Strengths, Weaknesses, and Combinations of Model-based and Model-free Reinforcement Learning

by

Kavosh Asadi Atui

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Reinforcement learning algorithms are conventionally divided into two approaches: a model-based approach that builds a model of the environment and then computes a value function from the model, and a model-free approach that directly estimates the value function. The first contribution of this thesis is to demonstrate that, with similar computational resources, neither approach dominates the other. Explicitly, the model-based approach achieves a better performance with fewer environmental interactions, while the model-free approach reaches a more accurate solution asymptotically by using a larger representation or eligibility traces. The strengths offered by each approach are important for a reinforcement learning agent and, therefore, it is desirable to search for a combination of the two approaches and get the strengths of both. The main contribution of this thesis is to propose a new architecture in which a model-based algorithm forms an initial value function estimate and a model-free algorithm adds on to and improves the initial value function estimate. Experiments show that our architecture, called the Cascade Architecture, preserves the data efficiency of the model-based algorithm. Moreover, we prove that the Cascade Architecture converges to the original model-free solution and thus prevents any imperfect model from impairing the asymptotic performance. These results strengthen the case for combining model-based and model-free reinforcement learning.

I learned the value of hard work by working hard.

– Margaret Mead.

Acknowledgements

First and foremost I want to thank my supervisor, Professor Richard Sutton, for being a great mentor. I specially thank Rich for showing me how to develop my own thoughts, and also for being plain and direct about my numerous weaknesses. To date, I consider reading his book earlier in my undergraduate studies as the best decision I have made in my academic career.

Next, I want to thank Joseph Modayil for his extreme amount of help and advice on every single piece of this thesis. As a skeptical person, I always had many questions for Joseph, and he always had good answers ready for every single one of the questions.

My sincere thanks goes also to the members of the RLAI lab. The insightful discussions I constantly had with Rupam, Marlos, Mohammad, Adam, and Pooria were valuable for my research. Moreover, they provided a friendly and warm environment for me, which made it easier to survive Edmonton's cold weather!

I finally want to thank my father Alireza. He literally does everything possible to help me succeed in my academic career. I am very lucky to have him as my father, and I hope I will also be able to serve as a great father in future and for his grandchildren.

I alone am responsible for all mistakes in this thesis.

Table of Contents

List of Figures

Chapter 1

Introduction

Reinforcement learning algorithms are conventionally divided into two approaches (Sutton and Barto, 1998). The first approach builds an internal model of the world from the agent's experience and then estimates the value function with the built model. This approach is referred to as the model-based approach. The second conventional approach directly goes from experience to a value function estimate. It does not construct a model and, therefore, is called the model-free approach. Figure 1.1 illustrates the difference between the internal structure of these two approaches.

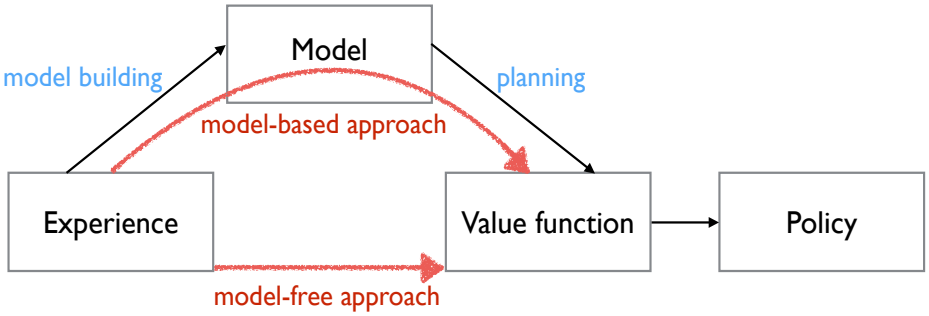


Figure 1.1: The structure of the two reinforcement learning approaches. The model-based approach estimates the value function by taking the indirect path of model construction followed by planning, while the model-free approach directly estimates the value function from experience.

A classically known advantage of the indirect model-based approach is that it often finds a good value function estimate with fewer environmental interactions and provides better interim performance (Sutton, 1990; Moore and Atkeson, 1993; Atkeson and Santamaria, 1997). This advantage, which

is formally called *data efficiency*, is offered by thinking about the outcome of the agent's actions in the imaginary experience generated by a model. Data efficiency is particularly valuable in settings where computational resources are abundant, but agent's actual experience is costly and should be used carefully.

The main benefit of the model-free approach is its computational efficiency. Due to a cheap computational demand, a model-free algorithm can usually support a representation larger than that of a model-based algorithm when the two algorithms are given the same amount of computation. Moreover, model-free algorithms can often use the idea of eligibility traces in a straightforward fashion. With a larger representation and using the eligibility traces, the model-free approach can perform better asymptotically.

The first contribution of this thesis is to establish that neither approach can dominate the other. Implicitly, a trade-off between the interim and the asymptotic advantages arises when choosing between these two approaches: the model-based approach can rapidly improve its behaviour and provide a better interim performance, while the model-free approach achieves a better asymptotic performance by using a larger representation or eligibility traces.

This trade-off raises a natural and important question: How can we usefully and flexibly combine these two approaches to exploit the strengths and to eliminate the weaknesses of each individual approach? The reinforcement learning literature has investigated this question and proposed a number of combinations which will be covered later in this thesis. The combination of the indirect model-based approach and the direct model-free approach has recently become important for neuroscientists as well, mainly because of the evidence suggesting that human and animal brains might employ a combination of the two approaches for control of behaviour (Daw et al., 2005).

The main contribution of this thesis is to develop a new architecture that flexibly and usefully combines the two approaches. In this architecture the model-based approach estimates an initial value function oblivious to the model-free approach. The model-free approach takes, in parallel, the initial

value function estimate, adds to it a weight vector, and adapts the weight vector from the agent's actual experience to form an improved value function estimate. This final value function estimate is then used to control the agent's behaviour. The idea behind our architecture, which we refer to it as the Cascade Architecture, can be used more generally to combine any number of parallel processes with a shared goal.

We assess the effectiveness of the Cascade Architecture by presenting empirical results on three reinforcement learning domains. We show that the Cascade combination of a model-based algorithm and a model-free algorithm preserves the interim advantage of the model-based algorithm and achieves an asymptotic performance similar to that of the model-free algorithm. We also prove that, in general, Cascade Architecture's asymptotic performance is same as the original model-free algorithm. This property of the Cascade Architecture sets the stage for using more extreme approximations to make smaller models, with which planning is cheaper, without any negative impact on the asymptotic solution.

The contributions of this thesis are:

- Showing the relative strengths and weaknesses of model-based and model-free reinforcement learning.
- Developing the Cascade Architecture as a way of combining model-based and model-free approaches.
- Assessing the effectiveness of the Cascade Architecture by presenting empirical and theoretical results.
- Showing a general equivalence between the linear TD(λ) solution and the linear β-model solution with a similar representation.

Chapter 2

Background

This chapter introduces the notation and reinforcement learning algorithms used in the rest of this thesis. The first four sections of this chapter, which discuss our notation and the idea of building a linear model, are particularly important because they only are discussed here and will be used in this thesis frequently. The subsequent sections review the family of temporal difference learning algorithms and Dyna-style planning with the linear model. These algorithms will be covered briefly in the next chapter and, therefore, are skippable for a reader familiar with them. For a more comprehensive background, we encourage the reader to study the standard reinforcement learning textbook (Sutton and Barto, 1998).

2.1 The Reinforcement Learning Problem

In the reinforcement learning problem an agent interacts with its environment in order to achieve a certain goal. The goal is to find a behaviour that achieves the highest cumulative reward in the environment.

Markov decision processes (MDP) is the standard mathematical formulation used for the reinforcement learning problem. An MDP is specified by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$. At each time step t the agent is in a state $S_t \in \mathcal{S}$ where $\mathcal{S}$ is the set of all possible states and $|\mathcal{S}| = N$ is the total number of states. The agent chooses an action $A_t \in \mathcal{A}$ which is responded by the environment with

a scalar reward signal R_{t+1}. The expected value of this signal is specified by:

$$\mathcal{R}_s^a = \mathbb{E}\Big\{ R_{t+1} \Big| S_t = s, A_t = a \Big\} \ . \tag{2.1}$$

Once the reward signal is provided, the agent is then moved to a next state $S_{t+1} \in \mathcal{S}$. The probability of this transition is determined by:

$$\mathcal{P}_{ss'}^a = \Pr(S_{t+1} = s' | S_t = s, A_t = a) \ . \tag{2.2}$$

Finally, γ is the discount rate representing the relative importance of future rewards compared to the importance of the immediate reward. The goal of the agent is to maximize discounted cumulative reward by finding and implementing a *policy* π. Formally, $\pi(a|s)$ gives the probability of taking a specific action a in a state s. Figure 2.1 diagrams the agent-environment interaction in the context of reinforcement learning.

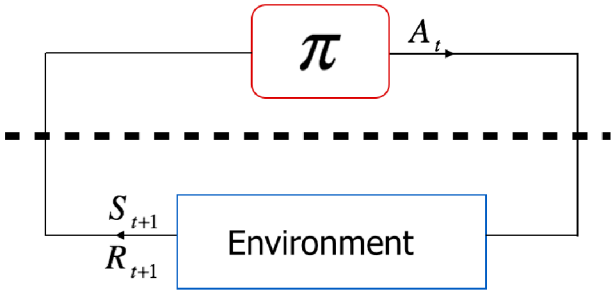


Figure 2.1: Agent-environment interaction in reinforcement learning.

We conclude this section by explaining the difference between reinforcement learning and planning. Typical planning algorithms take as input an internal *model* of the environment and search with the model for a value function or policy. In this context, a model is defined as anything that could be used by the agent to predict the outcome of its actions. To a reinforcement learning agent a model is not given a priori and, therefore, the agent should seek a good behaviour through interaction with the environment and by trial and error. This distinction arguably makes reinforcement learning a more general and challenging problem than planning.

2.2 Value Functions

All of the reinforcement learning algorithms considered in this thesis involve estimating *value functions*. Informally, value functions represent how good it is for the agent to be in a particular situation. In order to formally define value functions, we first need to define *return*. At each time step t we define return G_t as the discounted sum of future rewards:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \ . \tag{2.3}$$

For a specified policy π, we can now define the *state-value function* $v_\pi(s)$ as the function that maps a state s to a scalar denoting the expected return starting from state s and following policy π. More formally:

$$v_\pi(s) = \mathbb{E}_\pi\Big\{ G_t \Big| S_t = s \Big\} \ . \tag{2.4}$$

State-value function can be written in a recursive form based on the value of the successor states:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi\Big\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s \Big\} \\ &= \mathbb{E}_\pi\Big\{ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \Big| S_t = s \Big\} \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \left[\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \ \mathbb{E}_\pi\Big\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \Big| S_{t+1} = s' \Big\} \right] \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \Big[\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \ v_\pi(s') \Big] \ . \end{aligned} \tag{2.5}$$

It is also useful to define the value of an action in a particular state. Concretely, we define the *action-value function* $q_\pi(s,a)$ as the function that maps a state-action pair (s,a) to the expected return starting from state s, taking action a, and following policy π afterwards:

$$q_\pi(s,a) = \mathbb{E}_\pi\Big\{ G_t \Big| S_t = s, A_t = a \Big\} \ . \tag{2.6}$$

Similarly, it is possible to write the action-value function recursively:

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a'\in\mathcal{A}} q_\pi(s',a') \ . \tag{2.7}$$

Bellman equations, such as the equations (2.5) and (2.7), are a core part of most reinforcement learning algorithms.

2.3 Function Approximation

Algorithms that estimate the value functions using equations (2.5) and (2.7) and with a table with one entry for each state are called tabular algorithms. Such algorithms require $\mathcal{O}(N)$ memory to store an estimate of the state-value function. This memory demand is impractical in an application with a large state space. The second and the more critical issue with the tabular setting is the lack of generalization. In a large-scale problem, the agent encounters a new state most of the time and, without generalizing from previously experienced states, it will be unable to behave well. Due to these issues, we move beyond the tabular algorithms.

An appealing answer to the issues presented in the tabular setting is the idea of function approximation. The idea is to approximate a target function by selecting a function, among a specific class of functions, that best matches the target function. In the approximate setting, a state s is represented by a feature vector $\boldsymbol{\phi}(s)$ with n features where the number of features is typically much smaller than the number of states. In other words $n \ll N$. Similarly, a state-action pair can be represented by a feature vector $\boldsymbol{\phi}(s, a)$.

A particular class of functions offering a sound theoretical foundation is the class of linear functions. As we will show in the next two sections, linear function approximation is commonly used to approximate the underlying dynamics of MDPs ($\mathcal{P}$ and $\mathcal{R}$) and value functions.

2.4 Model Building with Linear Function Approximation

Constructing a model of the environment can be an expensive process. For instance, estimating $\{\mathcal{R}, \mathcal{P}\}$ without using any form of approximation requires $\mathcal{O}(N^2|\mathcal{A}|)$ of memory. Moreover, planning with such a model requires a

large amount of computational resources. We, therefore, use linear function approximation for constructing an approximate model of the environment. In this case a column vector and a matrix are learned for every possible action: a vector $\mathbf{b}_a$ called the reward model, and a matrix $\mathbf{F}_a$ called the transition model. One way to construct this model is by solving for the minimum of the two least-squares objective functions defined as:

$$\mathbf{b}^a = \arg\min_{\mathbf{b}} \sum_s d_\pi(s)\Big(\mathcal{R}^a_s - \mathbf{b}^\top \boldsymbol{\phi}(s)\Big)^2 \qquad \forall a \in \mathcal{A} \qquad \text{and} \tag{2.8}$$

$$\mathbf{F}^a = \arg\min_{\mathbf{F}} \sum_s d_\pi(s)\Big\| \sum_{s'} \mathcal{P}^a_{ss'}\ \boldsymbol{\phi}(s') - \mathbf{F}\boldsymbol{\phi}(s)\Big\|^2 \qquad \forall a \in \mathcal{A}\ . \tag{2.9}$$

In equations (2.8) and (2.9), $d_\pi(s)$ is called the stationary distribution of the state s under a policy π.

The set $\{\mathbf{b}^a, \mathbf{F}^a |\ \forall a \in \mathcal{A}\}$ denotes the standard one-step linear model (Parr et al., 2008; Sutton et al., 2008). In the scope of this thesis, this model is simply referred to as the linear model. It is possible to incrementally construct the linear model by a stochastic gradient descent method (Sutton et al., 2008) using estimates of the gradient of the objective functions (2.8) and (2.9). This leads to the updates:

$$\mathbf{b}^a_{t+1} \leftarrow \mathbf{b}^a_t + \alpha\Big(R_{t+1} - \mathbf{b}^{a\top}_t \boldsymbol{\phi}(S_t)\Big)\boldsymbol{\phi}(S_t) \qquad \text{for } a = A_t \qquad \text{and} \tag{2.10}$$

$$\mathbf{F}^a_{t+1} \leftarrow \mathbf{F}^a_t + \alpha\Big(\boldsymbol{\phi}(S_{t+1}) - \mathbf{F}^a_t \boldsymbol{\phi}(S_t)\Big)\boldsymbol{\phi}(S_t)^\top \qquad \text{for } a = A_t\ , \tag{2.11}$$

where α is called the step-size parameter. For an easier analysis, it is useful to write the minimization problems (2.8) and (2.9) in matrix format. Let $\mathbf{P}^a$ be an $N \times N$ matrix where $\mathbf{P}^a_{ss'} = \mathcal{P}^a_{ss'}$ and let $\mathbf{r}_a$ be a column vector with N components where $\mathbf{r}^a_s = \mathcal{R}^a_s$ (treating the subscripts as their indices for vectors and matrices). Assuming a uniform stationary distribution, the minimization problems (2.8) and (2.9) can be written as:

$$\mathbf{b}^a = \arg\min_{\mathbf{b}} ||\boldsymbol{\Phi}\mathbf{b} - \mathbf{r}^a||^2 \qquad \forall a \in \mathcal{A} \qquad \text{and} \tag{2.12}$$

$$\mathbf{F}^a = \arg\min_{\mathbf{F}} ||\boldsymbol{\Phi}\mathbf{F}^\top - \mathbf{P}^a\boldsymbol{\Phi}||^2 \qquad \forall a \in \mathcal{A}\ , \tag{2.13}$$

where the thin matrix $\mathbf{\Phi} \in \mathcal{R}^{N \times n}$ is called the *representation matrix*. The rows of the representation matrix correspond to the agent's states, and the columns of this matrix are the features.

There are many ways in which the linear model can be useful. In section 2.6 we focus on a particular style of planning with the linear model.

2.5 Model-free Reinforcement Learning with Linear Function Approximation

In the linear setting, the state-value function is approximated by a linear function of the features:

$$v_\pi(s) \approx \hat{v}(s) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s) = \sum_{i=1}^{n} \boldsymbol{\theta}_i \ \boldsymbol{\phi}_i(s) \ . \tag{2.14}$$

Sometimes the goal is to estimate the state-value function for a fixed policy, rather than finding a good policy. This is called the *prediction* setting. In the prediction setting, the family of temporal difference learning (TD) algorithms (Sutton, 1988) are often used to directly learn the weight vector $\boldsymbol{\theta}$ from the agent's experience. Linear TD(0) is the simplest member of this family of algorithms with the update:

$$\begin{aligned} &\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\Big(U_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t)\Big)\boldsymbol{\phi}(S_t) \ , \\ &\text{where } U_t = R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_{t+1}) \ , \end{aligned} \tag{2.15}$$

and where α is a step size parameter. Also, U_t is called the update target at time step t.

TD algorithms are also used for the full reinforcement learning problem, the *control* setting, in which the goal is to find the best policy. It is typical for a model-free control algorithm to estimate the action-value function, rather than the state-value function. In the linear setting, the action-value function is approximated by:

$$q_\pi(s,a) \approx \hat{q}(s,a) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s,a) = \sum_{i=1}^{n} \boldsymbol{\theta}_i \ \boldsymbol{\phi}_i(s,a) \ . \tag{2.16}$$

Linear Sarsa(0) (Rummery and Niranjan, 1994) is the simplest member of the family of TD control algorithms with the update:

$$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha\Big(U_t - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t, A_t)\Big)\boldsymbol{\phi}(S_t, A_t) \ , \tag{2.17}$$

$$\text{where } U_t = R_{t+1} + \gamma \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_{t+1}, A_{t+1}) \ . \tag{2.18}$$

Algorithm 1 gives a complete code for linear Sarsa(0) with a ϵ-greedy exploration policy.

Algorithm 1: linear Sarsa(0) with ϵ-greedy policy

Initialize $\boldsymbol{\theta}$
Obtain initial state S
$A \leftarrow \arg\max_a \Big\{\boldsymbol{\theta}^\top \boldsymbol{\phi}(S, a)\Big\}$ with prob. $(1-\epsilon)$ else from $\mathcal{A}$ at random
Take action A
for each time step **do**
  Receive next reward R and next state S'
  $A' \leftarrow \arg\max_a \Big\{\boldsymbol{\theta}^\top \boldsymbol{\phi}(S', a)\Big\}$ with Pr. $(1-\epsilon)$ else from $\mathcal{A}$ at random
  Take action A'
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\Big(R + \gamma \boldsymbol{\theta}^\top \boldsymbol{\phi}(S', A') - \boldsymbol{\theta}^\top \boldsymbol{\phi}(S, A)\Big)\boldsymbol{\phi}(S, A)$
  $S \leftarrow S'$
  $A \leftarrow A'$
end

Linear TD(0) and linear Sarsa(0) are considered to be computationally efficient, as their per-time-step complexity scales linearly with the number of features[1].

2.6 Dyna-style Planning with the Linear Model

The core of Dyna-style planning with the linear model (Sutton et al., 2008) is to apply linear TD(0) to the imaginary experience generated by the linear model. More precisely, this form of planning involves the update:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\mathrm{U_P} - \boldsymbol{\theta}^\top \boldsymbol{\phi}_P)\boldsymbol{\phi}_P \ , \tag{2.19}$$

$$\text{where } \mathrm{U_P} = \mathbf{b}^{a\top} \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}_P \ .$$

[1] In the scope of this thesis, we assume that the number of action are relatively small.

Here $\boldsymbol{\phi}_P$ is an arbitrary feature vector chosen by the planner and a is an action drawn from the agent's policy. Moreover, $\mathrm{U_P}$ is called the planning update target. Notice the similarity between this update rule and that of linear TD(0) presented by (2.15). There are, nevertheless, two distinctions between Dyna-style planning with the linear model and linear TD(0). First and foremost, the source of the experience used here to define the planning update target $\mathrm{U_P}$ is the linear model, as opposed to the agent's actual experience. Second, $\boldsymbol{\phi}_P$ is an arbitrary feature vector chosen by a planner, as opposed to the linear TD(0) update where the feature vector is related to the current state. At each time-step, the planner can choose one or more feature vectors arbitrarily and use them in the update rule (2.19).

It has been shown that choosing a good distribution for planning feature vectors potentially increases the speed of convergence (Sutton et al., 2008). As we will see in the next subsection, useful strategies have been proposed for choosing the planning feature vector. But, we stick to a general and arbitrary strategy at this point and introduce our own strategy in the upcoming chapter.

The particular model-based algorithm we regularly use in this thesis is the one that continually adapts the linear model and uses Dyna-style planning to estimate the state-value function. The complete code of this algorithm is given by Algorithm 2. Throughout this thesis, we refer to this model-based algorithm as control with the linear model.

Algorithm 2: control with the linear model and ϵ-greedy policy

Initialize $\boldsymbol{\theta}$ and $\{\mathbf{F}^a, \mathbf{b}^a | \forall a \in \mathcal{A}\}$
Obtain initial state S
for each time step **do**
  $A \leftarrow \arg\max_a \{\mathbf{b}_a^\top \boldsymbol{\phi}(S) + \gamma \boldsymbol{\theta}^\top \mathbf{F}_a \boldsymbol{\phi}(S)\}$ with Pr. 1-ϵ, else from $\mathcal{A}$ at random
  Take action A, receive R and S'
  $\mathbf{F}^a \leftarrow \mathbf{F}^a + \alpha\Big(\boldsymbol{\phi}(S') - \mathbf{F}^a \boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S)^\top \qquad$ for $a = A$
  $\mathbf{b}^a \leftarrow \mathbf{b}^a + \alpha\Big(R - \mathbf{b}^{a\top} \boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S) \qquad$ for $a = A$
  for p times for planning **do**
    choose $\boldsymbol{\phi}_P$ according to an arbitrary distribution μ
    $a \leftarrow \arg\max_a \{\mathbf{b}^{a\top} \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}_P\}$ with Pr. 1-ϵ, else from $\mathcal{A}$ at random
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha(\mathbf{b}^{a\top} \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}_P - \boldsymbol{\theta}^\top \boldsymbol{\phi}_P)\boldsymbol{\phi}_P$
  end
  $S \leftarrow S'$
end

Notice that the per-time-step complexity of control with the linear model is $\mathcal{O}(n^2)$ where n is the number of features.

2.6.1 Prioritized Sweeping

The idea of prioritized sweeping is to frequently choose those planning feature vectors that will result in a high change in the value function estimate. In the tabular case, this can be achieved by a model and by going backwards from the states that have recently changed in value to the states that potentially lead to them (Moore and Atkeson, 1993; Peng and Williams, 1996). In the function approximation setting, though, it is not possible to go backwards from individual states, but going backwards feature by feature is indeed possible (Sutton et al., 2008). More precisely, if there has been a large update in $\boldsymbol{\theta}_i$, the component with index i of the weight vector $\boldsymbol{\theta}$, then a feature j for which $\mathbf{F}_{ij}$ is large can be a good candidate for planning. Thus, $\mathbf{e}_j$, the unit basis vector of dimension j, will be selected as planning feature vector. Prior work discussed the variations of this idea, as well as their implementations, and showed its effectiveness in practice (Sutton et al., 2008).

2.7 Conclusion

This chapter covered the notation used in this thesis. It then introduced the two families of algorithms: the model-free family of temporal difference learning algorithms and the model-based family of algorithms that continually learn and plan with the linear model. In the next chapter we compare these two families of algorithms from a theoretical and empirical point of view.

Chapter 3

Relative Strengths and Weaknesses of Model-based and Model-free Approaches

This chapter investigates the strengths and weaknesses of model-based and model-free reinforcement learning as one of our main contributions. So far, this thesis has highlighted the difference between the two approaches in terms of their structure. Due to this difference, we naturally expect the two approaches to perform differently as well. This chapter identifies the strengths and weaknesses of each approach through an empirical and theoretical comparison. Recognizing the strengths and weaknesses presented here is crucial for reading the next chapters in which we combine the two approaches.

Model-based and model-free approaches are dissimilar in terms of their paths to the value function. The model-based approach is indirect, in the sense that it takes the indirect path of building a model and planning. In this context, by a model we mean any entity employed to predict the outcome of actions in various situations. Moreover, by planning we strictly mean going from a model to a value function estimate. In contrast, the model-free approach directly uses experience to update the value function estimate rather than going through a model. Due to this difference, each approach has its own strengths and weaknesses and has proven to be effective on different sets of applications.

The most prominent success of model-based reinforcement learning in recent years is, perhaps, in the domain of autonomous helicopter flight (Ng et al.,

2004). In this work a pilot was asked to fly a helicopter in order to provide sufficient experience for building a model of the environment. Having built the model, the agent used a planning algorithm to accomplish four acrobatic maneuvers. This work is regarded as the first one to autonomously complete these maneuvers. In this work model building, planning, and decision making happened sequentially, rather than simultaneously. Therefore, managing computational resources was not considered an important issue.

Model-free reinforcement learning has recently been successful as well, most notably in the framework of Atari 2600 games (Bellemare et al., 2012). In a recent work (Mnih et al., 2015), a model-free algorithm with a non-linear function approximator was used to play the games with a representation that took as input raw pixels only. This work showed that the model-free algorithm can outperform a human player on a subset of games. Unlike the autonomous helicopter domain, in the Atari domain the agent had access to an extensive amount of experience.

In this chapter, we will investigate the strengths and weaknesses of each approach. We will show that, consistent with the classical view, the model-based approach is data efficient and enables a better performance with limited environmental interactions. We then show that, through using a larger representation and eligibility traces, the model-free approach can provide better asymptotic performance.

3.1 Representative Family of Algorithms

Reinforcement learning has recently become a popular field and, therefore, various algorithms have been proposed lately. For the sake of an easier comparison, though, we consider one family of algorithms from each of the two approaches. The two representative families, while being simple, are among the fundamental reinforcement learning algorithms and are commonly used in practice. The next two sections proceed by presenting these representative families which have already been presented thoroughly in the

background chapter.

3.1.1 Representative Model-based Family

Our model-based representative uses experience to continually adapt the linear model and simultaneously implements Dyna-style planning to go from the linear model to a linear state-value function estimate. At each time-step planning takes a number of steps. This is denoted by p and is referred to as the number of planning steps. For the prediction setting, the code for this model-based algorithm is shown by Algorithm 3. We refer to this algorithm simply as prediction with the linear model.

Algorithm 3: prediction with the linear model

Initialize $\boldsymbol{\theta}$ and $\{\mathbf{F}, \mathbf{b}\}$, and obtain initial S
for each time step **do**
  Receive R, S'
  $\mathbf{F} \leftarrow \mathbf{F} + \alpha\Big(\boldsymbol{\phi}(S') - \mathbf{F}\boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S)^\top$
  $\mathbf{b} \leftarrow \mathbf{b} + \alpha\Big(R - \mathbf{b}^\top\boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S)$
  for p times for planning **do**
    for i from 1 to n **do**
      $\boldsymbol{\phi}_P \leftarrow \mathbf{e}_i$
      $\boldsymbol{\theta} \leftarrow \mathbf{b}^\top\boldsymbol{\phi}_P + \gamma\boldsymbol{\theta}^\top\mathbf{F}\boldsymbol{\phi}_P$
    end
  end
  $S \leftarrow S'$
end

As mentioned in the background chapter, Dyna-style planning involves a strategy for choosing the planning feature vector $\boldsymbol{\phi}_P$. For the empirical results presented in this chapter, we choose $\boldsymbol{\phi}_P$ in a specific manner. At each time-step, we iterate over every dimension $i \in \{1, ..., n\}$ and set $\boldsymbol{\phi}_P$ to be the unit basis vector $\mathbf{e}_i$ of that dimension. Finally, we choose the number of planning steps p to be 1, indicating that we iterate over every dimension once at each time-step.

Prediction with the linear model can easily be extended to the control setting. The control extension adapts a vector $\mathbf{b}^a$ and a matrix $\mathbf{F}^a$ for each action,

representing the approximate reward and transition model of the environment. The algorithm uses this model, along with the state-value function estimate, to find and then implement a policy. The complete algorithm, which we refer to as control with the linear model, is presented by Algorithm 4.

Algorithm 4: control with the linear model and ϵ-greedy policy

Initialize $\boldsymbol{\theta}$ and $\{\mathbf{F}^a, \mathbf{b}^a | \ \forall a \in \mathcal{A}\}$, and obtain initial S
for each time step **do**
  $A \leftarrow \arg\max_a\{\mathbf{b}^{a\top}\boldsymbol{\phi}(S) + \gamma\boldsymbol{\theta}^\top\mathbf{F}^a\boldsymbol{\phi}(S)\}$ with Pr. 1-ϵ, else from $\mathcal{A}$ at random
  Take action A, receive R, S'
  $\mathbf{F}^a \leftarrow \mathbf{F}^a + \alpha\Big(\boldsymbol{\phi}(S') - \mathbf{F}^a\boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S)^\top \qquad$ for $a = A$
  $\mathbf{b}^a \leftarrow \mathbf{b}^a + \alpha\Big(R - \mathbf{b}^{a\top}\boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S) \qquad$ for $a = A$
  for p times for planning **do**
    for i from 1 to n **do**
      $\boldsymbol{\phi}_P \leftarrow \mathbf{e}_i$
      $a \leftarrow \arg\max_a\{\mathbf{b}^{a\top}\boldsymbol{\phi}_P + \gamma\boldsymbol{\theta}^\top\mathbf{F}^a\boldsymbol{\phi}_P\}$ with Pr. 1-ϵ, else from $\mathcal{A}$ at random
      $\boldsymbol{\theta} \leftarrow \mathbf{b}^{a\top}\boldsymbol{\phi}_P + \gamma\boldsymbol{\theta}^\top\mathbf{F}^a\boldsymbol{\phi}_P$
    end
  end
  $S \leftarrow S'$
end

The computational complexity of this family of algorithms is $\mathcal{O}(n^2)$.

3.1.2 Representative Model-free Family

We use the family of TD algorithms as our representative model-free algorithm (Sutton, 1988). TD algorithms are, perhaps, the most fundamental family of reinforcement learning algorithms. Moreover, they are computationally cheap as they scale linearly with the number of features. Linear TD(0) is the simplest form of this family in the linear function approximation case. The code for this algorithm is given by Algorithm 5.

Algorithm 5: linear TD(0)

Initialize $\boldsymbol{\theta}$ and obtain initial state S
for each time step **do**
  Receive next reward R and next state S'
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\Big(R + \gamma\boldsymbol{\theta}^\top\boldsymbol{\phi}(S') - \boldsymbol{\theta}^\top\boldsymbol{\phi}(S)\Big)\boldsymbol{\phi}(S)$
  $S \leftarrow S'$
end

Linear Sarsa(0) is the extension of linear TD(0) to the control setting. The main distinction is that linear Sarsa(0) estimates the action-value function, rather than the state-value function. The code for linear Sarsa(0) is given by Algorithm 6.

Algorithm 6: linear Sarsa(0) with ϵ-greedy policy

Initialize $\boldsymbol{\theta}$ and obtain initial state S
$A \leftarrow \arg\max_a\{\boldsymbol{\theta}^\top\boldsymbol{\phi}(S,a)\}$ with Pr. $(1-\epsilon)$ else from $\mathcal{A}$ at random
Take action A
for each time step **do**
  Receive next reward R and next state S'
  $A' \leftarrow \arg\max_a\{\boldsymbol{\theta}^\top\boldsymbol{\phi}(S',a)\}$ with Pr. $(1-\epsilon)$ else from $\mathcal{A}$ at random
  Take action A'
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\Big(R + \gamma\boldsymbol{\theta}^\top\boldsymbol{\phi}(S',A') - \boldsymbol{\theta}^\top\boldsymbol{\phi}(S,A)\Big)\boldsymbol{\phi}(S,A)$
  $S \leftarrow S'$
  $A \leftarrow A'$
end

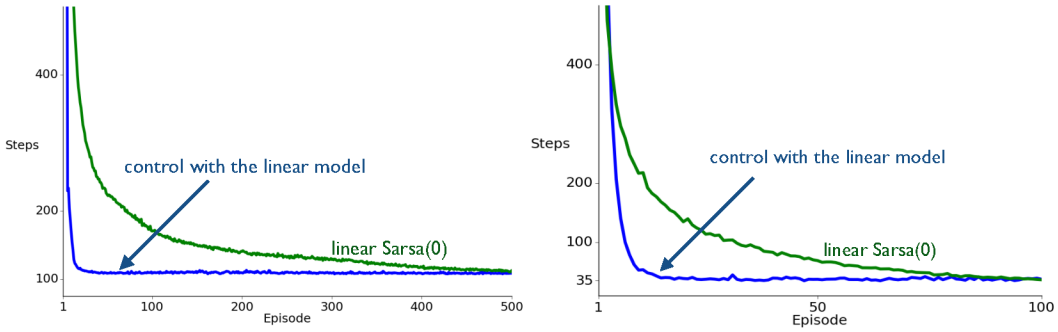
3.2 Model-based Approach is More Data efficient

The main advantage of the model-based approach is thought to be data efficiency. This approach is able to leverage experience by using it to build a model of the environment, rather than discarding the experience. Interesting theoretical results on the data efficiency of the two approaches have been found by prior work (Kakade, 2003; Strehl et al., 2009; Szita and Szepesvári, 2010). The latest work (Szita and Szepesvári, 2010) has shown similar upper bounds on the number of exploratory actions required for Q-learning, a model-free algorithm, and R-MAX, a model-based algorithm, before they find

a near-optimal policy. Yet the scope of all of those results are restricted to the tabular case; and in the approximate setting many of their assumptions are not true anymore. Also the unique result in the function approximation setting (De Farias and Van Roy, 2004) is based on the assumption of knowing a near optimal sampling policy, which is what we are interested to find in the first place.

We rely on empirical results, rather than theoretical results, to perform the comparison. Prior work on empirical comparison of the two approaches dates back to the comparison in the tabular setting (Sutton, 1990; Moore and Atkeson, 1993; Atkeson and Santamaria, 1997). One of the first comparisons in the linear setting (Boyan, 2002), presented LSTD as a model-based algorithm and showed its interim benefit over linear TD algorithms, though the scope of this work was restricted to the prediction setting. The most recent work in the linear setting (Sutton et al., 2008) performed a comparison between a model-free algorithm and Dyna Architecture which could be thought of as a combination of model-based and model-free approaches. The empirical results presented in this work showed a weak advantage for the Dyna Architecture over the linear TD algorithms. We aim to conduct more experiments in order to reach to a stronger result.

We ran two control experiments on standard reinforcement learning domains, namely mountain car and puddle world. The details of both domains are explained in (Sutton, 1996). In the mountain car domain, the initial position was chosen in an interval between -0.7 and -0.5 with a uniform distribution. Also the initial velocity was zero. The representation consisted of 10 tilings each with 100 features resulting in a representation with 1000 binary features. In the puddle world domain the initial state was the tuple (0.25,0.6) and the representation consisted of 8 tilings each with 64 features. Therefore, the representation consisted of 512 features. We set ϵ to zero and, instead, initialized the value function estimate optimistically and to vectors of zero in order to handle the exploration problem. Step-sizes were optimized for the best asymptotic performance over 500 independent runs.



(a) mountain car (b) puddle world

Figure 3.1: Comparison between linear Sarsa(0) and control with the linear model given a similar representation. We report the average performance over 500 independent runs. In both domains, the model-based algorithm outperforms the model-free algorithm in the interim, while they achieve the same asymptotic performance.

The experiments show that control with the linear model can outperform linear Sarsa(0) in terms of the interim performance. These results are consistent with the classical view on the model-based approach by showing its data efficiency. This strength is particularly valuable in situations where computation is abundant, but experience is costly.

A second observation is the similar asymptotic performance of the two algorithms. It has been shown that prediction with the linear model leads to an asymptotic solution equivalent to the linear TD(0) solution (Parr et al., 2008; Sutton et al., 2008). We next move to a review of this result, as we will generalize this result later in this chapter.

For a fixed policy π let the matrix $\mathbf{P} \in \mathcal{R}^{N \times N}$ contain the state-transition probabilities:

$$\mathbf{P}_{ss'} = \Pr(S_{t+1} = s' | S_t = s, \pi) \ . \tag{3.1}$$

Notice that this matrix incorporates both the transitions dynamics of the MDP as well as the fixed policy. Similarly, we define a column vector $\mathbf{r} \in \mathcal{R}^N$ where:

$$\mathbf{r}_s = \mathbb{E}_\pi \Big\{ R_{t+1} \Big| S_t = s \Big\} \ . \tag{3.2}$$

For an easier analysis, we assume that the stationary distribution is uniform, though the results are extendable to the more general case with an arbitrary

stationary distribution. We define a column vector $\mathbf{v} \in \mathcal{R}^N$ where:

$$\mathbf{v}_s = v_\pi(s) \tag{3.3}$$

We finally define a thin matrix $\mathbf{\Phi} \in \mathcal{R}^{N \times n}$ where $\mathbf{\Phi}_{si}$ indicates the value of feature i in the state s. We also assume that the columns of $\mathbf{\Phi}$ are linearly independent. With this notation and using linear function approximation, the vector $\mathbf{v}$ can be approximated by $\mathbf{v} \approx \mathbf{\Phi}\boldsymbol{\theta}$.

We are now ready to look for the linear model solution achieved by prediction with the linear model. The algorithm incrementally constructs the linear model $\{\mathbf{b}, \mathbf{F}\}$ by minimizing the objective functions:

$$\mathbf{b} = \arg\min_{\mathbf{b}} ||\mathbf{\Phi}\mathbf{b} - \mathbf{r}||^2 \qquad \text{and} \tag{3.4}$$

$$\mathbf{F} = \arg\min_{\mathbf{F}} ||\mathbf{\Phi}\mathbf{F}^\top - \mathbf{P}\mathbf{\Phi}||^2 \ . \tag{3.5}$$

The global minimum of these two minimization problems can be found by taking the derivative of the objective functions with respect to the variables and setting them to zero. This leads to the solutions:

$$\mathbf{b} = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{r} \qquad \text{and} \tag{3.6}$$

$$\mathbf{F} = \Big((\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{P}\mathbf{\Phi}\Big)^\top \ . \tag{3.7}$$

Given the linear model, the algorithm looks for a $\boldsymbol{\theta}$ that satisfies:

$$\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Phi}\mathbf{b} + \gamma\mathbf{\Phi}\mathbf{F}^\top\boldsymbol{\theta} \ . \tag{3.8}$$

Since both sides of the equation are in the span of $\mathbf{\Phi}$, there exists a $\boldsymbol{\theta}$ that exactly satisfies equation (3.8):

$$\begin{aligned}
\mathbf{\Phi}\boldsymbol{\theta} &= \mathbf{\Phi}\mathbf{b} + \gamma\mathbf{\Phi}\mathbf{F}^\top\boldsymbol{\theta} \\
\boldsymbol{\theta} &= \mathbf{b} + \gamma\mathbf{F}^\top\boldsymbol{\theta} && \text{by linearly independent columns of } \mathbf{\Phi} \\
\boldsymbol{\theta} &= (\mathbf{I} - \gamma\mathbf{F}^\top)^{-1}\mathbf{b} \\
\boldsymbol{\theta} &= \Big(\mathbf{I} - \gamma(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{P}\mathbf{\Phi}\Big)^{-1}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{r} && \text{by (3.6) and (3.7)} \\
\boldsymbol{\theta} &= \Big(\mathbf{\Phi}^\top\mathbf{\Phi} - \gamma\mathbf{\Phi}^\top\mathbf{P}\mathbf{\Phi}\Big)^{-1}\mathbf{\Phi}^\top\mathbf{r} && \text{by } \mathbf{A}^{-1}\mathbf{B}^{-1} = (\mathbf{B}\mathbf{A})^{-1}
\end{aligned} \tag{3.9}$$

We next move to finding the asymptotic solution of linear TD(0). This model-free algorithm directly looks for a $\boldsymbol{\theta}$ that satisfies:

$$\boldsymbol{\Phi\theta} = \boldsymbol{\Pi_\Phi}(\mathbf{r} + \gamma \mathbf{P\Phi\theta}) \ , \tag{3.10}$$

where $\boldsymbol{\Pi_\Phi} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top$ is the least-squares projection into the span of $\boldsymbol{\Phi}$. With a couple of clear steps shown below, the solution found by linear TD(0) is:

$$\begin{aligned}
&\boldsymbol{\Phi\theta} = \boldsymbol{\Pi_\Phi}(\mathbf{r} + \gamma \mathbf{P\Phi\theta}) \\
&\boldsymbol{\Phi\theta} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} + \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\gamma\mathbf{P\Phi\theta} \qquad \text{by } \boldsymbol{\Pi_\Phi} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top \\
&\boldsymbol{\Phi\theta} - \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\gamma\mathbf{P\Phi\theta} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} \\
&\boldsymbol{\theta} - (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\gamma\mathbf{P\Phi\theta} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} \qquad \text{by linearly independent columns of } \boldsymbol{\Phi} \\
&\Big(\mathbf{I} - (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\gamma\mathbf{P\Phi}\Big)\boldsymbol{\theta} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} \\
&\boldsymbol{\theta} = \Big(\mathbf{I} - (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\gamma\mathbf{P\Phi}\Big)^{-1}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} \\
&\boldsymbol{\theta} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \gamma\boldsymbol{\Phi}^\top\mathbf{P\Phi})^{-1}\boldsymbol{\Phi}^\top\mathbf{r} \qquad \text{by } \mathbf{A}^{-1}\mathbf{B}^{-1} = (\mathbf{BA})^{-1} \ .
\end{aligned} \tag{3.11}$$

Note that the solution (3.11) is equal to the solution (3.9), meaning that the linear TD(0) solution is equivalent to the linear model solution. We consider this a positive result for the model-based algorithm, because the design bias of having the linear model will not yield an asymptotic solution worse than that of linear TD(0).

The experimental results presented here supported the general view that the model-based algorithm is more data-efficient. Moreover, we showed that the model-based algorithm is as good as the model-free algorithm at asymptote. Nevertheless, this analysis did not take into account the different computational demands of the two algorithms, as if computation is not a concern. In the next section we control for the computation and repeat the experiments.

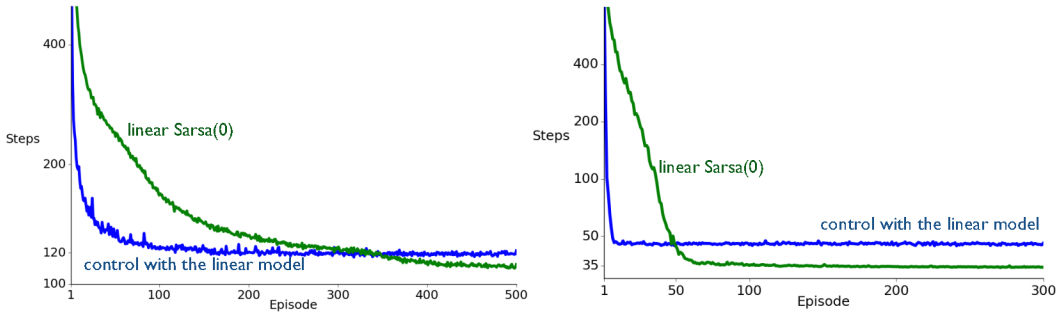
3.3 Model-free Approach Performs Better at Asymptote

The assumption of having an agent with infinite computational resources is rather unrealistic. There, normally, is a bound on the agent's computational resources, specially in a large-scale application with a fast and high dimensional stream of data. Ideally, the limited computational resources available to the agent should influence the agent's approach to a problem.

Model-based algorithms often require a larger amount of computation compared to their model-free counterparts. Our representative model-based algorithm have the per-time-step complexity of $\mathcal{O}(n^2)$ where n denotes the number of features. In contrast, typical model-free algorithms, such as linear TD(0) and linear Sarsa(0), scale linearly with the number of features of the representation. Due to this cheap computational demand, model-free algorithms can support a relatively larger representation when the same computational resources are available. As such, a more meaningful comparison would be the one that gives the same amount of computation to both of the algorithms.

We, therefore, repeat the two experiments presented in the previous section, this time with a much smaller representation for the model-based algorithm. In the mountain car domain, the model-free representation consists of 10 tilings each with 225 features and the model-based representation uses only 2 tilings with the same number of features. In the puddle world domain, the model-free algorithm uses 10 tilings each with 64 features while the model-based algorithm uses only 1 tiling with 64 features. These representations led the two algorithms to take approximately the same amount of running time on our machine. The results are shown in Figure 3.2 of this chapter.

In these results control with the linear model is still performing better on the interim. But, linear Sarsa(0) is now able to use a richer representation and search in a subspace that is closer to the actual value function. This leads the model-free algorithm to achieve a superior asymptotic performance.



(a) mountain car (b) puddle world

Figure 3.2: Comparison between linear Sarsa(0) and control with the linear model when same computational resources are given to both of the algorithms. In both domains, the model-based algorithm outperforms the model-free algorithm on the early episodes, but reaches to a poor asymptotic performance. The results are for the optimal parameters and are averaged over 500 runs.

These results actually conclude that, with similar computational resources, a model-free algorithm can perform better at asymptote. As the model-based algorithm is still able to find a better policy with limited environmental interactions, neither approach can outperform the other and there specifically is a trade-off between the interim and the asymptotic advantages.

3.4 Model-free Approach is More Compatible with Eligibility Traces

In this section, we consider the impact of eligibility traces when the same representation is used for both approaches. Eligibility traces are one of the core ideas in reinforcement learning. TD(λ) use eligibility traces to bridge the gap between TD(0), a full *bootstrapping* algorithm, and monte carlo methods that do not bootstrap. Here by bootstrapping we mean updating the value function estimate based on the estimate of the value of other states or state-action pairs. Linear TD(λ) proceeds with the update:

$$\begin{aligned} \boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t + \alpha\Big(R_{t+1} + \gamma\boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_{t+1}) - \boldsymbol{\theta}_t^\top \boldsymbol{\phi}(S_t)\Big)\mathbf{z}_t \ , \\ \mathbf{z}_t &\leftarrow \gamma\lambda\mathbf{z}_{t-1} + \boldsymbol{\phi}(S_t) \ . \end{aligned} \tag{3.12}$$

The vector $\mathbf{z}_t$ is called the accumulating eligibility trace at time-step t. Other forms of traces are also common and can be helpful in different situations.

It has been shown that there is a bound on the quality of the solution found by linear TD(λ). Interestingly, the bound becomes tighter as λ gets closer to 1. (Bertsekas and Tsitsiklis, 1996) Previously, we showed that the solution of prediction with the (one-step) linear model is equivalent to the linear TD(0) solution. We aim to extend the theoretical equivalence to general λ.

We prove that the solution found by linear TD(λ) is identical to the linear β-model solution. β-model (Sutton, 1995) is a mixture of multi-step models in which the weighting falls off exponentially with time:

$$\mathbf{r}_\beta = \sum_{t=0}^{\infty} (\gamma\beta\mathbf{P})^t \, \mathbf{r} = (\mathbf{I} - \gamma\beta\mathbf{P})^{-1}\mathbf{r} \qquad \text{and} \tag{3.13}$$

$$\mathbf{P}_\beta = (1-\beta)\sum_{t=0}^{\infty} (\gamma\beta\mathbf{P})^t \, \mathbf{P} \; . \tag{3.14}$$

Notice that, with $\beta = 0$, $\mathbf{r}_\beta$ and $\mathbf{P}_\beta$ will reduce to $\mathbf{r}$ and $\mathbf{P}$. A linear β-model can be learned by minimizing the following objective functions:

$$\mathbf{b}_\beta = \arg\min_{\mathbf{b}} ||\mathbf{\Phi b} - \mathbf{r}_\beta|| = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{r}_\beta \qquad \text{and} \tag{3.15}$$

$$\mathbf{F}_\beta = \arg\min_{\mathbf{F}} \; ||\mathbf{\Phi F}^\top - \mathbf{P}_\beta\mathbf{\Phi}|| = \Big((\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{P}_\beta\mathbf{\Phi}\Big)^\top \tag{3.16}$$

Given this model, linear β-model solution is found by finding a $\boldsymbol{\theta}$ that satisfies $\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Phi b}_\beta + \gamma\mathbf{\Phi F}_\beta^\top\boldsymbol{\theta}$. This leads to the solution:

$$\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Phi b}_\beta + \gamma\mathbf{\Phi F}_\beta^\top\boldsymbol{\theta} \tag{3.17}$$

$$\boldsymbol{\theta} = \mathbf{b}_\beta + \gamma\mathbf{F}_\beta^\top\boldsymbol{\theta} \tag{3.18}$$

$$\boldsymbol{\theta} = (\mathbf{I} - \gamma\mathbf{F}_\beta^\top)^{-1}\mathbf{b}_\beta \tag{3.19}$$

$$\boldsymbol{\theta} = (\mathbf{I} - \gamma(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{P}_\beta\mathbf{\Phi})^{-1}(\mathbf{\Phi}^\top\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{r}_\beta \qquad \text{by (3.15) and (3.16)} \tag{3.20}$$

$$\boldsymbol{\theta} = (\mathbf{\Phi}^\top\mathbf{\Phi} - \gamma\mathbf{\Phi}^\top\mathbf{P}_\beta\mathbf{\Phi})^{-1}\mathbf{\Phi}^\top\mathbf{r}_\beta \; . \qquad \text{by } \mathbf{A}^{-1}\mathbf{B}^{-1} = (\mathbf{BA})^{-1} \tag{3.21}$$

We now look for the linear TD(λ) solution. The algorithm solves for the fixed point of:

$$\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Pi}_{\mathbf{\Phi}}\Big(\mathcal{T}^\lambda(\mathbf{\Phi}\boldsymbol{\theta})\Big) \; , \tag{3.22}$$

$$\text{where } \mathcal{T}^\lambda(\mathbf{\Phi}\boldsymbol{\theta}) = \underbrace{(\mathbf{I} - \gamma\lambda\mathbf{P})^{-1}\mathbf{r}}_{\mathbf{r}_\lambda} + \gamma \underbrace{(1-\lambda)\sum_{t=0}^{\infty}(\gamma\lambda\mathbf{P})^t \, \mathbf{P}}_{\mathbf{P}_\lambda}\mathbf{\Phi}\boldsymbol{\theta} \; . \tag{3.23}$$

Note that, with $\lambda = 0$, the operator $\mathcal{T}$ will reduce to $\mathbf{r} + \gamma \mathbf{P} \mathbf{\Phi} \boldsymbol{\theta}$. The solution can then be found with a couple of steps:

$$\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Pi}_{\mathbf{\Phi}}(\mathbf{r}_\lambda + \gamma \mathbf{P}_\lambda \mathbf{\Phi}\boldsymbol{\theta}) \tag{3.24}$$

$$\mathbf{\Phi}\boldsymbol{\theta} = \mathbf{\Phi}(\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top (\mathbf{r}_\lambda + \gamma \mathbf{P}_\lambda \mathbf{\Phi}\boldsymbol{\theta}) \tag{3.25}$$

$$\boldsymbol{\theta} = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top (\mathbf{r}_\lambda + \gamma \mathbf{P}_\lambda \mathbf{\Phi}\boldsymbol{\theta}) \tag{3.26}$$

$$\boldsymbol{\theta} - \gamma (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{P}_\lambda \mathbf{\Phi}\boldsymbol{\theta} = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{r}_\lambda \tag{3.27}$$

$$\boldsymbol{\theta} = \left(\mathbf{I} - \gamma (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{P}_\lambda \mathbf{\Phi}\right)^{-1} (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{r}_\lambda \tag{3.28}$$

$$\boldsymbol{\theta} = (\mathbf{\Phi}^\top \mathbf{\Phi} - \gamma \mathbf{\Phi}^\top \mathbf{P}_\lambda \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{r}_\lambda \qquad \text{by } \mathbf{A}^{-1}\mathbf{B}^{-1} = (\mathbf{B}\mathbf{A})^{-1} \tag{3.29}$$

Setting λ equal to β in this solution will make it identical to solution (3.21). Thus, prediction with a linear β-model and linear TD(λ) yield the same solution when $\lambda = \beta$ and when the same representation is used. This result makes a more general statement: with a similar representation, there is no asymptotic benefit for linear TD(λ) compared to solving for the linear β-model.

In the control setting, the story is different. We show that, in the control setting, eligibility traces can provide substantial benefit for the model-free algorithm, while it is not the case for the model-based algorithm. In the control setting, building a useful multi-step model requires having some form of sub-goal (Sutton et al., 1999); yet automatically finding useful subgoals is an open question. On the other hand, model-free control algorithms can use eligibility traces in a straightforward fashion. One of these algorithms is linear Sarsa(λ) (Algorithm 7) which is the control extension of linear TD(λ). Although we lack a concrete theory at this point, using a λ close to 1 often leads to better empirical results. We move to a designed domain to show an example of such a situation.

We created a domain, referred to as the hallway domain, with 19 states and 4 actions (Up, Right, Down, Left) in each state. To introduce stochasticity, a chosen action will be executed only with probability 0.9, otherwise a random action will be executed. The reward signal is 1 upon the transition to the terminal state 18 and is 0 otherwise. There is also no discounting. The agent

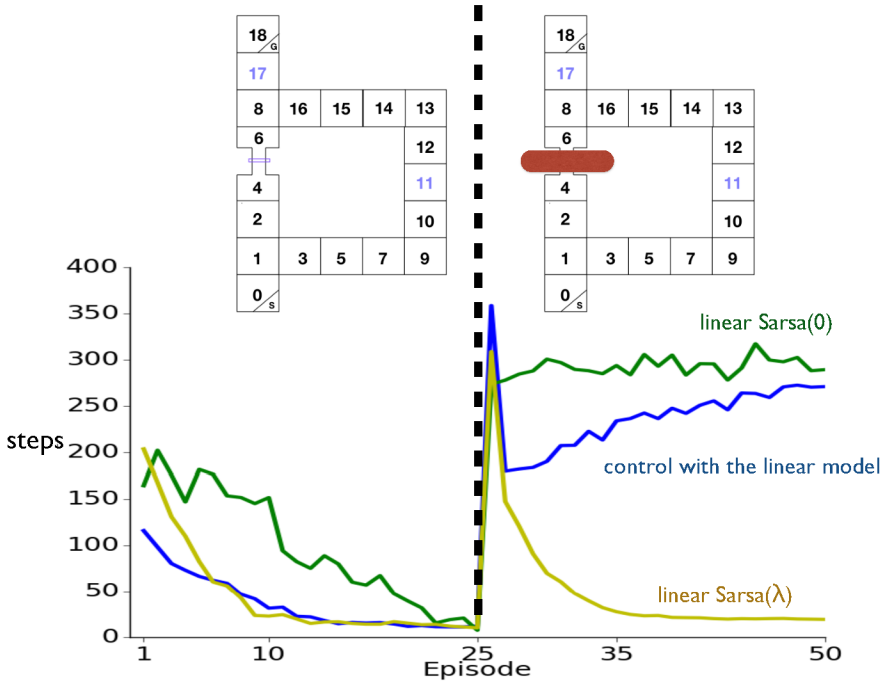


Figure 3.3: Comparison between linear Sarsa(0), linear Sarsa(λ), and control with the linear model on the hallway domain. After we block the shorter path, both one-step algorithms (linear Sarsa(0) and control with the linear model) fail to find the optimal policy, whereas linear Sarsa(λ) with $\lambda = 0.9$ can find the best policy.

starts from the state 0. The value function estimate has one entry for each state except for states 11 and 17 that are aggregated and share the same entry. After episode number 24 we block the path between the states 4 and 6. As such, the agent should use the longer path to the goal and encounter states 11 and 17.

Interestingly, linear Sarsa(λ) with $\lambda = 0.9$ is the only algorithm among the three algorithms that finds the optimal policy. The two single-step algorithms (linear Sarsa(0) and control with the linear model), fail to find the optimal policy. This result shows the last strength of model-free approach. It basically shows that, even when the same representation is used, model-free algorithms can perform better than the model-based algorithms due to using the eligibility traces.

Algorithm 7: linear Sarsa(λ) with accumulating traces and ϵ-greedy policy

Initialize $\boldsymbol{\theta}$ and $\mathbf{e}$, and obtain initial state S
$A \leftarrow \arg\max_a\{\boldsymbol{\theta}^\top \boldsymbol{\phi}(S,a)\}$ with prob. $(1-\epsilon)$ else from $\mathcal{A}$ at random
Take action A
for each time step **do**
  Receive next reward R and next state S'
  $A' \leftarrow \arg\max_a\{\boldsymbol{\theta}^\top \boldsymbol{\phi}(S',a)\}$ with prob. $(1-\epsilon)$ else from $\mathcal{A}$ at random
  Take action A'
  $\mathbf{e} \leftarrow \gamma\lambda\mathbf{e} + \boldsymbol{\phi}(S,A)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\Big(R + \gamma\boldsymbol{\theta}^\top\boldsymbol{\phi}(S',A') - \boldsymbol{\theta}^\top\boldsymbol{\phi}(S,A)\Big)\mathbf{e}$
  $S \leftarrow S'$
  $A \leftarrow A'$
end

3.5 Conclusion

The results presented in this chapter established an implicit trade-off when choosing between the indirect model-based approach and the direct model-free approach. Concretely, the model-based approach leads to better performance with limited environmental interactions, though it also is computationally more expensive. On the other hand, the model-free approach can use a larger representation when controlling for computational complexity and yields a better asymptotic performance. Additionally, the model-free approach is more compatible with eligibility traces, which is a powerful and useful idea in reinforcement learning. The drawback to the model-free approach, however, is that it requires more experience to reach a good performance.

In the next two chapters, our desire is to have the strengths of both approaches, if possible, by looking for the combinations of the two approaches.

Chapter 4

Existing Forms of Combination

We concluded the previous chapter by a desire to combine model-based and model-free approaches in order to have the merits of both of them. In this chapter we survey the prior work in reinforcement learning and neuroscience on combining the two approaches. Moreover, we discuss the limitations and advantages of the existing combinations. In light of the limitations presented here, in the next chapter we develop a more useful and flexible combination, the Cascade Architecture.

Previous work in reinforcement learning and neuroscience has explored ways to combine the indirect path of model building and planning with the direct path taken by model-free reinforcement learning. The goal of this research ranges from flexible planning in reinforcement learning (Silver et al., 2008) to understanding the mind in cognitive science and introducing the computational models that can account for psychological experiments (Daw et al., 2005; Gershman et al., 2014). From the computational point of view, though, the primary reason to combine indirect and direct paths is to exploit the strengths and to eliminate the weaknesses of each approach. In the next couple of sections, we list three forms of combination and discuss their advantages and limitations.

4.1 Dyna Architecture (Single Value Function Estimate)

One of the natural ways to combine the two approaches is to allow them to contribute to a single value function estimate. The Dyna Architecture (Sutton, 1990) was, perhaps, the first attempt to implement this idea. One of the fundamental elements of the Dyna Architecture is to characterize the two approaches as identical in a radical sense. Dyna's model-based process uses a reinforcement learning algorithm applied to the imaginary experience generated by a model, and Dyna's model-free process applies the same algorithm to the agent's actual experience and updates the same value function estimate. In other words, the only distinction between the two processes here is in the sources of experience that each of them employs. Figure 4.1 diagrams the Dyna Architecture.

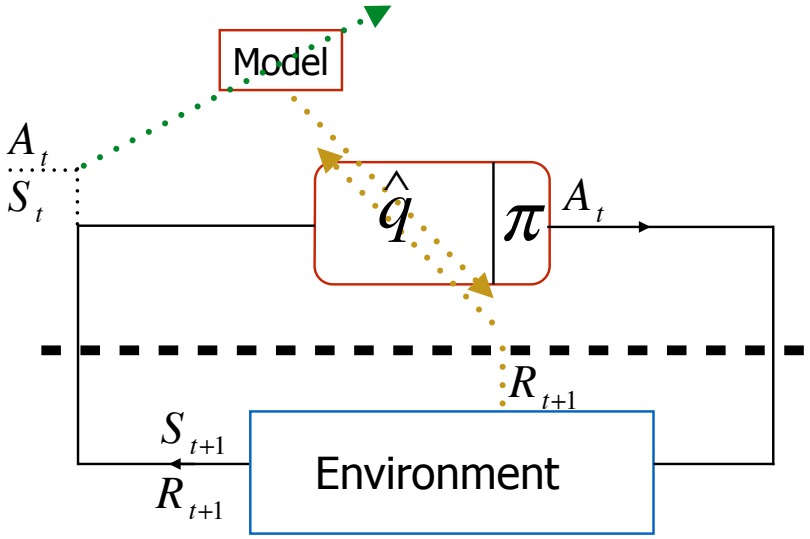


Figure 4.1: An illustration of the Dyna architecture. Both planner and model-free learner contribute to a single value function estimate.

A powerful attribute of the Dyna Architecture is to keep acting, learning, and planning independent while operating as fast as they can. They all run in parallel with none of them waiting for the other one. This flexibility is crucial as it enables the designer to allocate computational resources according to the

requirements. This flexibility has also allowed Dyna to explain psychological studies on human and animal decision-making. For example, it has been suggested (Gershman et al., 2014) that the brain uses, in a cooperative manner, both a goal-directed (model-based) approach as well as a habitual (model-free) approach to decision making and is able to flexibly allocate the computational resources based on the situation.

The structure of Dyna's value function estimate can be problematic for, at least, two reasons. First, imagine a situation where one approach leads to a better solution. Because both approaches contribute to the same value function estimate, the net solution can be worse than the solution of the preferred approach alone. In other words, there is no clear way to prevent the propagation of error and the negative impact of one approach on the other. Therefore, Dyna's model-free process may slow down the model-based process at the early stages of learning, while the model-based process may degrade the asymptotic solution found by the model-free process.

A second issue with a single value function estimate is the inability to support distinct representations, objective functions, and parameters. Recall that the model-free approach is computationally cheaper than the model-based approach when they both use the same representation. As such, model-free approach can support a larger representation than the model-based approach without increasing the computational complexity. The model-based process of linear Dyna algorithm (Sutton et al., 2008), for example, requires $\mathcal{O}(n^2)$ computation per-time-step, while the computational complexity of linear Dyna's model-free process scales linearly with the number of features. Therefore, the model-free process can support a representation with a significantly larger number of features without increasing the per-time-step complexity of linear Dyna. This flexibility is not offered by the Dyna Architecture. Similarly, it is unclear how to use eligibility traces within the Dyna Architecture, because the model-based and model-free processes will then have different objective functions while contributing to the same value function estimate.

4.2 Switching Between Separate Value Function Estimates

Given the limitations of the Dyna Architecture, it seems reasonable for each approach to update its exclusive value function estimate, a design choice we refer to as decoupling the value function estimates. Decoupling can provide the flexibility needed to support different representations, parameters, and objective functions. Moreover, decoupling prevents the negative effects of each approach on the other.

One way to use the decoupled value function estimates for decision making is to choose between the two value function estimates at each particular time-step and act based on the policy derived from the chosen value function estimate. Besides the two value function estimates, in this case the agent also requires to adapt a selector signal σ. Figure 4.2 illustrates this form of combination.

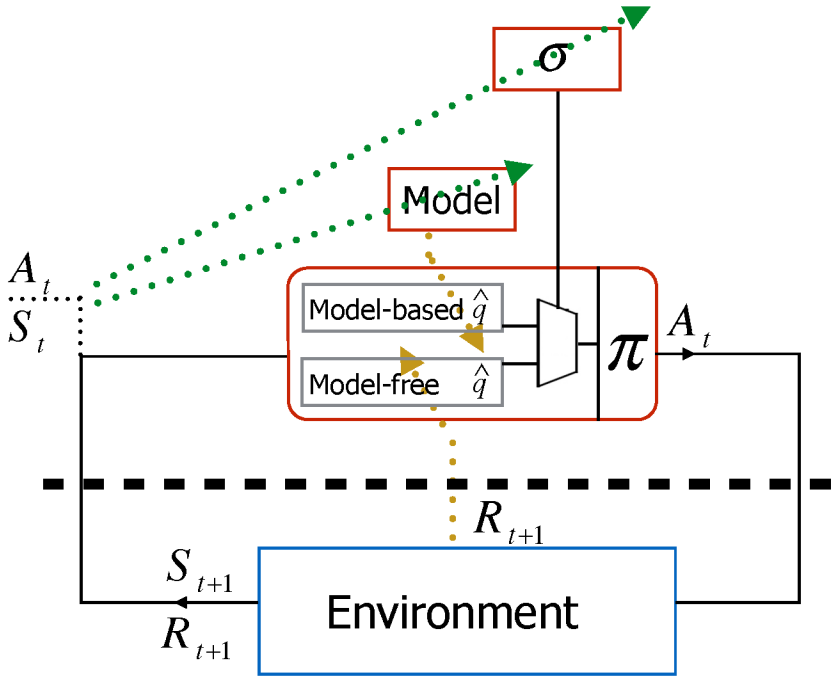


Figure 4.2: Switching between separate value function estimates. This form of combination involves decoupling the value function estimates and also adapting a selector signal that chooses, at each time step, the preferred value function estimate.

Prior work (Daw et al., 2005) has suggested a selector signal, specifically for the

tabular case, which is mainly based on the uncertainty in the estimation of the two value functions and choosing the one with a higher certainty. Nevertheless, choosing a selector signal is not obvious from the outset and it is difficult to find an effective selector signal for the general case. Moreover, adapting a good selector signal may, itself, require a large amount of computation and exacerbate the already challenging problem of resource allocation.

4.3 Weighted Average of Separate Value Function Estimates

Another form of combination is to decouple the value function estimates, but also to form a third value function estimate by taking a weighted average of the value function estimates from the two approaches. Let us assume that the direct model-free process forms its own value function estimate $\hat{q}_{\text{MF}}(s, a)$ and that the indirect model-based process estimates another value function $\hat{q}_{\text{MB}}(s, a)$. This form of combination computes a third value function estimate by taking a weighted average over $\hat{q}_{\text{MF}}(s, a)$ and $\hat{q}_{\text{MB}}(s, a)$. More precisely:

$$\hat{q}(s, a) = \frac{w^t_{\text{MB}}\ \hat{q}_{\text{MB}}(s, a) + w^t_{\text{MF}}\ \hat{q}_{\text{MF}}(s, a)}{w^t_{\text{MB}} + w^t_{\text{MF}}}\ . \tag{4.1}$$

This form of combination is illustrated by Figure 4.3.

One suggestion (Gläscher et al., 2010) for setting w^t_{MB} and w^t_{MF} is in a fixed schedule:

$$w^t_{\text{MB}} = e^{-kt} \qquad \text{and} \tag{4.2}$$

$$w^t_{\text{MF}} = 1 - e^{-kt}\ , \tag{4.3}$$

where k is a parameter by which the decay rate is tuned. In effect, this combination will softly switch from the model-based value function estimate to the model-free value function estimate. Choosing the weights in this way is intuitive, because we expect the model-based approach to perform better initially and the model-free approach to perform better asymptotically. But, this combination is then not fully extendable to non-stationary environments.

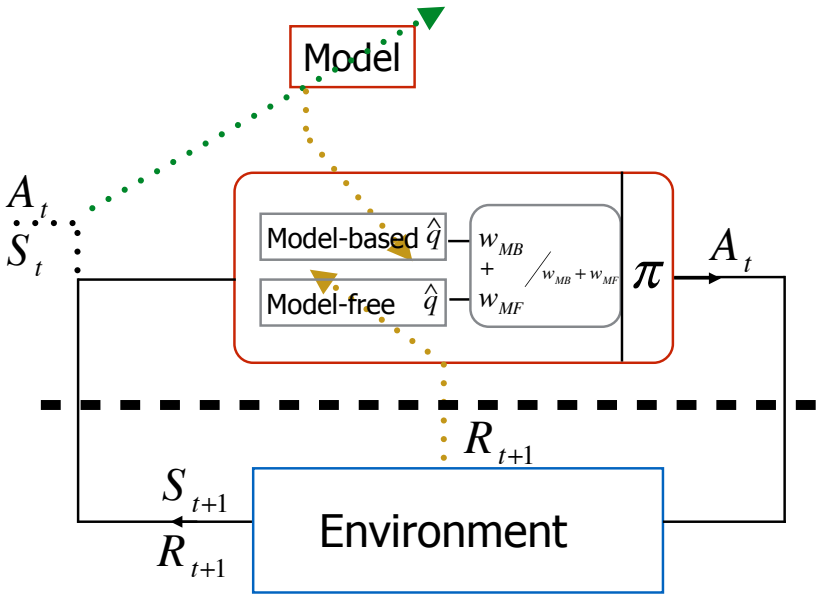


Figure 4.3: Weighted average of separate value function estimates.

In non-stationary domains, in which the benefits of planning are more noticeable, we require continual planning to rapidly adapt to the changes in the environment.

4.4 Conclusion

We presented three ideas for combining the two conventional approaches to reinforcement learning. Unfortunately, these three combinations are unable to have the strengths of both approaches: to offer the data efficiency of the model-based approach, and to maintain the asymptotic accuracy of the model-free approach. The Dyna architecture cannot prevent the negative impacts of one approach on the other. The switching idea requires an additional selector signal that is often difficult to find. Finally, the weighted combination with the weights set in a fixed schedule is not applicable to non-stationary environments. The Cascade Architecture, presented in the next chapter, will overcome these limitations.

Chapter 5

The Cascade Architecture

This chapter presents our main contribution, the Cascade Architecture for combining model-based and model-free approaches to reinforcement learning, and also shows Cascade Architecture's effectiveness in preserving the strengths of model-based and model-free reinforcement learning. Moreover, we prove here that in the prediction setting, the Cascade Architecture achieves the same asymptotic solution as the original model-free solution and, therefore, prevents any imperfect model from impairing its asymptotic performance.

The Cascade Architecture flexibly and usefully combines the two approaches in order to exploit their strengths and to eliminate their weaknesses. The main idea behind the Cascade Architecture is to have a series of processes improving the predictions of the previous ones. This idea has been pursued by prior work (Fahlman and Lebiere, 1990; Silver et al., 2008). We review an instance of this idea in the context of supervised learning, before moving to the context of reinforcement learning.

5.1 Cascade-Correlation Algorithm

The Cascade-Correlation algorithm (Fahlman and Lebiere, 1990) is a supervised learning algorithm for training a neural network. The algorithm begins training the network without any hidden units until the algorithm is no longer able to significantly reduce the error. At this point, if the error on the entire training set is higher than a threshold, the algorithm adds a

hidden unit that is highly correlated with the error. Once the unit is added, its input weights are frozen. In the next iteration, the algorithm trains the network using the input features as well as the newly added hidden unit. Cascade-Correlation continues to iterate until the residual error is less than the specified threshold. Figure 5.1 illustrates the Cascade-Correlation algorithm.

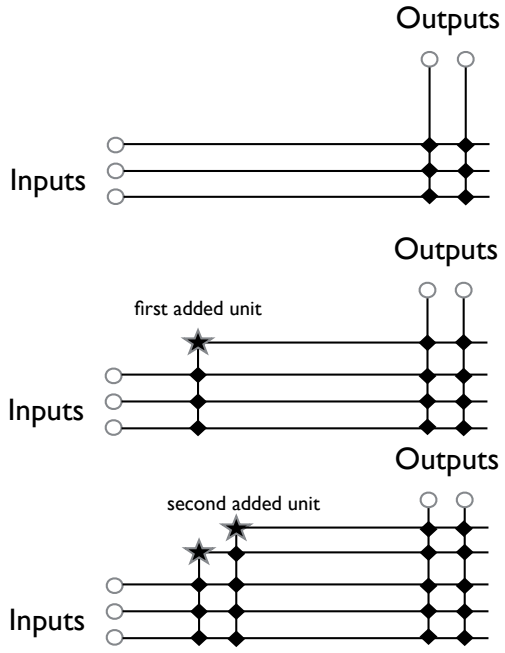


Figure 5.1: An illustration of the Cascade-Correlation algorithm. The neural network starts with no hidden units and adds hidden units that are correlated with the error until the error on the training set is less than a specified threshold.

5.2 Principles of the Cascade Architecture

Inspired by the Cascade-Correlation algorithm, we refer to the new architecture as the *Cascade Architecture*. Here are the principles of the Cascade Architecture:

- Cascade Architecture consists of a series of processes: each process takes as input the net contribution of the previous processes while being oblivious to the subsequent processes.
- The processes of the Cascade Architecture have a common goal and are cooperating and working towards that common goal. The goal could be

to find the best classifier, to minimize a prediction error, or to maximize reward by finding the best policy.

- Each process only performs local updates and does not update the parameters of any other processes.

In this thesis we focus on the *additive Cascade*, in which each process adds its contribution to the sum of the contributions of the previous processes. In general, though, it is straightforward and useful to have other forms of Cascade as well. Figure 5.2 illustrates the additive Cascade Architecture.

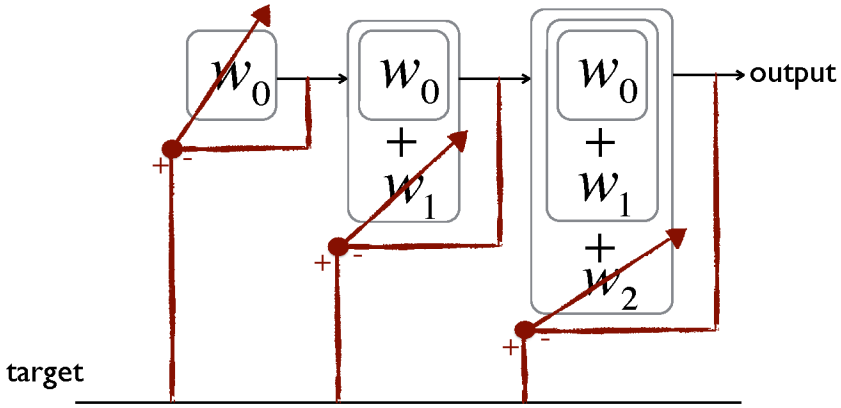


Figure 5.2: Three parallel processes combined in an additive Cascade. All of the processes are simultaneously adapting their own parameters. A line crossing an entity (a parameter vector or a rectangle) denotes that the entity is changing according to the corresponding error. Any rectangle provides a valid solution to the problem. For instance, $\mathbf{w}_1$, by itself, cannot make a valid solution. Whereas, $\mathbf{w}_0$ and $\mathbf{w}_1$ together can provide a valid solution to the problem.

5.3 Linear Cascade

Having defined the principles of the Cascade Architecture, we introduce a new reinforcement learning algorithm that combines direct and indirect reinforcement learning with linear function approximation. The algorithm, called linear Cascade, consists of a model-based process that forms the initial value function estimate. The initial value function estimate is defined as:

$$\hat{q}_p(s,a) = \mathbf{b}^{a\top}\boldsymbol{\phi}(s) + \gamma\boldsymbol{\theta}^\top\mathbf{F}^a\boldsymbol{\phi}(s)\ . \tag{5.1}$$

This value function estimate is learned by incrementally adapting the linear model $\{\mathbf{b}^a, \mathbf{F}^a | \ \forall a \in \mathcal{A}\}$ and planning with the linear model in a Dyna style. Note that all the parameters ($\boldsymbol{\theta}$, $\mathbf{b}^\bullet$, and $\mathbf{F}^\bullet$) of the initial value function estimate are learned by the model-based process.

The model-free process comes next by contributing to the final value function estimate. It takes the initial value function estimate $\hat{q}_p$, adds its contribution $\mathbf{w}$, and adapts $\mathbf{w}$ to further reduce the error. Since the final value function is the output of the entire algorithm, we also refer to it as the Cascade value function estimate $\hat{q}_c$. The Cascade value function estimate takes the form:

$$\hat{q}_c(s,a) = \hat{q}_p(s,a) + \mathbf{w}^\top \boldsymbol{\varphi}(s,a) \ . \tag{5.2}$$

It is only the model-free process that updates the weight vector $\mathbf{w}$. In effect, the model-free process adapts $\mathbf{w}$ to reduce the residual error of $\hat{q}_p$ (which itself could be thought of as a bias term for the model-free process), and to form a more accurate value function estimate. To learn the weight vector $\mathbf{w}$, we use linear Sarsa(0) algorithm with the update:

$$\begin{aligned} &\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha \delta_t \boldsymbol{\varphi}(S_t, A_t) \\ &\text{where } \delta_t = R_{t+1} + \gamma \hat{q}_c(S_{t+1}, A_{t+1}) - \hat{q}_c(S_t, A_t) \end{aligned} \tag{5.3}$$

Notice that we have used different notations ($\boldsymbol{\phi}$ and $\boldsymbol{\varphi}$) for feature vectors of the two processes to denote that they can be different. More generally, within the Cascade Architecture each approach has the flexibility to employ its own representation, parameters, and objective function. This flexibility is desired and, as we will show later, yields a significant advantage. Figure 5.3 illustrates the internal structure of the linear Cascade and Algorithm 8 gives the complete code for this algorithm.

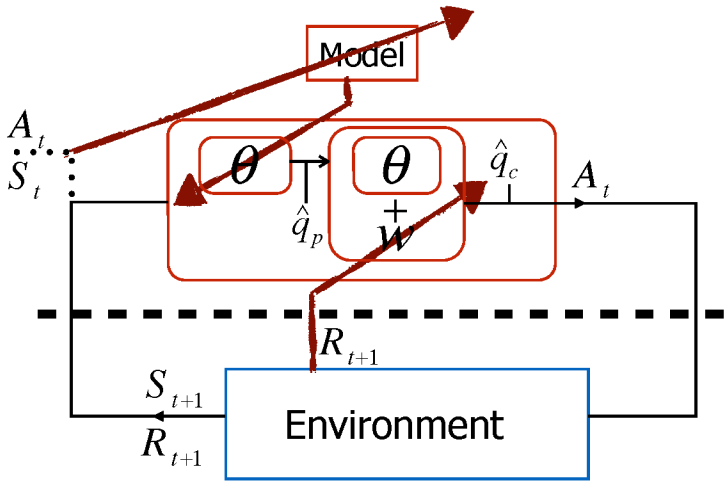


Figure 5.3: The structure of the linear Cascade algorithm. This algorithm forms two value function estimates. The first one is adapted only by the model-based process, while the second one consists of a weight vector adapted by the model-free process added to the first value function estimate.

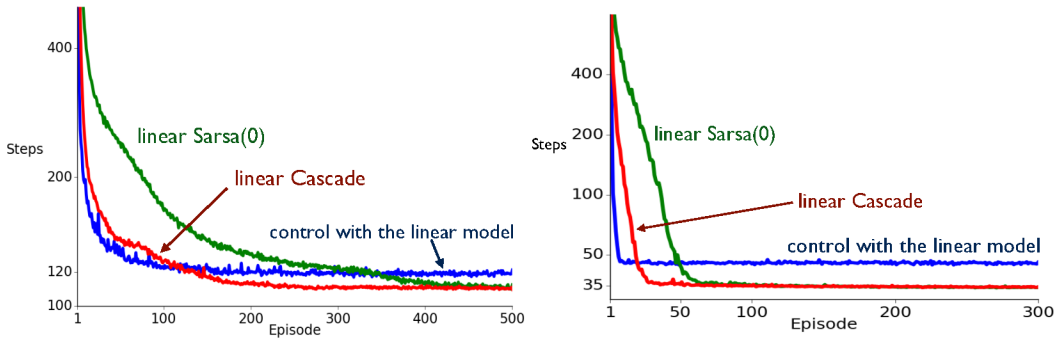
Algorithm 8: linear Cascade for control with ϵ-greedy policy

Initialize $\boldsymbol{\theta}$, $\mathbf{w}$, and $\{\mathbf{b}^a, \mathbf{F}^a | \ \forall a \in \mathcal{A}\}$ and obtain initial state S
$A \leftarrow \arg\max_{a \in \mathcal{A}} \{\hat{q}_c(S, a)\}$ with Pr. $1 - \epsilon$ else from $\mathcal{A}$ at random
for each time step **do**
  Take action A, Receive next reward R and receive next state S'
  $A' \leftarrow \arg\max_a \{\hat{q}_c(S', a)\}$ with Pr. $1 - \epsilon$ else from $\mathcal{A}$ at random
  $\mathbf{F}^a \leftarrow \mathbf{F}^a + \alpha_p \Big(\boldsymbol{\phi}(S') - \mathbf{F}^a \boldsymbol{\phi}(S)\Big) \boldsymbol{\phi}(S)^\top \qquad$ for $a = A$
  $\mathbf{b}^a \leftarrow \mathbf{b}^a + \alpha_p \Big(R - \mathbf{b}^{a\top} \boldsymbol{\phi}(S)\Big) \boldsymbol{\phi}(S) \qquad$ for $a = A$
  for p times for planning **do**
    for i in range 1 to n **do**
      $\boldsymbol{\phi}_P \leftarrow \mathbf{e}_i$
      $a \leftarrow \arg\max_a \{\mathbf{b}^{a\top} \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}_P\}$ with Pr. $1 - \epsilon$ else from $\mathcal{A}$ at random
      $\boldsymbol{\theta}^i \leftarrow \mathbf{b}^{a\top} \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}_P$
    end
  end
  $\delta \leftarrow R + \gamma \hat{q}_c(S', A') - \hat{q}_c(S, A)$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha_c \delta \boldsymbol{\varphi}(S, A)$
  $S \leftarrow S'$
  $A \leftarrow A'$
end
define $\hat{q}_c(s, a)$:
  return $\hat{q}_p(s, a) + \mathbf{w}^\top \boldsymbol{\varphi}(s, a)$
define $\hat{q}_p(s, a)$:
  return $\mathbf{b}^{a\top} \boldsymbol{\phi}(s) + \gamma \boldsymbol{\theta}^\top \mathbf{F}^a \boldsymbol{\phi}(s)$

5.4 Empirical Results

In chapter 3 we reported experiments on the mountain car and the puddle world domains comparing linear Sarsa(0) and prediction with the linear model. In one of the experiments, we controlled for the computational resources used by the two algorithms. In this case the model-based algorithm performed better with limited experience while the cheaper model-free algorithm found a better asymptotic solution by using a larger representation.

Here we repeat that particular experiment in order to compare the newly introduced algorithm, linear Cascade, with linear Sarsa(0) and control with the linear model. In both domains, linear Cascade's model-free process used the same representation as linear Sarsa(0) and linear Cascade's model-based process used the same representation as control with the linear model. These representations were kept similar to the ones employed in chapter 3. We ran each algorithm for 500 times and computed the average performance over the runs. The step-sizes were optimized for the best asymptotic performance. Linear Cascade used the step-sizes optimized individually for linear Sarsa(0) and control with the linear model. The results are shown by Figure 5.4.



(a) mountain car (b) puddle world

Figure 5.4: Comparison between linear Sarsa(0), control with the linear model, and linear Cascade on the mountain car and the puddle world domains. Linear Cascade is showing a good interim performance and is also able to achieve an asymptotic performance similar to linear Sarsa(0).

As shown by the graphs, linear Cascade finds a good policy with limited experience. Moreover, the asymptotic performance of the linear Cascade is

identical to that of linear Sarsa(0), and they both are performing better than control with the linear model.

In the above experiments presented by Figure 5.4, the running time of the linear Cascade was the sum of the running time of linear Sarsa(0) and control with the linear model. We repeated the experiment on the mountain car domain, this time forcing the three algorithms to exactly take the same amount of computation. To that end, we increased the number of features used by linear Sarsa(0) and control with the linear model. Linear Sarsa(0) used 15 tilings each with 225 features, while linear Cascade's model-free process still used 10 tilings each with 225 features. Moreover, control with the linear model used 3 tilings each with 225 features, whereas linear Cascade's model-based process still used 2 tilings each with 225 features. The results are shown by figure 5.5.

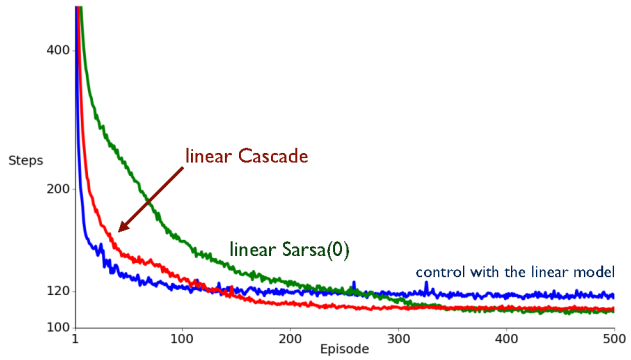


Figure 5.5: Another comparison between linear Sarsa(0), control with the linear model, and linear Cascade on the mountain car domain where the three algorithms have the exact same running time. The initial advantage of control with the linear model over linear Cascade is slightly increased. Also, linear Sarsa(0) is performing slightly better than the linear Cascade on the limit. However, the benefits from combining the two algorithms in a Cascade are still present.

5.5 Convergence Analysis

In order to analyze the convergence of the linear Cascade, we first introduce an alternative interpretation of the role of the model-free process in the linear Cascade. The role of the model-free process can be thought of as predicting an augmented reward signal. In the prediction case and after a sample transition

from S to S' with reward R, the TD error is:

$$\begin{aligned}\delta &= R + \gamma(\mathbf{b}^\top + \gamma\boldsymbol{\theta}^\top\mathbf{F})\boldsymbol{\phi}(S') + \gamma\mathbf{w}^\top\boldsymbol{\varphi}(S') \\ &\quad - (\mathbf{b}^\top + \gamma\boldsymbol{\theta}^\top\mathbf{F})\boldsymbol{\phi}(S) - \mathbf{w}^\top\boldsymbol{\varphi}(S) \\ &= R + (\mathbf{b}^\top + \gamma\boldsymbol{\theta}^\top\mathbf{F})\Big(\gamma\boldsymbol{\phi}(S') - \boldsymbol{\phi}(S)\Big) + \mathbf{w}^\top\Big(\gamma\boldsymbol{\varphi}(S') - \boldsymbol{\varphi}(S)\Big)\end{aligned}$$

We define the augmented reward signal, $\tilde{R}$, as the original reward signal plus the contribution of the model-based process:

$$\tilde{R} = R + (\mathbf{b}^\top + \gamma\boldsymbol{\theta}^\top\mathbf{F})\Big(\gamma\boldsymbol{\phi}(S') - \boldsymbol{\phi}(S)\Big).$$

Given this new reward, the temporal difference error used by the model-free process is:

$$\delta = \tilde{R} + \gamma\mathbf{w}^\top\boldsymbol{\varphi}(S') - \mathbf{w}^\top\boldsymbol{\varphi}(S) \tag{5.4}$$

This signal could be thought of as a noisy approximation of a well-defined reward signal in a standard Markov reward process after the convergence of $\boldsymbol{\theta}$.

For the algorithms used in the linear Cascade, both algorithms will converge under standard conditions. We restrict our attention for convergence to the setting where the state space is finite and the rewards are deterministic. We assume that (i) the Markov chain induced by the policy is aperiodic and irreducible, (ii) that both sets of features are linearly independent, (iii) that the linear transition model $\mathbf{F}$ has a numerical radius less than one, and (iv) the step-sizes are decreasing at a rate that satisfies $\sum_{k=1}^{\infty}\alpha_k = \infty$ and $\sum_{k=1}^{\infty}\alpha_k^2 < \infty$. The convergence of prediction with the linear model to the LSTD(0) solution under these conditions is described by the prior work (Sutton et al., 2008). The convergence of linear TD algorithms with probability one under these conditions is also proved (Tsitsiklis and Van Roy, 1997). For the convergence of linear TD algorithms in the linear Cascade, note that the weights are effectively updated with $\tilde{R}$. As linear TD algorithms converge with a noisy reward signal, the model-free process of the linear Cascade converges as well.

Algorithm 9: linear Cascade for prediction

Initialize $\boldsymbol{\theta}$, $\mathbf{w}$, and $\{\mathbf{b}, \mathbf{F}\}$ and obtain initial state S
for each time step **do**
  Receive next reward R and receive next state S'
  $\mathbf{F} \leftarrow \mathbf{F} + \alpha_p \Big(\boldsymbol{\phi}(S') - \mathbf{F}\boldsymbol{\phi}(S) \Big) \boldsymbol{\phi}(S)^\top$
  $\mathbf{b} \leftarrow \mathbf{b} + \alpha_p \Big(R - \mathbf{b}^\top \boldsymbol{\phi}(S) \Big) \boldsymbol{\phi}(S)$
  for p times for planning **do**
    for i in range 1 to n **do**
      $\boldsymbol{\phi}_P \leftarrow \mathbf{e}_i$
      $\boldsymbol{\theta}^i \leftarrow \mathbf{b}^\top \boldsymbol{\phi}_P + \gamma \boldsymbol{\theta}^\top \mathbf{F} \boldsymbol{\phi}_P$
    end
  end
  $\delta \leftarrow R + \gamma \hat{v}_c(S') - \hat{v}_c(S)$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha_c \delta \boldsymbol{\varphi}(S)$
  $S \leftarrow S'$
  $A \leftarrow A'$
end
define $\hat{v}_c(s)$:
  return $\hat{v}_p(s) + \mathbf{w}^\top \boldsymbol{\varphi}(s)$
define $\hat{v}_p(s)$:
  return $\mathbf{b}^\top \boldsymbol{\phi}(s) + \gamma \boldsymbol{\theta}^\top \mathbf{F} \boldsymbol{\phi}(s)$

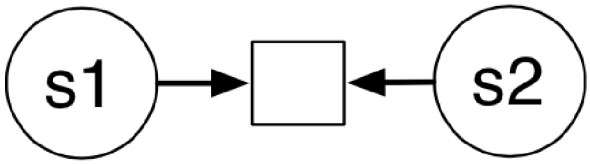
A more important question is related to the convergence point of the Cascade Architecture. We next show that the Cascade Architecture achieves an asymptotic performance equivalent to using a model-free algorithm alone, when the model-free process of the Cascade Architecture finds a globally optimum for the weights for an objective J, both processes use linear function approximation, and the features used for the model-based process can be represented as a linear function of the features used for the model-free process.

Theorem: Asymptotic equivalence of the solutions found by the Cascade Architecture and the model-free process alone: Consider two adaptive processes P and L combined in the Cascade Architecture where both use linear function approximation over states, both converge on the original problem, and they also converge in the Cascade. Suppose that the second process L adapts the weight vector to converge to a global minimum of an objective function J. Suppose also that the feature vector $\boldsymbol{\phi}$ used by P is representable by the feature vector $\boldsymbol{\varphi}$ used by L, so there is a matrix $\mathbf{M}$ such that for all states s,

$\boldsymbol{\phi}(s) = \mathbf{M}\boldsymbol{\varphi}(s)$. Then the asymptotic solution of the Cascade Architecture and the asymptotic solution of the second process alone are equivalent according to J.

Proof: Let the asymptotic solution from the second process alone be given by the weight vector $\mathbf{u}$, where $f_u(s) = \mathbf{u}^\top \boldsymbol{\varphi}(s)$ and $\mathbf{u} = \arg\min_{\mathbf{z}} J(f_{\mathbf{z}})$. Let the asymptotic solution for the Cascade Architecture be given by the weight vector $\mathbf{h}$ for P and the weight vector $\mathbf{w}$ for L, where $g_{\mathbf{h},\mathbf{w}}(s) = \mathbf{h}^\top \boldsymbol{\phi}(s) + \mathbf{w}^\top \boldsymbol{\varphi}(s)$ and $\mathbf{w} = \arg\min_{\mathbf{z}} J(g_{\mathbf{h},\mathbf{z}})$. Note that $\mathbf{h}^\top \boldsymbol{\phi}(s) + \mathbf{z}^\top \boldsymbol{\varphi}(s) = \mathbf{h}^\top \mathbf{M}\boldsymbol{\varphi}(s) + \mathbf{z}^\top \boldsymbol{\varphi}(s) = (\mathbf{M}^\top \mathbf{h} + \mathbf{z})^\top \boldsymbol{\varphi}(s)$. For any choice of $\mathbf{h}$, the set of functions considered by the Cascade, namely $\{g_{\mathbf{h},\mathbf{z}}\}_{\mathbf{z}}$, and the set of functions considered by L, namely $\{f_{\mathbf{z}}\}_{\mathbf{z}}$, are identical as $g_{\mathbf{h},\mathbf{z}} = f_{\mathbf{M}^\top \mathbf{h}+\mathbf{z}}$. Hence, the weights found by optimizing J must yield equivalent asymptotic solutions. □

This equivalence theorem is stated for general processes in the Cascade Architecture with different representations, objective functions, and parameters for each process. From the theorem, the model-based process can not impair the asymptotic performance, which sets the stage for using more extreme approximations in model building.



State	R	Value	$\boldsymbol{\varphi}$	$\boldsymbol{\phi}$
s1	1	1	(1)	(1)
s2	-1	-1	(-1)	(0)

Figure 5.6: The two-state MRP example. An example where Cascade cannot achieve the performance of the original model-free process.

We finally present a two state MRP example presented in Figure 5.6 as an example where the Cascade Architecture does not achieve the asymptotic performance of the original model-free solution when the feature vectors in the model-free process can not represent the feature vectors in the model-based

process. The MRP enters the states s1 and s2 with equal probability and transitions immediately to termination. First notice that the pure model-free process will estimate the values exactly with a weight of one. The model-based process of the Cascade will converge to a weight of one as well, but will have a residual error of 0 for s1 and -1 for s2. As the model-free process of the Cascade can not represent this difference, the Cascade Architecture will not have a zero error, and cannot achieve the exact values.

5.6 Compatibility with the Eligibility Traces

One of the strengths of the model-free approach, presented in chapter 3, was the compatibility with the eligibility traces. The flexibility offered by the Cascade Architecture allows us to use eligibility traces in the linear Cascade. The model-free process of the linear Cascade can use linear Sarsa(λ) with the following update[1]:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \alpha\Big(R_{t+1} + \gamma\hat{q}_c(S_{t+1}, A_{t+1}) - \hat{q}_c(S_t, A_t)\Big)\mathbf{z}_t \qquad , \tag{5.5}$$

$$\mathbf{z}_t \leftarrow \gamma\lambda\mathbf{z}_{t-1} + \boldsymbol{\varphi}(S_t, A_t) \tag{5.6}$$

We revisit the hallway experiment discussed in the previous chapter where the eligibility traces were essential for reaching the optimal solution. This time we also ran the linear Cascade with $\lambda = 0.9$. The results are shown in Figure 5.7.

[1]The update is shown for the accumulating trace. In general other forms can be used as well.

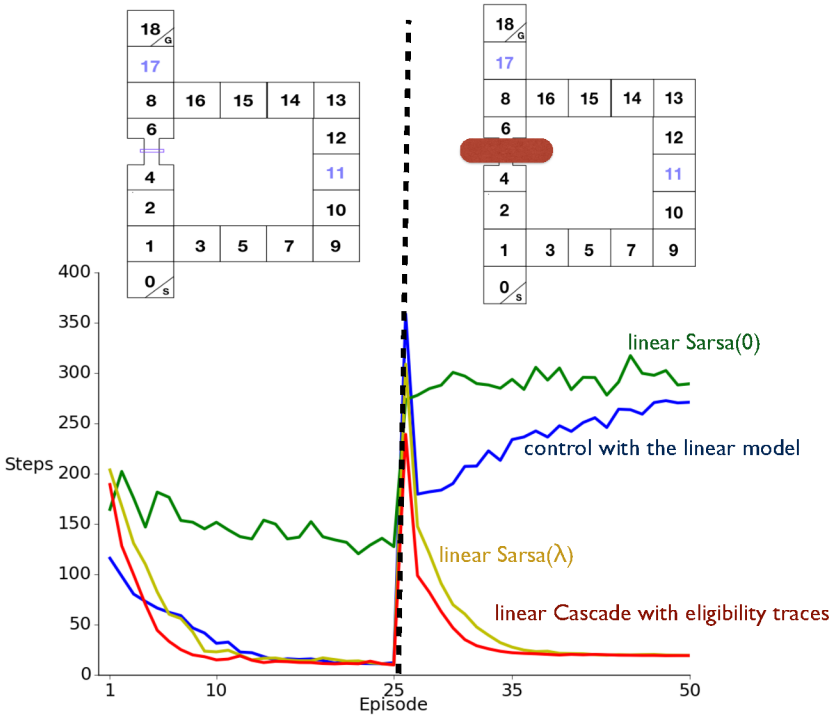


Figure 5.7: Control results on the hallway domain. Linear Cascade can reach the optimal policy similar to linear Sarsa(λ), whereas single-step algorithms like linear Sarsa(0) and control with one-step linear model are performing poorly.

This result shows that the Cascade Architecture is able to preserve the benefits coming from using the eligibility traces, as the last strength of the model-free approach introduced in chapter 3.

5.7 Dyna 2

Dyna-2 (Silver et al., 2008) is another useful way of combining model-based and model-free reinforcement learning. We consider Dyna-2 as an instance of the Cascade Architecture, because it satisfies the three principles of the Cascade. In Dyna-2 model-based and model-free processes are combined in a series, the processes work towards predicting the value function, and they both update their own sets of parameters.

Dyna-2 has a more specialized benefit than our Cascade. A model of the agent's environment, the game of Go, is available to Dyna-2. In Dyna-2, model-free process forms a permanent value function estimate, and

the model-based process improvs the permanent value function estimate by learning a transient weight vector that is reinitialized at each time step. Also, the two processes are given the same set of features.

5.8 Conclusion

This chapter presented the Cascade Architecture for combining model-based and model-free approaches. We showed that the Cascade Architecture enables fast adaptation with limited environmental interactions. The Cascade Architecture guarantees convergence to the original model-free solution and, finally, it is compatible with eligibility traces. The results presented here open up the possibility for using more extreme approximations in model building without asymptotic harm. More generally, these results strengthen the case for combining model-based and model-free reinforcement learning.

Chapter 6

Conclusion and Future Work

This final chapter is a chance to review our contributions, to discuss the limitations of our work, and to present ideas for future work.

6.1 Contributions

This thesis took important steps towards understanding the longstanding debate between model-based and model-free reinforcement learning. Our important finding was that neither approach can dominate the other. The Model-based approach performs better with fewer environmental interactions, whereas the model-free approach can perform better asymptotically by using a larger representation or the eligibility traces. This trade-off was the primary reason for us to search for an architecture that combines the two approaches.

The most important contribution of this thesis was to develop the Cascade Architecture as an effective way of combining model-based and model-free approaches. To establish Cascade's effectiveness, we showed its data efficient performance in our experimental results. We also showed that the Cascade Architecture is able to reach the original model-free solution and, therefore, prevent the model-based approach to degrade the asymptotic performance. The idea behind the Cascade Architecture is also generally useful for combining parallel processes with a common goal.

Another contribution presented here was to extend the prior work and show a more general equivalence between the linear TD(λ) solution and the linear

β-model solution when they both use an identical representation. While this thesis frequently mentioned and investigated the differences between model-based and model-free approaches, this general equivalence showed the value of looking for potential similarities between the two approaches.

6.2 Limitations

The main limitation of this thesis, in our view, is the lack of theoretical results on data efficiency of the Cascade Architecture. While we made statements regarding Cascade's asymptotic performance, we relied on empirical evaluations to assess Cascade's data efficiency. Having results of this form seems unlikely at this point, because the field of reinforcement learning lacks theoretical results related to data efficiency of even simpler algorithms in the approximate case. Finding data efficiency results will, unfortunately, get even harder when we move beyond the linear function approximation setting.

A second limitation of this work was to consider only a single family of algorithms from each conventional approach. Due to the recent popularity of the field of reinforcement learning, many model-based and model-free algorithms have been proposed. But, we considered only a small number of algorithms for the sake of an easier analysis. Nevertheless, the algorithms considered her are fundamental and also widely used.

6.3 Future Work

The first idea for future work is to go beyond the linear function approximation case and to the more general case of non-linear function approximation. This future work is partly motivated by the recent success of the field of deep learning, and partly by the desire to extend the results to a more general case. This direction will, of course, bring its own challenges and questions that are beyond the scope of this thesis.

Another idea is to go beyond one-step models to the more general case of multi-step models. This thesis mentioned β-model as a form of multi-step

models and explained its benefit. The scope of this thesis was, however, restricted to single-step models. Planning with multi-step models can even be more useful and, therefore, we generally want to be able to model the environment on multiple time-scales.

As shown in chapter 5, the Cascade Architecture supports planning with smaller models while reaching to the solution found by the model-free process. In other words, Cascade prevents any imperfect model from degrading the asymptotic performance. Another promising idea for future work would then be to exploit this property by pushing for models that are smaller in size and reduce the costs of planning.

Finally, the Cascade Architecture can be used in future to combine multiple algorithms from each individual approach. For example, a Cascade combination of multiple model-based algorithms each with different models can provide benefits. Cascade can also be used to combine multiple model-free algorithms with eligibility traces where each processes uses a different value of λ.

Bibliography

Atkeson, C. G., Santamaria, J. C. (1997). A comparison of direct and model-based reinforcement learning. In *Proceedings of the 14th International Conference on Robotics and Automation*, pp. 3557–3564.

Bellemare, M. G., Naddaf, Y., Veness, J., Bowling, M. (2012). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research 47*:253–279.

Bertsekas, D. P., Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA.

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

Daw, N. D., Niv, Y., Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience 8*:1704–1711.

de Farias, D. P., Van Roy, B. (2004). On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research 29*:462–478.

Fahlman, S. E., Lebiere, C. (1989). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, pp. 524–532.

Gershman, S. J., Markman, A. B., Otto, A. R. (2014). Retrospective revaluation in sequential decision making: A tale of two systems. *Journal of Experimental Psychology: General 143*:182–194.

Gläscher, J., Daw, N. D., Dayan, P., O'Doherty, J. P. (2010). States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron 66*:585–595.

Kakade, S. M., (2003). *On the sample complexity of reinforcement learning.* PhD thesis, University College London, England.

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

Moore, A. W., Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning 13*:103–130.

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. (2004). Autonomous inverted helicopter flight via reinforcement learning. In *International Symposium on Experimental Robotics*.

Parr, R., Li, L., Taylor, G., Painter-Wakefield, C., Littman, M. L. (2008). An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 752–759.

Peng, J., Williams, R. J. (1996). Incremental multi-step Q-learning. *Machine Learning 22*:283–290.

Rummery, G. A., Niranjan, M. (1994). Online Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166. Engineering Department, Cambridge University, England.

Silver, D., Sutton, R. S., Müller, M. (2008). Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 968–975.

Strehl, A. L., Li, L., Littman, M. L. (2009). Reinforcement learning in finite MDPs: PAC analysis. *Journal of Machine Learning Research 10*:2413–2444.

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

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the 7th International Conference on Machine Learning*, pp. 216–224.

Sutton, R. S. (1995). TD models: Modeling the world at a mixture of time scales. In *Proceedings of the 12th International Conference on Machine Learning*, pp. 531–539.

Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, pp. 1038–1044.

Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT press, Cambridge, MA.

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

Sutton, R. S., Szepesvári, C., Geramifard, A., Bowling, M. (2008). Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pp. 528–536.

Szita, I., Szepesvári, C. (2010). Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 1031–1038.

Tsitsiklis, J. N., Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control 42*:674–690.