz reward function with constant $K_R = K_{d_{\mathcal{S}},\mathbb{R}}(R)$. Then $\forall s \in \mathcal{S}$ and $\bar{K} \in [0, \frac{1}{\gamma})$*

$$\big|V_T(s) - V_{\widehat{T}}(s)\big| \leq \frac{\gamma K_R \Delta}{(1-\gamma)(1-\gamma\bar{K})} \ .$$

Proof. We first define the function $f(s) = \frac{R(s)}{K_R}$. It can be observed that $K_{d_{\mathcal{S}},\mathbb{R}}(f) = 1$. We now write:

$$\begin{aligned}
&V_T(s) - V_{\widehat{T}}(s) \\
&= \sum_{n=0}^{\infty} \gamma^n \int R(s')\big(T^n_{\mathcal{G}}(s' \mid \delta_s) - \widehat{T}^n_{\mathcal{G}}(s' \mid \delta_s)\big)\ ds' \\
&= K_R \sum_{n=0}^{\infty} \gamma^n \int f(s')\big(T^n_{\mathcal{G}}(s' \mid \delta_s) - \widehat{T}^n_{\mathcal{G}}(s' \mid \delta_s)\big)\ ds'
\end{aligned}$$

Let $\mathcal{F} = \{h : K_{d_{\mathcal{S}},\mathbb{R}}(h) \leq 1\}$. Then given $f \in \mathcal{F}$:

$$\begin{aligned}
&K_R \sum_{n=0}^{\infty} \gamma^n \int f(s')\big(T^n_{\mathcal{G}}(s'|\delta_s) - \widehat{T}^n_{\mathcal{G}}(s'|\delta_s)\big) ds' \\
\leq\ & K_R \sum_{n=0}^{\infty} \gamma^n \underbrace{\sup_{f\in\mathcal{F}} \int f(s')\big(T^n_{\mathcal{G}}(s' \mid \delta_s) - \widehat{T}^n_{\mathcal{G}}(s' \mid \delta_s)\big) ds'}_{:=W\left(T^n_{\mathcal{G}}(.|\delta_s), \widehat{T}^n_{\mathcal{G}}(.|\delta_s)\right) \text{due to duality (5.3.1)}} \\
=\ & K_R \sum_{n=0}^{\infty} \gamma^n \underbrace{W\big(T^n_{\mathcal{G}}(. \mid \delta_s), \widehat{T}^n_{\mathcal{G}}(. \mid \delta_s)\big)}_{\leq \sum_{i=0}^{n-1} \Delta(\bar{K})^i \text{ due to Theorem 5.4.3}} \\
\leq\ & K_R \sum_{n=0}^{\infty} \gamma^n \sum_{i=0}^{n-1} \Delta(\bar{K})^i \\
=\ & K_R \Delta \sum_{n=0}^{\infty} \gamma^n \frac{1-\bar{K}^n}{1-\bar{K}} \\
=\ & \frac{\gamma K_R \Delta}{(1-\gamma)(1-\gamma\bar{K})} \ .
\end{aligned}$$

We can derive the same bound for $V_{\widehat{T}}(s) - V_T(s)$ using the fact that Wasserstein distance is a metric, and therefore symmetric, thereby completing the proof. □

Regarding the tightness of these bounds, I can show that when the transition model is deterministic and linear then Theorem 5.4.3 provides a tight bound. Moreover, if the reward function is linear, the bound provided by Theorem 5.5.1 is tight.

5.6 Experiments with One-step Models

My first goal in this section is to compare TV, KL, and Wasserstein in terms of the ability to best quantify error of an imperfect model, and do so in a simple and clear setting.

To this end, I built finite MRPs with random transitions, $|\mathcal{S}| = 10$ states, and $\gamma = 0.95$. In the first case the reward signal is randomly sampled from $[0, 10]$, and in the second case the reward of an state is the index of that state, so small Euclidean norm between two states is an indication of similar values. For 10^5 trials, I generated an MRP and a random model, and then computed model error and planning error (Figure 5.3).

We understand a good metric as the one that computes a model error with a high correlation with value error. I show these correlations for different values of γ in Figure 5.4.

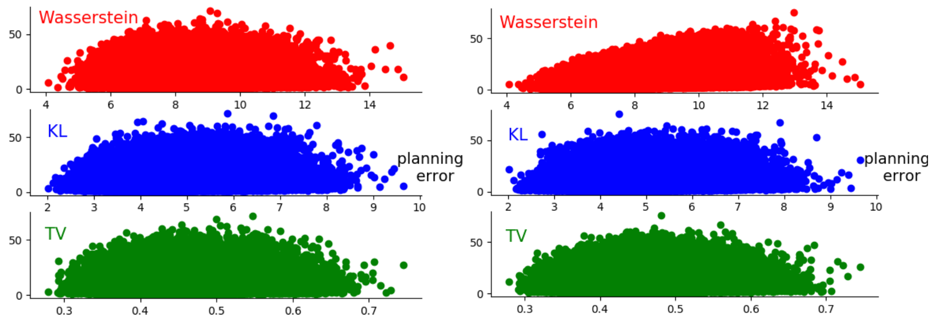


Figure 5.3: Value error (x axis) and model error (y axis). When the reward is the index of the state (right), correlation between Wasserstein error and value-prediction error is high. This highlights the fact that when closeness in the state-space is an indication of similar values, Wasserstein can be a powerful metric for model-based RL. Note that Wasserstein provides no advantage given random rewards (left).

It is known that controlling the Lipschitz constant of neural nets can help in terms of improving generalization error due to a lower bound on Rademacher complexity [74, 18]. It then follows from Theorems 5.4.3 and 5.5.1 that controlling the Lipschitz constant of a learned transition model can achieve better error bounds for multi-step and value predictions. To enforce this constraint during learning, we bound the Lipschitz constant of various operations used in building neural network.

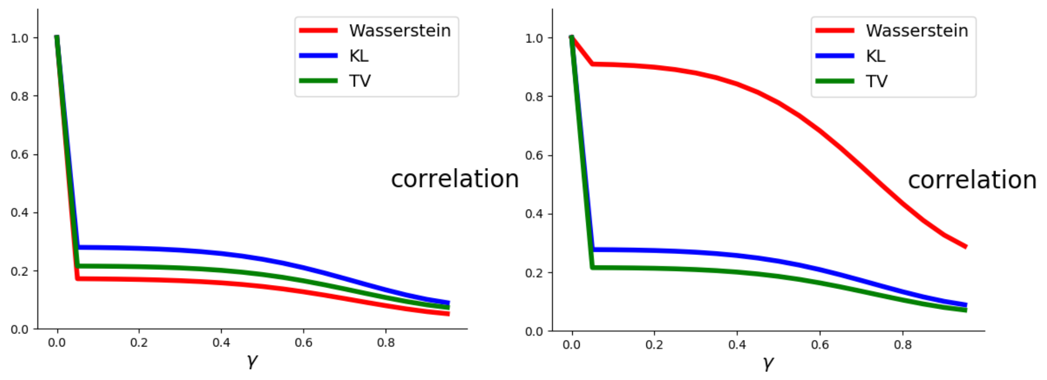


Figure 5.4: Correlation between value-prediction error and model error for the three metrics using random rewards (left) and index rewards (right). Given a useful notion of state similarities, low Wasserstein error is a better indication of planning error.

Function f	Definition	Lipschitz constant $K_{\|\|_p,\|\|_p}(f)$		
		$p=1$	$p=2$	$p=\infty$
ReLu : $R^n \to R^n$	$\max\{0, x_i\}$	1	1	1
$+b : R^n \to R^n, \forall b \in R^n$	$:= x+b$	1	1	1
$R^n \to R^m,\ \forall W \in R^{m\times n}$	$\times W(x) := Wx$	$\sum_j \|W_j\|_\infty$	$\sqrt{\sum_j \|W_j\|_2^2}$	$\sup_j \|W_j\|_1$

Table 5.1: Lipschitz constant for various functions used in a neural network. Here, W_j denotes the jth row of a weight matrix W.

The bound on the constant of the entire neural network then follows from Lemma 5.4.2. In Table 5.1, we provide Lipschitz constant for operations used in our experiments. We quantify these results for different p-norms $\|\cdot\|_p$.

Given these simple methods for enforcing Lipschitz continuity, I performed empirical evaluations to understand the impact of Lipschitz continuity of transition models, specifically when the transition model is used to perform multi-step state-predictions and policy improvements. I chose two standard domains: Cart Pole and Pendulum. In Cart Pole, I trained a network on a dataset of $15 * 10^3$ tuples $\langle s, a, s'\rangle$. During training, I ensured that the weights of the network are smaller than k. For each k, I performed 20 independent model estimation, and chose the model with median cross-validation error.

Using the learned model, along with the actual reward signal of the environment, I then performed stochastic actor-critic RL. [19, 103] This required an interaction between the policy and the learned model for relatively long trajectories. To measure the usefulness of the model, I then tested the learned policy on the actual domain. I repeated this experiment on Pendulum. To train the neural transition model for this domain we used 10^4 samples. Notably, I used deterministic policy gradient [92] for training the policy network with the hyper parameters suggested by [61]. I report these results in Figure 5.5.

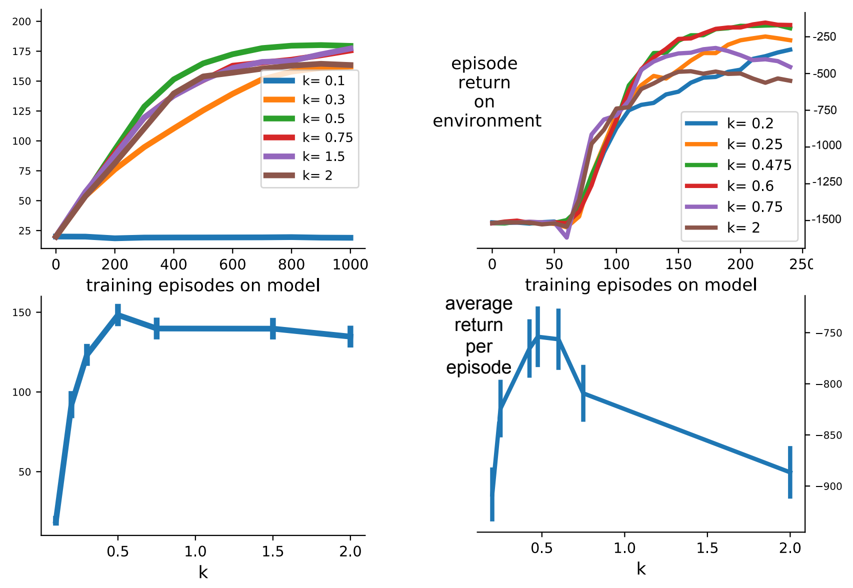


Figure 5.5: Impact of controlling the Lipschitz constant of learned models in Cart Pole (left) and Pendulum (right). An intermediate value of k (Lipschitz constant) yields the best performance.

Observe that an intermediate Lipschitz constant yields the best result. Consistent with the theory, controlling the Lipschitz constant in practice can combat the compounding errors and can help in the value estimation problem. This ultimately results in learning a better policy.

I next examined if the benefits carry over to stochastic settings. To capture stochasticity we need an algorithm to learn a Lipschitz model class (Definition 5.2.1). I used an EM algorithm to joinly learn a set of functions f, parameterized by $\theta = \{\theta^f : f \in F_g\}$, and a distribution over functions g. Note that in practice our dataset only consists of a set of samples $\langle s, a, s' \rangle$ and does not include the function the sample is drawn from. Hence, I consider this as our latent variable z. As is standard with EM, we start with the log-likelihood objective (for simplicity of presentation I assume a single action in the derivation):

$$\begin{aligned} L(\theta) &= \sum_{i=1}^{N} \log p(s_i, {s_i}'; \theta) \\ &= \sum_{i=1}^{N} \log \sum_{f} p(z_i = f, s_i, {s_i}'; \theta) \\ &= \sum_{i=1}^{N} \log \sum_{f} q(z_i = f | s_i, {s_i}') \frac{p(z_i = f, s_i, {s_i}'; \theta)}{q(z_i = f | s_i, {s_i}')} \\ &\geq \sum_{i=1}^{N} \sum_{f} q(z_i = f | s_i, {s_i}') \log \frac{p(z_i = f, s_i, {s_i}'; \theta)}{q(z_i = f | s_i, {s_i}')} \;, \end{aligned}$$

where I used Jensen's inequality and concavity of log in the last line. This derivation leads to the following EM algorithm.

In the M step, find θ_t by solving for:

$$\arg\max_{\theta} \sum_{i=1}^{N} \sum_{f} q_{t-1}(z_i = f | s_i, {s_i}') \log \frac{p(z_i = f, s_i, {s_i}'; \theta)}{q_{t-1}(z_i = f | s_i, {s_i}')}$$

In the E step, compute posteriors:

$$q_t(z_i = f | s_i, {s_i}') = \frac{p(s_i, {s_i}' | z_i = f; {\theta_t}^f) g(z_i = f; \theta_t)}{\sum_f p(s_i, {s_i}' | z_i = f; {\theta_t}^f) g(z_i = f; \theta_t)} \ .$$

Note that we assume each point is drawn from a neural network f with probability:

$$p\big(s_i, {s_i}' | z_i = f; {\theta_t}^f\big) = \mathcal{N}\Big(\big|{s_i}' - f(s_i, {\theta_t}^f)\big|, \sigma^2\Big) \ ,$$

and with a fixed variance σ^2 tuned as a hyper-parameter.

I first used a supervised-learning domain to evaluate the EM algorithm in a simple setting. I generated 30 points from the following 5 functions:

$$\begin{aligned} f_0(x) &= \tanh(x) + 3 \\ f_1(x) &= x * x \\ f_2(x) &= \sin(x) - 5 \\ f_3(x) &= \sin(x) - 3 \\ f_4(x) &= \sin(x) * \sin(x) \ , \end{aligned}$$

and trained 5 neural networks to fit these points. Iterations of a single run is shown in Figure 5.6 and the summary of results is presented in Figure 5.7. Observe that the EM algorithm is effective, and that controlling the Lipschitz constant is again useful.

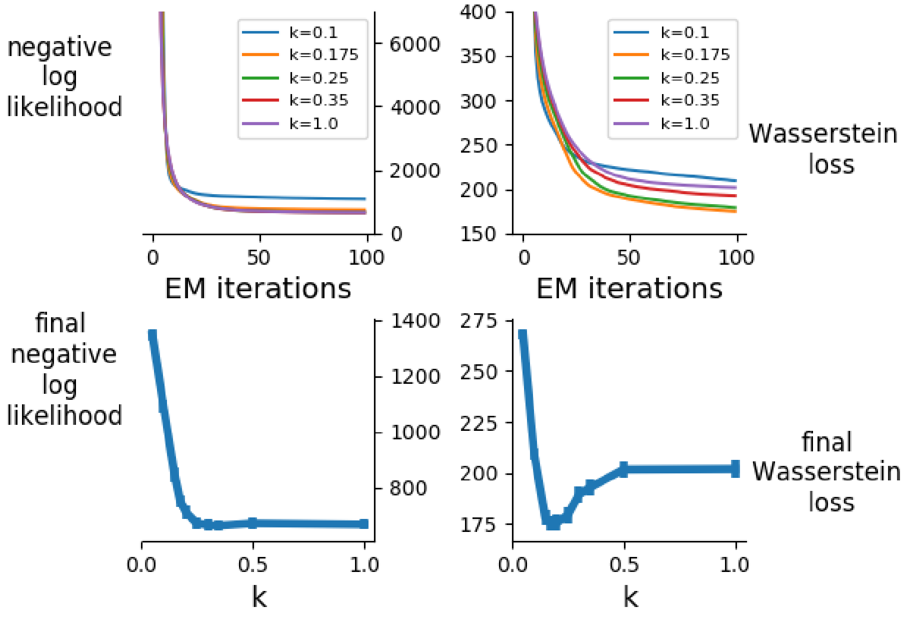


Figure 5.7: Impact of controlling the Lipschitz constant in the supervised-learning domain. Notice the U-shape of final Wasserstein loss with respect to Lipschitz constant k.

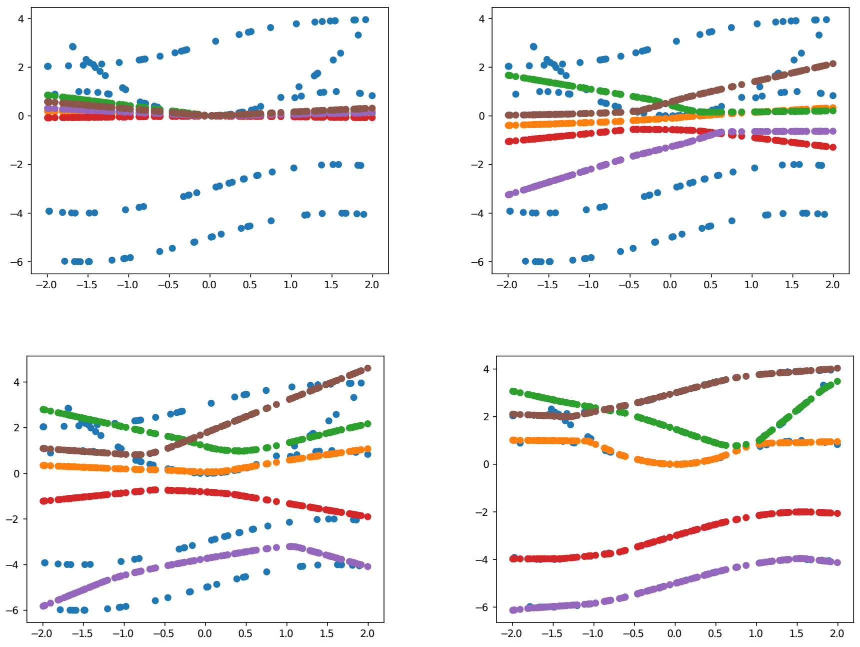


Figure 5.6: A stochastic problem solved by training a Lipschitz model class using EM. The top left figure shows the functions before any training (iteration 0), and the bottom right figure shows the final results (iteration 50).

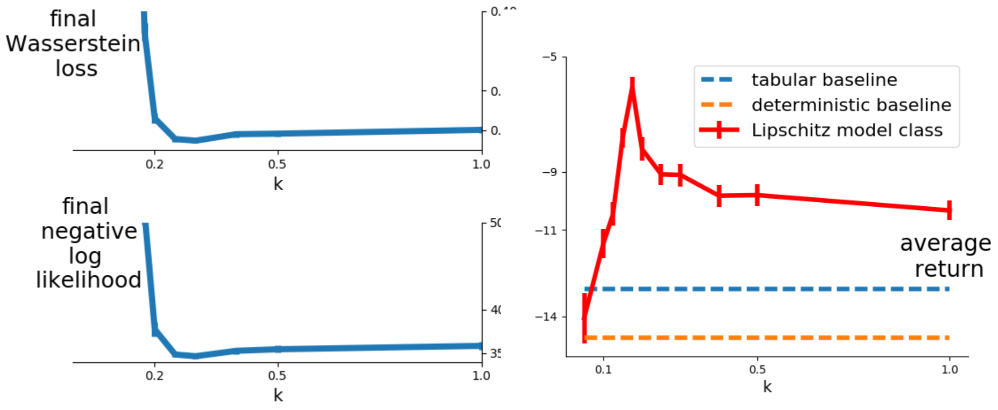


Figure 5.8: Performance of a Lipschitz model class on the gridworld domain. I show model test accuracy (left) and quality of the policy found using the model (right). Notice the poor performance of tabular and expected models.

I next applied EM to train a transition model for an RL setting, namely the gridworld domain from Moerland et al. [68]. Here a useful model needs to capture the stochastic behavior of the two ghosts. I modify the reward to be -1 whenever the agent is in the same cell as either one of the ghosts and 0 otherwise. I performed environmental interactions for 1000 time-steps and measured the return. I compared against standard tabular methods[102], and a deterministic model that predicts expected next state [105, 75]. In all cases I used value iteration for planning.

Results in Figure 5.8 show that tabular models fail due to no generalization, and expected models fail since the ghosts do not move on expectation, a prediction not useful for planner. Performing value iteration with a Lipschitz model class outperforms the baselines.

5.7 M^3 – A Multi-step Model for Model-based Reinforcement Learning

M^3 is an extension of the one-step model—rather than only predicting a single step ahead, it learns to predict $h \in \{1, ..., H\}$ steps ahead using H different functions:

$$\widehat{T}_h(s, \mathbf{a}_h) \approx T_h(s, \mathbf{a}_h)\ ,$$

where $\mathbf{a}_h = \langle a_1, a_2, ..., a_h \rangle$. By M^3, I mean the set of these H functions:

$$\mathrm{M}^3 := \left\{\widehat{T}_h \mid h \in : h \in [1, H]\right\}\ .$$

This model is different than the few examples of multi-step models studied in prior work: Sutton [100] as well as van Seijen et al., [113] considered multi-step models that do not take actions as input, but are implicitly conditioned on the current policy. Similarly, option models are multi-step models that are conditioned on one specific policy and a termination condition [82, 104, 91]. Finally,

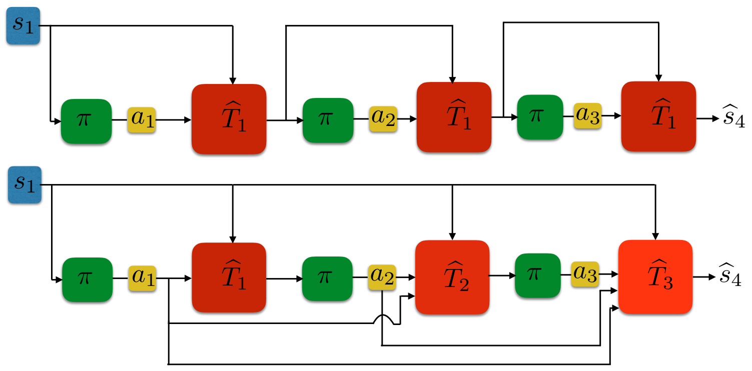


Figure 5.9: (top) a 3-step rollout using a one-step model. (bottom) a 3-step rollout using a multi-step model M^3. Crucially, at each step of the multi-step rollout, the agent uses s_1 as the starting point. The output of each intermediate step is only used to compute the next action.

Silver et al. [93] introduced a multi-step model that directly predicts next values, but the model is defined for prediction tasks.

I now introduce a new rollout procedure using the multi-step model. Note that by an H-step rollout we mean sampling the next action using the agent's fixed policy $\pi : \mathcal{S} \mapsto \Pr(\mathcal{A})$, then computing the next state using the agent's model, and then iterating this procedure for $H-1$ more times. The new rollout procedure that obviates the use of model output as its input in the following timestep. To this end, we derive an approximate expression for:

$$T_H(s, s', \pi) := \Pr(s_{t+H} = s' \mid s_t = s, \pi) \ .$$

Key to our approach is to rewrite $\widehat{T}_H(s, s', \pi) \approx T_H(s, s', \pi)$ in terms of predictions conditioned on action sequences as shown below:

$$\widehat{T}_H(s, s', \pi) \quad := \quad \Pr(s_{t+H} = s' \mid s_t = s, \pi) = \sum_{\mathbf{a}_H} \Pr(\mathbf{a}_H \mid s, \pi) \ \mathbb{1}\big(s' = \underbrace{\widehat{T}_H(s, \mathbf{a}_H)}_{\text{available by M}^3} \ \big) \ .$$

Observe that given the M^3 model introduced above, $\widehat{T}_H(s, \mathbf{a}_h)$ is actually available—we only need to focus on the quantity $\Pr(\mathbf{a}_H | s, \pi)$. Intuitively, we need to compute the probability of taking a sequence of actions of length H in the next H steps starting from s. This probability is clearly determined by the states observed in the next $H-1$ steps, and could be written as follows:

$$\begin{aligned} \Pr(\mathbf{a}_H | s, \pi) = \Pr(a_H | \mathbf{a}_{H-1}, s, \pi) \Pr(\mathbf{a}_{H-1} | s_t = s, \pi) \quad &= \quad \Pr\big(a_H | \widehat{T}_{H-1}(s, \mathbf{a}_{H-1}), \pi\big) \Pr(\mathbf{a}_{H-1} | s, \pi) \\ &= \quad \pi\big(a_H | \underbrace{\widehat{T}_{H-1}(s, \mathbf{a}_{H-1})}_{\text{available by M}^3}\big) \Pr(\mathbf{a}_{H-1} | s, \pi) \ . \end{aligned}$$

We can compute $\Pr(\mathbf{a}_H | s, \pi)$ if we have $\Pr(\mathbf{a}_{H-1} | s, \pi)$. Continuing for $H-1$ more steps:

$$\Pr(\mathbf{a}_H \mid s, \pi) = \pi(a_1 \mid s) \prod_{h=2}^{H} \pi\big(a_h \mid \underbrace{\widehat{T}_{h-1}(s, \mathbf{a}_{h-1})}_{\text{available by M}^3}\big) \ ,$$

which we can compute given the $H-1$ first functions of M^3, namely $\widehat{T}_h$ for $h \in [1, H-1]$. Finally, to compute a rollout, we sample from $\widehat{T}_H(s, s', \pi)$ by sampling from the policy at each step:

$$\widehat{s}_{H+1} = \widehat{T}_H(s, \mathbf{a}_H) \quad \text{where} \quad a_h \sim \pi\big(\cdot \mid \widehat{T}_{h-1}(s, \mathbf{a}_{h-1})\big) \ .$$

Notice that, in the above rollout with M^3, we have used the first state s as the starting point of every single rollout step. Crucially, we do not feed the intermediate state predictions to the model as input. We hypothesize that this approach can combat the compounding error problem by removing one source of error, namely feeding the model a noisy input, which is otherwise present in the rollout using the one-step model. We illustrate the rollout procedure in Figure 5.9 for a better juxtaposition of this new rollout procedure and the standard rollout procedure performed using the one-step model.

5.8 Experiments with Multi-step Models

My goal now is to investigate if the multi-step model can perform better than the one-step model in several model-based scenarios. I set up experiments in background planning and decision-time planning to test this hypothesis.

5.8.1 Background Planning

I compare the one-step model and the multi-step model in the context of value-function estimation. For this experiment, I used the all-action variant of actor-critic algorithm, in which the value function $\widehat{Q}$ (or the critic) is used to estimate the policy gradient $\mathbf{E}_s \sum_a \nabla_w \pi(a|s;w)\widehat{Q}(s,a;\theta)$ [103, 3]. Note that it is standard to learn the value function model-free:

$$\theta \leftarrow \theta + \alpha\big(G_1 - \widehat{Q}(s_t, a_t; \theta)\big)\nabla_\theta \widehat{Q}(s_t, a_t, \theta) \ ,$$

where $G_1 := r_t + \widehat{Q}(s_{t+1}, a_{t+1}, \theta)$. Mnih et al. generalize this objective to a multi-step target $G_H := (\sum_{i=0}^{H-1} r_{t+i}) + \widehat{Q}(s_{t+h}, a_{t+h}; \theta)$.

In the model-free case, we compute G_H using the single trajectory observed during environmental interaction. However, because the policy is stochastic, G_H is a random variable with some variance. To reduce variance, I can use a learned model to generate an arbitrary number of rollouts (5 in our experiments), compute G_H for each rollout, and average them. We compared the effectiveness of both one-step and the multi-step models in generating useful rollouts for learning. To ensure meaningful comparison, for each algorithm, we perform the same number of value-function updates. We used three standard RL domains from Open AI Gym [30], namely Cart Pole, Acrobot, and Lunar Lander. Results are summarized in Figures 5.10 and 5.11.

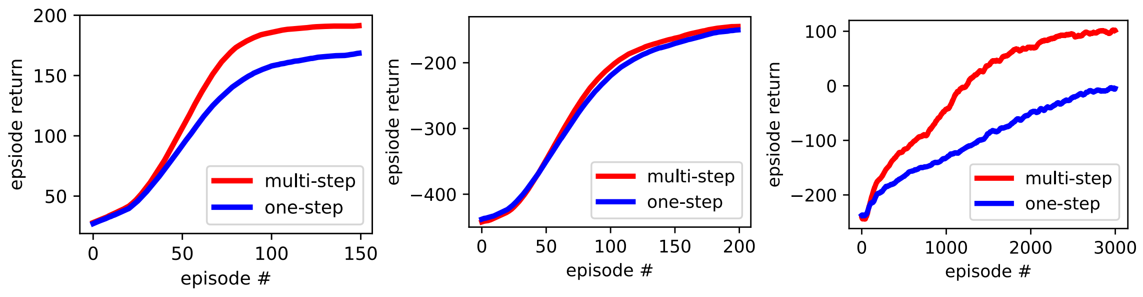


Figure 5.10: A comparison of actor critic equipped with the learned models (Cart Pole, Acrobot, and Lunar Lander). We set the maximum look-ahead horizon $H = 8$. Results are averaged over 100 runs, and higher is better. The multi-step model consistently matches or exceeds the one-step model.

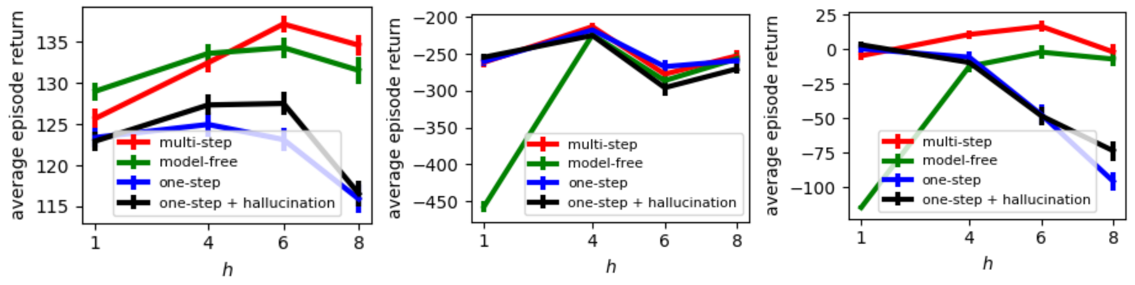


Figure 5.11: Area under the curve, which corresponds to average episode return, as a function of the look-ahead horizon h. Results for all three domains (Cart Pole, Acrobot, and Lunar Lander) are averaged over 100 runs. We add two additional baselines, namely the model-free critic, and a model-based critic trained with hallucination [107, 115]

.

5.8.2 Decision-time Planning

I now use the model for action selection. A common action-selection strategy is to choose $\arg\max_a \widehat{Q}(s, a)$, called the model-free strategy, hereafter. Our goal is to compare the utility of model-free strategy with its model-based counterparts. Our desire is to compare the effectiveness of the one-step model with the multi-step model in this scenario.

A key choice in decision-time planning is the strategy used to construct the tree. One approach is to expand the tree for each action in each observed state [14]. The main problem with this strategy is that the number of nodes grow exponentially. Alternatively, using a learned action-value function $\widehat{Q}$, at each state s we can only expand the most promising action $a^* := \arg\max_a \widehat{Q}(s, a)$. Clearly, given the same amount of computation, the second strategy can benefit from performing deeper look aheads. The two strategies are illustrated in Figure 5.12 (left).

Note that because the model is trained from experience, it is still only accurate up to a certain depth. Therefore, when we reach a specified planning horizon, H, we simply use $\max_a \widehat{Q}(s_H, a)$ as

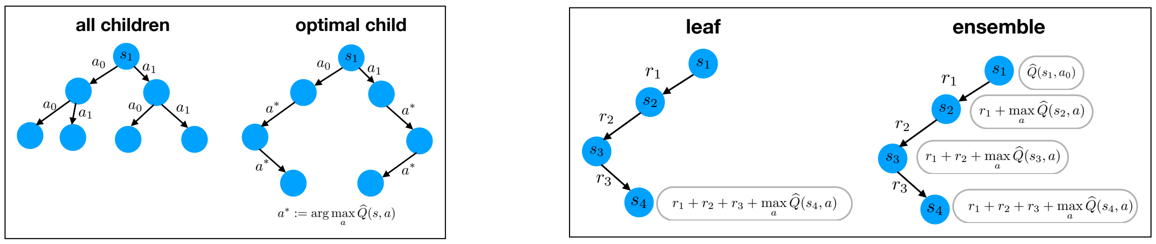

all children
optimal child
$a^* := \arg\max_a \widehat{Q}(s,a)$
leaf
ensemble
$r_1 + r_2 + r_3 + \max_a \widehat{Q}(s_4,a)$
$\widehat{Q}(s_1,a_0)$
$r_1 + \max_a \widehat{Q}(s_2,a)$
$r_1 + r_2 + \max_a \widehat{Q}(s_3,a)$
$r_1 + r_2 + r_3 + \max_a \widehat{Q}(s_4,a)$

Figure 5.12: Tree construction (left) and action-value estimation (right) strategies.

an estimate of future sum of rewards from the leaf node s_H. While this estimate can be erroneous, I observed that it is necessary to consider, because otherwise the agent will be myopic in the sense that it only looks at the short-term effects of its actions.

The second question is how to determine the best action given the built tree. One possibility is to add all rewards to the value of the leaf node, and go with the action that maximizes this number. As shown in Figure 5.12 (right), another idea is to use an ensemble where the final value of the action is computed using the mean of the H different estimates along the rollout. This idea is based on the notion that in machine learning averaging many estimates can often lead to a better estimate than the individual ones [90, 34].
The two tree-expansion strategies, and the two action-value estimation strategies together constitute four possible combinations. To find the most effective combination, I first performed an experiment in the Lunar Lander setting where, given different pretrained $\widehat{Q}$ functions, I computed the improvement that the model-based policy offers relative to the model-free policy. I trained these $\widehat{Q}$ function using the DQN algorithm [66] and stored weights every 100 episodes, giving us 20 snapshots of $\widehat{Q}$. The models were also trained using the same amount of data that a particular $\widehat{Q}$ was trained on. I then tested the four strategies (no learning was performed during testing). For each episode, I took the frozen $\widehat{Q}$ network of that episode, and compared the performance of different policies given $\widehat{Q}$ and the trained models. In this case, by performance I mean average episode return over 20 episodes.

Results, averaged over 200 runs, are presented in Figure 5.13 (left), and show an advantage for the ensemble and optimal-action combination (labeled optimal ensemble). Note that, in all four cases, the model used for tree search was the one-step model, and so this served as an experiment to find the best combination under this model. I then performed the same experiment with the multi-step model as shown in Figure 5.13 (right) using the best combination (i.e. optimal action expansion with ensemble value computation). We clearly see that M^3 is more useful in this scenario as well.

I further investigated whether the superiority in terms of action selection can actually accelerate DQN training as well. In this scenario, we ran DQN under different policies, namely model-free, model-based with the one-step model, and model-based with M^3. In all cases, we chose a random action with probability $\epsilon = 0.01$ for exploration. See Figure 5.14, which again shows the benefit of

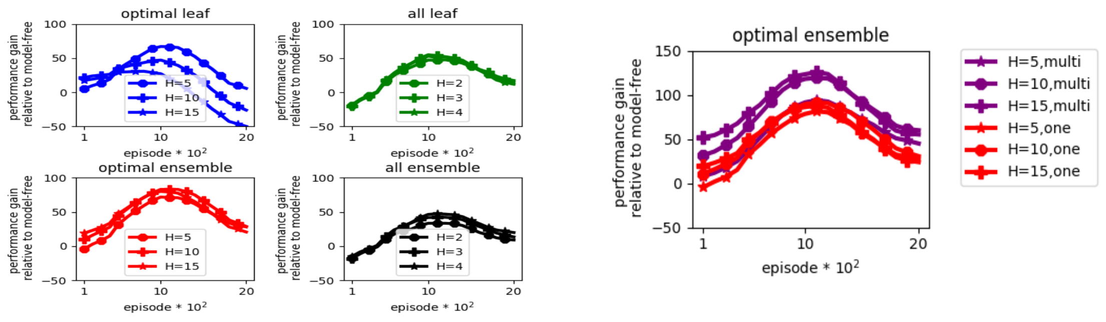
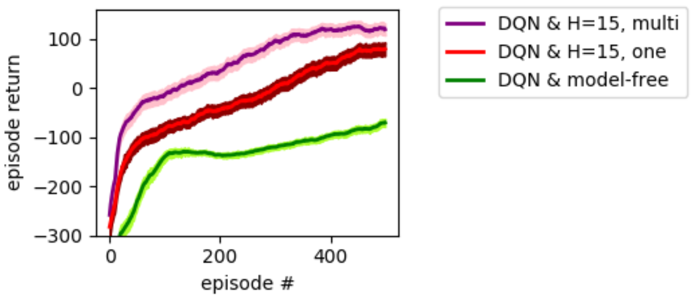


Figure 5.13: A comparison between tree expansion and value-estimation strategies when using the one-step model for action selection (left). Comparison between the one-step model and M^3 for action selection (right). x-axis denotes the $\widehat{Q}$ of agent at that episode, and y-axis denotes performance gain over model-free. Performance is defined as episode return averaged over 20 episodes. Note the inverted-U. Initially, $\widehat{Q}$ and the model are both bad, so model provides little benefit. Towards the end $\widehat{Q}$ gets better, so using the model is not beneficial. However, we get a clear benefit in the intermediate episodes because the model is faster to learn than $\widehat{Q}$.

Figure 5.14: A comparison between the two models in the context of model-based RL. Action selection with the multi-step model can significantly boost sample efficiency of DQN. All models are trained online from the agent's experience. Results are averaged over 100 runs, and shaded regions denote standard errors.



the multi-step model for decision-time planning in model-based RL.

5.9 Conclusion

In this chapter, I took an important step towards understanding the effects of smoothness in model-based RL. I showed that Lipschitz continuity of an estimated model plays a central role in multi-step prediction error, and in value-estimation error. I also showed the benefits of employing Wasserstein for model-based RL. I introduced a multi-step model, and showed its superiority for long-horizon planning when compared with one-step models.

Bibliography

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the international conference on Machine learning (ICML)*, 2006.

[2] Alejandro Agostini and Enric Celaya. Reinforcement learning with a gaussian mixture model. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, 2010.

[3] Cameron Allen, Kavosh Asadi, Melrose Roderick, Abdel-rahman Mohamed, George Konidaris, and Michael Littman. Mean actor critic. *arXiv preprint arXiv:1709.00503*, 2017.

[4] Brandon Amos, Lei Xu, and J Zico Kolter. Input convex neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[5] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[6] Kavosh Asadi, Evan Cater, Dipendra Misra, and Michael L. Littman. Equivalence between Wasserstein and value-aware model-based reinforcement learning. In *ICML workshop on Prediction and Generative Modeling in Reinforcement Learning*, 2018.

[7] Kavosh Asadi and Michael L. Littman. An alternative softmax operator for reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[8] Kavosh Asadi, Dipendra Misra, Seungchan Kim, and Michel L Littman. Combating the compounding-error problem with a multi-step model. *arXiv preprint arXiv:1905.13320*, 2019.

[9] Kavosh Asadi, Dipendra Misra, and Michael L. Littman. Lipschitz continuity in model-based reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[10] Kavosh Asadi, Ronald E Parr, George D Konidaris, and Michael L Littman. Deep radial-basis value functions for continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[11] Kavosh Asadi and Jason D Williams. Sample-efficient deep reinforcement learning for dialog control. *arXiv preprint arXiv:1612.06000*, 2016.

[12] Kavosh Asadi Atui. *Strengths, Weaknesses, and Combinations of Model-based and Model-free Reinforcement Learning.* PhD thesis, University of Alberta, 2015.

[13] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[14] Kamyar Azizzadenesheli, Brandon Yang, Weitang Liu, Emma Brunskill, Zachary C. Lipton, and Animashree Anandkumar. Sample-efficient deep RL with generative adversarial tree search. *CoRR*, abs/1806.05780, 2018.

[15] J Andrew Bagnell and Jeff G Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2001.

[16] Leemon C Baird and A Harry Klopf. Reinforcement learning with high-dimensional, continuous actions. 1993. Unpublished.

[17] Chris L Baker, Joshua B Tenenbaum, and Rebecca R Saxe. Goal inference as inverse planning. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, 2007.

[18] Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 2002.

[19] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 1983.

[20] Gleb Beliakov, Humberto Bustince Sola, and Tomasa Calvo Sánchez. *A Practical Guide to Averaging Functions.* Springer, 2016.

[21] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[22] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 1952.

[23] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 1957.

[24] Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

[25] Dimitri Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 1975.

[26] Dimitri Bertsekas and John Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995, Proceedings of the 34th IEEE Conference on*, 1995.

[27] S.P. Boyd and L. Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[28] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[29] Richard P Brent. *Algorithms for minimization without derivatives.* Courier Corporation, 2013.

[30] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[31] David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *Complex Systems*, 1988.

[32] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12(May):1655–1695, 2011.

[33] Guido Bugmann. Normalized Gaussian radial basis function networks. *Neurocomputing*, 1998.

[34] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *ICML*, 2004.

[35] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[36] T.M. Cover and J.A. Thomas. *Elements of Information Theory.* John Wiley and Sons, 2006.

[37] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian Q-learning. In *Proceedings of AAAI Conference on Artificial Intelligence*, 1998.

[38] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 1977.

[39] Omar Darwiche Domingues, Pierre Ménard, Matteo Pirotta, Emilie Kaufmann, and Michal Valko. Regret bounds for kernel-based reinforcement learning. *arXiv preprint arXiv:2004.05599*, 2020.

[40] Amir-Massoud Farahmand, Andre Barreto, and Daniel Nikovski. Value-Aware Loss Function for Model-based Reinforcement Learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[41] Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[42] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2016.

[43] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[44] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[45] Geoffrey J. Gordon. Reinforcement learning with function approximation converges to a region, 2001. Unpublished.

[46] Omer Gottesman, Fredrik Johansson, Matthieu Komorowski, Aldo Faisal, David Sontag, Finale Doshi-Velez, and Leo Anthony Celi. Guidelines for reinforcement learning in healthcare. *Nat Med*, 2019.

[47] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2016.

[48] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[49] Karl Hinderer. Lipschitz continuity of value functions in Markovian decision processes. *Mathematical Methods of Operations Research*, 2005.

[50] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 1989.

[51] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2015.

[52] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 1996.

[53] Sham Kakade, Michael J Kearns, and John Langford. Exploration in metric state spaces. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2003.

[54] Michael J. Kearns and Satinder P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

[55] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. Deepmellow: removing the need for a target network in deep q-learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[56] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, 2008.

[57] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, 2008.

[58] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[59] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2011.

[60] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

[61] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[62] Sungsu Lim, Ajin Joseph, Lei Le, Yangchen Pan, and Martha White. Actor-expert: A framework for using cction-value methods in continuous action spaces. *arXiv preprint arXiv:1810.09103*, 2018.

[63] Michael L. Littman and Csaba Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1996.

[64] Michael Lederman Littman. *Algorithms for Sequential Decision Making.* PhD thesis, Department of Computer Science, Brown University, 1996.

[65] Edward Lorenz. *Predictability: does the flap of a butterfly's wing in Brazil set off a tornado in Texas?* na, 1972.

[66] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[67] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

[68] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Learning multimodal transition dynamics for model-based reinforcement learning. *arXiv preprint arXiv:1705.00470*, 2017.

[69] John Moody and Christian J Darken. Fast learning in networks of locally-tuned processing units. *Neural computation*, 1989.

[70] Alfred Müller. Optimal selection from distributions with unknown parameters: Robustness of bayesian models. *Mathematical Methods of Operations Research*, 1996.

[71] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *arXiv preprint arXiv:1702.08892*, 2017.

[72] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. In *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[73] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A unified view of entropy-regularized Markov decision processes. *arXiv preprint arXiv:1705.07798*, 2017.

[74] Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-based capacity control in neural networks. In *Proceedings of the Conference on Learning Theory (COLT)*, 2015.

[75] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the international conference on Machine learning (ICML)*, 2008.

[76] Jason Pazis and Ron Parr. Generalized value functions for large action sets. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.

[77] Jason Pazis and Ronald Parr. Pac optimal exploration in continuous space markov decision processes. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2013.

[78] Jan Peters, Katharina Mülling, and Yasemin Altun. Relative entropy policy search. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2010.

[79] Bernardo Ávila Pires and Csaba Szepesvári. Policy error bounds for model-based reinforcement learning with factored linear models. In *Conference on Learning Theory (COLT)*, 2016.

[80] Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Policy gradient in lipschitz Markov decision processes. *Machine Learning*, 2015.

[81] Michael JD Powell. Radial basis functions for multivariable interpolation: A review. *Algorithms for Approximation*, 1987.

[82] Doina Precup and Richard S Sutton. Multi-time models for temporally abstract planning. In *Advances in neural information processing systems (NIPS)*, 1998.

[83] Martin L Puterman. Markov decision processes.: Discrete stochastic dynamic programming, 2014.

[84] Emmanuel Rachelson and Michail G. Lagoudakis. On the locality of action domination in sequential decision making. In *International Symposium on Artificial Intelligence and Mathematics*, 2010.

[85] Jonathan Rubin, Ohad Shamir, and Naftali Tishby. Trading value and information in mdps. In *Decision Making with Imperfect Decision Makers*. Springer, 2012.

[86] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[87] Stuart J Russell and Peter Norvig. Artificial intelligence: A modern approach, 1995.

[88] Moonkyung Ryu, Yinlam Chow, Ross Anderson, Christian Tjandraatmadja, and Craig Boutilier. CAQL: continuous action q-learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[89] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017.

[90] Robert E Schapire. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*. Springer, 2003.

[91] David Silver and Kamil Ciosek. Compositional planning using optimal option models. *arXiv preprint arXiv:1206.6473*, 2012.

[92] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

[93] David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, et al. The predictron: End-to-end learning and planning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.

[94] Sean R Sinclair, Siddhartha Banerjee, and Christina Lee Yu. Adaptive discretization for episodic reinforcement learning in metric spaces. In *Proceedings of the ACM Conference on Measurement and Analysis of Computing Systems*, 2019.

[95] Satinder Singh, Tommi Jaakkola, Michael L. Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 2000.

[96] Zhao Song, Ron Parr, and Lawrence Carin. Revisiting the softmax bellman operator: new benefits and new perspectives. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

[97] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 2009.

[98] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC model-free reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2006.

[99] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the International Conference on Machine Learning (ICML)*, 1990.

[100] Richard S Sutton. TD models: Modeling the world at a mixture of time scales. *Machine Learning*, 1995.

[101] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 1996.

[102] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[103] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NIPS)*, 2000.

[104] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 1999.

[105] Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael H. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the Conference in Uncertainty in Artificial Intelligence (UAI)*, 2008.

[106] Csaba Szepesvári. Algorithms for reinforcement learning, 2010.

[107] Erik Talvitie. Model regularization for stable sample rollouts. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

[108] Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

[109] Sebastian B. Thrun. The role of exploration in learning control. In *Handbook of Intelligent Control: Neural, Fuzzy, and Adaptive Approaches*. 1992.

[110] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society.*, 1996.

[111] Emanuel Todorov. Linearly-solvable markov decision problems. In *Proceedings of Advances in Neural Information Processing (NIPS)*, 2006.

[112] Ahmed Touati, Adrien Ali Taiga, and Marc G Bellemare. Zooming for efficient model-free reinforcement learning in metric spaces. *arXiv preprint arXiv:2003.04069*, 2020.

[113] Harm van Seijen and Rich Sutton. A deeper look at planning as learning from replay. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[114] Harm Van Seijen, Hado Van Hasselt, Shimon Whiteson, and Marco Wiering. A theoretical and empirical analysis of Expected Sarsa. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2009.

[115] Arun Venkatraman, Martial Hebert, and Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[116] Christopher Watkins. Learning from delayed rewards. *King's College, Cambridge*, 1989.

[117] Christopher Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[118] Jason D Williams, Asadi Kavosh, and Geoffrey Zweig. Hybrid code networks: practical and efficient end-to-end dialog control with supervised and reinforcement learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

[119] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.