Fall 2014

ADAPTIVE STEP-SIZES FOR REINFORCEMENT LEARNING

William C. Dabney
Computer Sciences

ADAPTIVE STEP-SIZES FOR REINFORCEMENT LEARNING

A Dissertation Presented

by

WILLIAM DABNEY

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

Computer Science

ADAPTIVE STEP-SIZES FOR REINFORCEMENT LEARNING

A Dissertation Presented

by

WILLIAM DABNEY

Approved as to style and content by:

Andrew G. Barto, Chair

Sridhar Mahadevan, Member

Benjamin Marlin, Member

Hari Jagannathan Balasubramanian, Member

Lori A. Clarke, Chair
Computer Science

for our shared quest

ACKNOWLEDGMENTS

The journey from an initial interest in research to completing a PhD is a long and challenging one. Sometimes an idea works out well, and sometimes it leads nowhere. All of the uncertainty can be the greatest challenge of the whole process. I was very fortunate to share my time in graduate school with people who have made the uncertainty tolerable and shared with me some of the best times of my life.

From an early age I was fascinated by the idea of artificial intelligence (AI). My early attempts were encouraged by my parents, Bill Dabney and Cathy Barnum, but received their biggest boost from two early influences. First, from my step-dad, Tom Barnum, who guided me through the idea of creating an intelligent program that would improve its play at the 20-questions game. He posed questions to my 12 year-old mind and left me to play with Visual Basic on his computer in order to sort out the answers for myself. As a result I created a simple decision-tree learning algorithm that played 20-questions and improved when it guessed wrong. The second big influence came immediately afterwards when I sold the program to my sister, Catie, for $10. Of course, having a copyright lawyer as a mother, this soon erupted into a licensing dispute. There were many lessons learned that week.

No teacher from my childhood had the foundational and long-lasting impact on my intellectual development that Tom provided by continually presenting interesting problems and asking hard questions. My mother, Cathy, recognized my potential and defended me against an early science teacher who was more concerned with bureaucracy than with learning. Thank you for protecting my love of learning from being snuffed out early on. My father and step-mother, Bill and Dorothy Dabney, have always shown pride and support in every way I could have asked for.

It was not until Amy McGovern's undergraduate AI course at the University of Oklahoma that I received a formal introduction to the field. I owe a huge debt to Dr. McGovern for teaching the class with such obvious enthusiasm and for giving us all far more than could be digested in a single semester. Through her undergraduate AI class I was able to join Dr. McGovern's research experiences for undergraduates program where I got my first taste of research and learned about reinforcement learning. These early experiences were instrumental in developing my excitement for research, and taught me how fun the whole process could be.

When I came to UMass for graduate school it was to work with my advisor Andy Barto. From reading the book written by Andy and his first PhD student I was, and still am, certain that reinforcement learning is the most interesting and promising area of research in artificial intelligence. I expected a brilliant super-star of the field with ego to match, but I was only half right. Andy is an exceptional advisor for his ability to quickly digest new ideas, ask incredibly challenging questions, and guide research contributions away from the incremental and toward the fundamental. He does all of this while maintaining an incredibly down-to-earth personality and humble demeanor that made me feel safe in sharing my less than brilliant ideas and questions. Thank you Andy, you have changed my life and shown me the best example of the type of researcher I aim to become.

I have always had a fondness for math, not arithmetic which I continue to fumble on a daily basis, but the elegance and power of symbolic reasoning found in mathematics. Sridhar Mahadevan has an ability to communicate mathematically beyond anyone I have ever met. He has thoroughly challenged me both in his courses and in our research collaborations. He has made me a better researcher and strengthened my mathematically maturity greatly.

While in graduate school I have worked with a couple of other professors outside my research area that have left a lasting impact on me. Robert Moll taught me important lessons about teaching computer science to new students. James Allen helped to expose me to new areas of computer science research that have greatly expanded my knowledge about topics outside of machine learning. And, while not a professor, Leeanne Leclerc has my undying gratitude because without her constant help I would have never made it through graduate school.

A PhD is a somewhat solitary process. It is your research and you are the expert on it. But fortunately for my own sanity and intellectual growth I have shared graduate school with a group of good friends and excellent researchers. George Konidaris has been like a second advisor to me. He has masterfully played the part of the senior graduate student and mentor, imparting apt advice and was always available to discuss my research ideas and frustrations. Forgive the fatalism, but George is a man destined to be a great research professor. He already is, in all but title. Philip Thomas has been a good friend and great research collaborator. He writes with clarity and has the greatest of mathematical integrity. Thank you for showing me that research collaboration can be fun and produce fantastic results. Scott Niekum somehow managed to make graduate school feel fun and exciting while producing work faster than should be humanly possible. His lightening fast rate of progress did more than most to motivate me to graduate sooner rather than later.

The Autonomous Learning Lab is the group of students I would have chosen to work and socialize with even if we did not share so many research interests. My fellow graduate students had a profound impact on my work and my ability to work. They made my thinking and my research better. There are many more that deserve thanks, but research has given me a bias for brevity. Reducing any of these relationships to a few sentences is so absurd, but it would require an entire dissertation to go into all the ways in which these amazing characters have shaped my life.

Colin Barringer's influence on me certainly skews toward the social, and through him I met one of my closest friends, JP Bracey. Both of whom have helped me live a fuller life while pursuing my PhD. I had the great fortune of living with incredible roommates for much of my time at UMass. TJ Brunette and I shared great adventure rock climbing in Sunderland. Danilla Musante kept me honest with myself. Lauri, Hanna, and Amanda of the neuroscience department brought me great friendships and a healthy dose of insanity, thank you so much. In particular, thank you Lauri Kurdziel for being such a good friend and introducing me to West Coast Swing. Blair Lyonev I have so much to thank you for, but here let me say thank you for bringing me to tears with your writing.

My favorite part of teaching has always been to walk through concepts one-on-one and get to watch understanding and skill grow in one mind. Although I cannot take credit for teaching her much of anything, I did get to see that process in Presley Pizzo and it was amazing to watch. I may be acknowledging too many friends and not enough collaboration, but that is perhaps indicative of having learned more in my hours socializing than in my hours working.

When I first visited UMass Gene Novark, in his characteristic style, tried to dispel my rosy expectations for graduate school. When that did not work he gave me a glass of bourbon and sent me back to Oklahoma. Even though his and others' stern warnings did not deter me, I learned something important from that first impression of UMass: graduate school is harder than you expect, but you can get through it while having fun with the help of good friends. Good friends and good bourbon.

ABSTRACT

ADAPTIVE STEP-SIZES FOR REINFORCEMENT LEARNING

SEPTEMBER 2014

WILLIAM DABNEY

B.Sc. Computer Science, UNIVERSITY OF OKLAHOMA

B.Sc. Mathematics, UNIVERSITY OF OKLAHOMA

M.Sc. Computer Science, UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew G. Barto

The central theme motivating this dissertation is the desire to develop reinforcement learning algorithms that "just work" regardless of the domain in which they are applied. The largest impediment to this goal is the sensitivity of reinforcement learning algorithms to the step-size parameter used to rescale incremental updates. Adaptive step-size algorithms attempt to reduce this sensitivity or eliminate the step-size parameter entirely by automatically adjusting the step size throughout the learning process. Such algorithms provide an alternative to the standard "guess-and-check" methods used to find parameters known as *parameter tuning.*

However, the problems with parameter tuning are currently masked by the way experiments are conducted and presented. In this dissertation we seek algorithms that perform well over a broad subset of reinforcement learning problems with minimal parameter tuning. To accomplish this we begin by addressing the limitations of current empirical methods in reinforcement learning and propose improvements with benefits far outside the area of adaptive step-sizes.

In order to study adaptive step-sizes in reinforcement learning we show that the general form of the adaptive step-size problem is a combination of two dissociable problems (*adaptive scalar step-size* and *update whitening*). We then derive new parameter-free adaptive scalar step-size algorithms for the reinforcement learning algorithm Sarsa(λ) and use our improved empirical methods to conduct a thorough experimental study of step-size algorithms in reinforcement learning. Our adaptive algorithms (VES and PARL2) both eliminate the need for a tunable step-size parameter and perform at least as well as Sarsa(λ) with an optimized step-size value. We conclude by developing natural temporal difference algorithms that provide an approximate solution to the update whitening problem and improve performance over their non-natural counterparts.

TABLE OF CONTENTS

Page

CHAPTER

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Reinforcement learning (RL) provides a framework for learning in sequential decision making problems [Sutton and Barto, 1998a]. Within this framework an *agent* interacts with the *environment* by taking actions and observing the consequences of those actions. The RL problem is to learn what actions to take based upon delayed feedback provided by a *reward function*. The interactions between agent and environment are often modeled as a Markov Decision Process (MDP) which defines a *domain* in which an RL algorithm may be used. The RL algorithm provides the intelligence of the agent. The agent's ability to adapt to the conditions of the domain and learn actions which maximize reward is controlled by the RL algorithm.

The central theme motivating this dissertation is the desire to develop RL algorithms that "just work" regardless of the domain in which they are applied. However, the *No Free Lunch Theorem* provides some important perspective and sanity here [Wolpert and Macready, 1997]. We cannot expect any learning algorithm to be equally effective over all possible RL domains. The use of tunable algorithm parameters allows the learning process to be biased in a domain-dependent way, partially off-setting the consequences of the no free lunch theorem. The process for choosing these parameter values is best described as "guess and check", a process commonly known as *parameter tuning*. Parameter tuning involves *guessing* a value for the different tunable parameters and running the learning algorithm to *check* how well it performs.

Evaluating an RL algorithm without evaluating its parameter tuning process provides an occluded and biased measure of the algorithm's performance. We propose that the concept of *ecological optimality* is the most appropriate paradigm in which to evaluate the parameter tuning process of an RL algorithm. Under this paradigm we seek an algorithm that performs well with minimal parameter tuning on a broad set of domains. This set of domains should be representative of the types of problems we wish to solve. In Chapter 3 we provide evidence supporting this claim and present an improved set of empirical methods for RL that facilitate research into ecologically optimal RL algorithms.

The step-size parameter controls the rate at which new information is accumulated by the learning algorithm and is among the most pervasive tunable parameters in machine learning. An *adaptive step-size* for RL algorithms which automatically adjusts the step-size value during the learning process would provide a significant step towards eliminating the need for parameter tuning. RL algorithms are often more sensitive to the value of the step-size parameter than to any other parameter and the choice of step-size parameter can profoundly affect the performance of the learning algorithm. Research on step sizes is dominated by contributions from the fields of convex optimization and stochastic approximation, but many of the assumptions and requirements of these fields do not transfer to the RL setting.

Recent research into adaptive step sizes has focused on the development of matrix and vector value adaptive step sizes [Duchi et al., 2011b, Schaul et al., 2012, Ross et al., 2013]. Whereas the scalar step-size we have discussed so far scales the magnitude of updates performed by an algorithm, matrix value step-sizes both scale the magnitude and modify the direction of updates. However, a matrix-valued adaptive step-size solves the combination of two problems: the *adaptive scalar step-size problem* and the *update whitening problem*. Informally, the update whitening problem is to adjust the direction of updates to correct for approximation errors caused by minimizing a linear approximation of the loss function.

In the remainder of Chapter 4 we derive novel adaptive scalar step-size algorithms for RL and bring to bear our newly developed empirical methods to perform a thorough study of adaptive step-sizes in RL in general and adaptive step-size algorithms in particular. Finally, in Chapter 5 we develop and present the *Natural Temporal Difference Learning* class of algorithms which approximately solve the update whitening problem in action-value based RL. First, we will review some background information on RL and adaptive step-sizes in Chapter 2, and in Chapter 3 we introduce an improved set of empirical methods for RL which are used in later chapters.

1.1 Contributions

This dissertation makes the following contributions towards the problems discussed above:

1. *Empirical methods for RL.* We begin by motivating the need for these methods with a review of recently published work in RL and illustrate the limitations of current practices. Our proposed set of methods allow for more informative experimental results and make the evaluation of parameter tuning explicit.

2. *Parameter-free adaptive scalar step-sizes for RL.* Chapter 4 presents our derivation of three parameter-free algorithms for the adaptive scalar step-size problem and a thorough empirical study of their behavior.

3. *Natural Temporal Difference Learning.* Chapter 5 presents a class of algorithms for approximately solving the update whitening problem for action-value based RL.

Together these contributions advance the study of the adaptive step-size problem in RL; beginning with the empirical methods needed, continuing to the definition of the adaptive scalar step-size and update whitening problems, and concluding with the derivation and evaluation of adaptive algorithms for these two problems.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter we review the two areas at the heart of this work: reinforcement learning and adaptive step sizes. We cover important background material for each topic and review related work on adaptive step sizes. Both areas have an extensive body of research surrounding them, but research into adaptive step-sizes for RL is relatively rare.

2.1 Reinforcement Learning

Consider the sequential decision making setting shown in Figure 2.1. In this setting the *agent* interacts with an *environment*. At each time step t the agent observes the current state s_t and a *reward signal* r_t. The agent then takes an action a_t and the time step increments by one. The environment in turn produces new state, s_{t+1}, and reward signal, r_{t+1}. The reward r_{t+1} provides evaluative feedback on the desirability of the action a_t leading to state s_{t+1}. A mapping from states to actions is referred to as a *policy*, and is denoted by π. Policies may be deterministic, always resulting in the same action in a given state such as $\pi(s) = a$, or stochastic shown with $a \sim \pi(s)$, taking different actions based on a probability distribution conditioned on the state. The Reinforcement Learning (RL) problem faced by the agent is to learn a policy that maximizes the sum of discounted rewards, where a discount factor γ encodes the trade-off between maximizing immediate reward versus long-term reward. Alternately, the problem may also be specified in terms of maximizing the average reward, but we focus on the discounted setting.

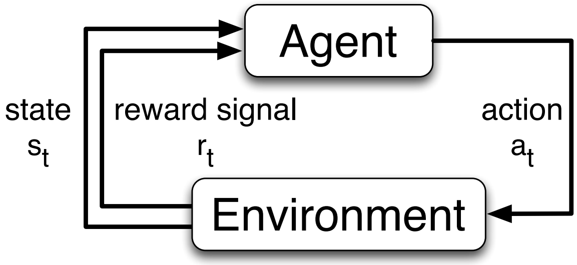


Figure 2.1: The Reinforcement Learning problem setting [Sutton and Barto, 1998b].

Classically, this agent-environment interaction is defined by a Markov Decision Process (MDP) [Bellman, 1957]. An MDP is defined by the tuple $(S, \mathcal{A}, R, P, \rho_0, \gamma)$, with state set S, action set $\mathcal{A}$, reward function $R : S \times \mathcal{A} \times S \to \mathbb{R}$, discount factor $\gamma \in [0, 1]$, and transition probabilities P. For any two states $s, s' \in S$, and action $a \in \mathcal{A}$, $P(s'|s, a)$ gives the probability of transitioning into state s' after taking action a in state s.

When the agent-environment interactions can be broken into independent subsequences the MDP is said to be *episodic*, whereas when this is not the case and interactions form an on-going uninterrupted process, the MDP is said to be *continuing*. The start state (continuing), or the first state of each episode (episodic), is sampled from the probability distribution ρ_0. When the state of the environment is only partially observable by the agent then the interaction is a Partially Observable MDP (POMDP) [Lovejoy, 1991]. When the state set is finite the interaction is said to be a *finite* MDP, and when the state set is a continuous space, such as a subset of $\mathbb{R}^n$, then the interaction is said to be a *continuous* MDP. Similarly, a MDP may be classified as having a finite or continuous action set. In the case of continuous MDPs the transition probabilities are given by a probability density function and $P(s'|s, a)$ should be interpreted as the probability of transitioning into a small neighborhood around state s' after taking action a in state s.

The agent attempts to select actions that maximize the expected sum of discounted rewards, also referred to as the expected *discounted return*:

$$\mathbb{E}[R_t] = \mathbb{E}[\sum_{i=0}^{\infty} \gamma^i r_{t+i+1}].$$

The expected discounted return from state s when following policy π is given by the *state-value function* $V^\pi(s)$, which can also be expressed recursively with the *Bellman equation*:

$$V^\pi(s) = \mathbb{E}_\pi[R_t \mid s_t = s] = \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s].$$

Similarly, the *action-value function* gives the expected discounted return of taking action a from state s and following policy π afterward:

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi[r_{t+1} + \gamma Q^\pi(s_{t+1}, a_{t+1}) \mid s_t = s, a_t = a].$$

From this basic overview of the RL problem, we now turn to a brief review of RL algorithms relevant our work.

2.1.1 Function Approximation

RL algorithms often involve estimating the state-value or action-value functions of some policy. Such an estimation problem involves the use of some form of *function approximation*. A function approximation is a mapping from parameter vectors, also referred to as weight vectors, to functions. The weights of the function approximation are adjusted by the RL algorithm to better approximate some target function.

For finite MDPs with small state and action sets a common choice of function approximation is *tabular*, that is, every state-action pair is represented independently in a look-up table. Tabular function approximation is a special case of *linear function* *approximation*, where the function output is given by a linear combination of *basis functions* which map inputs to real numbers. The weights of the linear combination are the tunable parameters of the function approximation (i.e. the weight vector).

We denote by Q_w the action-value function approximation with weight vector $w \in \mathbb{R}^n$ where $Q_w(s,a)$ gives the approximate action-value for state s and action a. The gradient of $Q_w(s,a)$ with respect to its parameters, w, is the vector of partial derivatives with respect to each of the n parameters (weights): $\nabla Q_w(s,a) \equiv \frac{\partial Q_w(s,a)}{\partial w} \equiv \left[\frac{\partial Q_w(s,a)}{\partial w_0}, \dots, \frac{\partial Q_w(s,a)}{\partial w_{n-1}}\right]$.

2.1.2 Policy Iteration

The *policy improvement theorem* [Sutton and Barto, 1998b], states that for any pair of deterministic policies π, π',

$$Q^\pi(s, \pi'(s)) \geq V^\pi(s),\ \forall s \in S \quad \Longrightarrow V^{\pi'}(s) \geq V^\pi(s),\ \forall s \in S.$$

Definition 2.1. *The* ***greedy policy*** *with respect to an action-value function, $Q : S \times \mathcal{A} \to \mathbb{R}$, chooses any action a in state s such that $a = \arg\max_{a' \in \mathcal{A}} Q(s, a')$.*

By implication, let π be any deterministic policy with action-value function Q^π, then the greedy policy with respect to Q^π will be a *policy improvement* in the sense of the above inequality. This motivates the *policy iteration* algorithm (Algorithm 1) which generates a sequence of policies such that each policy is greedy with respect to the preceding policy's action-value function [Howard, 1960]. At each iteration policy iteration first performs *policy evaluation* to find the action-value function of the current policy, followed by *policy improvement*, which produces the greedy policy with respect to this action-value function. When policy iteration over deterministic policies is applied to finite MDPs the sequence of policies generated is guaranteed to converge to an optimal policy π^*, such that for all policies π: $V^{\pi^*}(s) \geq V^\pi(s),\ \forall s \in S$.

This result requires that at each iteration the action-value function be computed exactly.

Algorithm 1 Policy Iteration

```
Initialize initial policy $\pi_0$ arbitrarily
for $t = 0, 1, 2, \dots$ do
    $Q_{w_{t+1}}(s,a) = Q^{\pi_t}(s,a),\ \forall s,a$                                ▷ Policy Evaluation
    $\pi_{t+1} = \arg\max_{\pi} \mathbb{E}_{s\sim d^{\pi},a\sim\pi}\left[Q_{w_{t+1}}(s,a)\right]$   ▷ Policy Improvement
end for
```

When the action-value function can only be represented approximately, such as in the case of continuous MDPs, the guaranteed policy convergence no longer holds. Bertsekas and Tsitsiklis [1996] proved that if the approximation errors are bounded on each iteration approximate policy iteration produces a sequence of policies with state-value functions that converge to a region around the optimal state-value function. Least-Squares Policy Iteration (LSPI) is an approximate policy iteration algorithm that uses the method of least-squares to compute the approximate action-value function from a set of examples generated under the current policy [Lagoudakis and Parr, 2001, 2003].

2.1.3 Optimistic Approximate Policy Iteration

Optimistic approximate policy iteration extends approximate policy iteration to allow the policy improvement step to be performed before policy evaluation has fully converged [Tsitsiklis, 2003]. The Sarsa(λ) algorithm—so named because its update depends on the tuple (s, a, r, s', a') where $s, s' \in S$, $a, a' \in \mathcal{A}$, and $r = R(s, a, s')$—can be interpreted as an optimistic approximate policy iteration algorithm. Sarsa(λ), given by Algorithm 2, may also be understood as a Temporal Difference (TD) learning bootstrapping algorithm [Sutton and Barto, 1998b]. That is, it uses the current action-value estimates Q_t to estimate the λ-return:

$$R_t^{\lambda} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_t^{(n)}, \text{ where}$$
$$R_t^{(n)} = \sum_{i=1}^{n}\gamma^{i-1}r_{t+i} + \gamma^n Q_t(s_{t+n}, a_{t+n}).$$

Minimizing the error between Q_t and R_t^{λ}, while treating the λ-return as an independent random variable, leads to the Sarsa(λ) algorithm and the concept of *eligibility traces* [Sutton and Barto, 1998b]. An eligibility trace is the exponentially decaying credit assigned to past states and actions for the reward received on the current time step. The parameter λ represents the eligibility trace decay rate. As $\lambda \to 1$ the λ-return approaches an estimate of the expected discounted return computed through Monte Carlo rollouts of the policy, whereas when $\lambda = 0$ the λ-return error, $R_t^{\lambda} - Q_t(s_t, a_t)$, becomes equivalent to the one-step Bellman error:

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t).$$

Algorithm 2 Sarsa(λ)

```
Given MDP $M = (S, \mathcal{A}, P, \rho_0, R, \gamma)$, and policy class $\pi$
Initialize $\lambda \in [0, 1)$, $\{\alpha_t \in (0, 1]\}_{t=0}^{\infty}$, $w_0 = 0$
$s_0 \sim \rho_0(\cdot)$,    $a_0 \sim \pi(s_0; w_0)$
for $t = 0, 1, 2 \ldots$ do
    $\pi_t = \pi(w_t)$
    $r_{t+1} \sim R(\cdot \mid s_t, a_t)$,    $s_{t+1} \sim P(\cdot \mid s_t, a_t)$,    $a_{t+1} \sim \pi_t(s_{t+1})$
                                                        ▷ Sarsa($\lambda$) Update
    $\delta_t = r_t + \gamma Q_{w_t}(s_{t+1}, a_{t+1}) - Q_{w_t}(s_t, a_t)$
    $e_t = \gamma\lambda e_{t-1} + \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w}$
    $w_{t+1} = w_t + \alpha_t \delta_t e_t$
end for
```

Sarsa(λ) plays a central role in this dissertation. Although the methods could be readily applied to a variety of related algorithms we focus on Sarsa(λ) because it is a simple algorithm with few parameters and works well in practice. In particular, notice that Sarsa(λ) requires parameters λ, the eligibility decay rate, $\{\alpha_t\}_{t=0}^{\infty}$, a sequence of scalar valued step sizes, and a policy class π. Throughout this work we will use the ϵ-greedy policy class wherever Sarsa(λ) is involved. The ϵ-greedy policy, over action-value function Q_t, takes a random action with probability ϵ and otherwise follows the greedy policy:

$$\pi_\epsilon(s, a; Q_t) = \begin{cases} 1 - \dfrac{\epsilon(|\mathcal{A}| - 1)}{|\mathcal{A}|} & \text{if } a = \arg\max Q_t(s, \cdot) \\ \dfrac{\epsilon}{|\mathcal{A}|} & \text{otherwise.} \end{cases}$$

Performing intelligent exploration is a large and active area of research in RL [Dearden et al., 1999, Strens, 2000, Duff, 2003, Şimşek and Barto, 2006, Asmuth et al., 2009]. However, despite its simplicity, we use ϵ-greedy to avoid adding additional complexity to our studies and because of its widespread use in the field.

2.1.4 Policy Gradient

Policy iteration and related methods such as Sarsa(λ) attempt to minimize value function error for the current policy, and then update the policy greedily with respect to that value function. In a sense, this is an indirect method of finding an optimal policy, whereas a direct method would be to use stochastic gradient ascent to directly maximize the expected discounted return with respect to the policy.[1]

Let π_θ be a differentiable policy parameterized by weight vector θ and let $R(\pi_\theta, M)$ be the expected discounted return under policy π_θ on MDP M:

$$R(\pi_\theta, M) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 \sim \rho_0(\cdot), M]. \tag{2.1}$$

[1] This is a false dichotomy that is only accurate at the high level in which these algorithms are typically considered. The two methods, policy iteration and policy gradient, appear to be intricately related as primal/dual problems. However, a full exposition on this research direction is off-topic for this work.

Further, let d^π be the long term steady-state distribution over states when following fixed policy π. The *policy gradient* theorem [Sutton et al., 2000] states that for MDP M:

$$\frac{\partial R(\pi_\theta, M)}{\partial \theta} = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta(s)}\left[\frac{\partial \pi_\theta(s,a)}{\partial \theta} Q^\pi(s,a)\right]. \tag{2.2}$$

That is, for a sufficiently small step size α, $\theta + \alpha\frac{\partial R(\pi_\theta,M)}{\partial\theta}$ will result in an improved policy with higher expected discounted return. However, in this form the action-value function, Q^π, must be known exactly. Sutton et al. [2000] further proved that if *compatible* function approximation is used, then the policy gradient theorem holds for $f_w : S \times \mathcal{A} \to \mathbb{R}$ used to approximate Q^π. The function approximation f_w, parameterized by weight vector $w \in \mathbb{R}^n$, is compatible when:

$$\frac{\partial f_w(s,a)}{\partial w} = \frac{\partial \pi_\theta}{\partial \theta}\frac{1}{\pi_\theta(s,a)}. \tag{2.3}$$

This fundamental result of policy gradients helps to explain the *actor-critic* framework which has many features in common with the policy iteration algorithms previously discussed. In actor-critic algorithms the agent is explicitly divided into two components: the *actor* implements a policy, and the critic which estimates the action-value function used to evaluate and update the policy.

2.1.5 Natural Actor-Critic (NAC)

Around the same time that the policy gradient theorems were published another method was introduced to the optimization and machine learning research communities. Amari and Douglas [1998a] proposed that instead of assuming a Euclidean distance metric when performing gradient descent (ascent) a more appropriate distance metric may be the Riemannian metric. Suppose the space of weight vectors exists on a Riemannian manifold such that measuring distance between any two vectors, w and $w + \delta w$ with Riemannian metric tensor $G(w)$ is given by:

$$d^2(w, w + \delta w) = \delta w^\top G(w) \delta w. \tag{2.4}$$

The *natural gradient* descent update, for minimizing a loss function $\mathcal{J}(w_t)$, is the application of steepest descent on a Riemannian manifold:

$$w_{t+1} = w_t - \alpha_t G^{-1}(w_t) \frac{\partial \mathcal{J}(w_t)}{\partial w}. \tag{2.5}$$

In addition to significantly improving the convergence speeds of stochastic gradient descent, natural gradients were shown to be asymptotically Fisher efficient and in some cases could be estimated without the computational cost of explicitly inverting a matrix [Amari, 1998]. This means that, under assumptions typical for stochastic optimization, the variance of the algorithm attains the Cramér-Rao bound in the limit (i.e. is a minimum variance unbiased estimator).

When natural gradients were finally applied to the policy gradient algorithm the policy gradient theorem for function approximation became especially important. When using compatible function approximation the natural policy gradient reduces to the weight vector of the action-value function estimate under the current policy [Kakade, 2002]. The use of natural gradients with policy gradient and actor-critic algorithms led to many variations on the theme [Peters and Schaal, 2008, Bhatnagar et al., 2009, Thomas, 2012]. These are commonly referred to as Natural Policy Gradient (NPG) and Natural Actor-Critic (NAC) algorithms.

When using NAC in experiments we will use NAC-LSTD, which uses the method of least-squares to estimate the action-value function [Peters and Schaal, 2008]. Additionally, we will use the *soft-max* policy for any experiments involving NAC. The soft-max policy with respect to action-value function Q_w, parameterized by weight vector $w \in \mathbb{R}^n$, is given by:

$$\pi_\tau(s, a; Q_w) = \frac{\mathrm{e}^{Q_w(s,a)/\tau}}{\sum_{b \in \mathcal{A}} \mathrm{e}^{Q_w(s,b)/\tau}}. \tag{2.6}$$

The parameter $\tau > 0$ biases the policy to be more stochastic ($\tau > 1$) or less stochastic ($\tau < 1$), where less stochastic means more likely to follow a greedy policy.

2.2 Adaptive Step-Sizes

The general problem of choosing a step size, or an adaptive step-size algorithm, is an important aspect in many methods for optimization. As such there is an abundance of research into adaptive step-size algorithms in a variety of different settings. This dissertation is focused on a much narrower problem, step-sizes in RL algorithms, but because little work has focused on methods solely for RL we will review work on the more general problem of step sizes in methods for unconstrained minimization before discussing more closely related work on adaptive step-sizes for RL.

Let $\mathcal{J} : \mathbb{R}^n \to \mathbb{R}$ be a loss function over weight vectors, $w \in \mathbb{R}^n$, and let A be an incremental algorithm. Given a sequence of step sizes $\{\alpha_t\}_{t=0}^{\infty}$, for each step t algorithm A produces update direction $\triangle w_t \in \mathbb{R}^n$ seeking to minimize $\mathcal{J}(w_{t+1})$ with:

$$w_{t+1} = w_t - \alpha_t \triangle w_t. \tag{2.7}$$

If $\mathcal{J}(w_t)$ is differentiable then $\triangle w_t = -\nabla \mathcal{J}(w_t) + \xi_t$ is a *noisy descent direction*, where noise term $\xi_t \in \mathbb{R}^n$ is mean zero with finite variance and the gradient $\nabla \mathcal{J}(w) \equiv \left[\frac{\partial \mathcal{J}(w)}{\partial w_0}, \cdots, \frac{\partial \mathcal{J}(w)}{\partial w_{n-1}}\right]$ is the vector of partial derivatives. Then, the sequence of weight vectors $\{w_t\}_{t=0}^{\infty}$ is dependent upon the sequence of step sizes $\{\alpha_t\}_{t=0}^{\infty}$. The adaptive step-size problem is to generate a sequence of step sizes $\{\alpha_t\}_{t=0}^{\infty}$, where $\alpha_t > 0$ $\forall t$, which minimize the loss incurred by the learning algorithm at each step t.

With this general form in place, the rest of the chapter separates existing methods into three categories based on the form the step size takes: matrix, vector or scalar. The matrix-valued adaptive step size $\alpha_t \in \mathbb{R}^{n\times n}$ is restricted to be a positive definite matrix. The vector-valued adaptive step size and the scalar adaptive step size are both special cases of this form. A vector-valued step size forms a diagonal matrix with positive entries whereas a scalar step size is a positive multiple of the identity matrix. Following this we discuss the current state-of-the-art in adaptive step sizes for RL.

2.2.1 Matrix-Valued Step Sizes

When $\mathcal{J}$ is differentiable and $\triangle w_t$ is a noisy descent direction then matrix-valued step sizes are equivalent to performing steepest descent with the quadratic norm $\|\cdot\|_{\alpha_t^{-1}}$, or with a change of coordinates given by multiplication by the matrix α_t^{-1} [Boyd and Vandenberghe, 2004]. The choice of α_t impacts the condition number of the problem, so that successful algorithms will tend to compute matrices that give lower condition numbers to the transformed problem. The condition number of a function is a measure of how much the function value changes due to small perturbations of the input value. For matrices this corresponds to how much the inverse of the matrix is affected by small perturbations to the matrix. For an optimization problem the condition number is for the loss function, which in some cases is equivalent to the condition number of a related matrix.

When the loss function is twice differentiable, $\mathcal{J} \in C^2$, and the descent direction is noiseless, then choosing $\alpha_t = H^{-1}(w_t)$, where $H(w_t) = \nabla^2 \mathcal{J}(w_t)$ is the Hessian of the loss function at w_t, results in Equation 2.7 becoming Newton's method (Newton-Raphson method). If the loss function is not quadratic in w then the use of a scalar step size is still required: $\alpha_t = \tilde{\alpha}_t H^{-1}(w_t)$ with $\tilde{\alpha}_t \in (0, 1]$. However, if $\mathcal{J}$ is quadratic, then Newton's method with a step size of 1 is guaranteed to converge in one step.

Furthermore, in the often used case of $\mathcal{J}(w) = \|Aw - b\|_2^2$, where $A \in \mathbb{R}^{n \times m}$ and $b \in \mathbb{R}^m$, Newton's method is equivalent to the standard linear least squares solution:

$$\begin{aligned} w_{t+1} &= w_t + H^{-1}(w_t)\triangle w_t \\ &= (A^T A)^{-1} A^T b. \end{aligned}$$

Applying this same method, but updating H^{-1} recursively using the matrix inversion lemma [Woodbury, 1950, Hager, 1989],

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1},$$

yields the recursive least squares algorithm [Kushner and Yin, 2003]. The recursion is due to maintaining the current inverse matrix A^{-1} and updating its value directly.

Finally, when α_t is chosen to be the inverse of an appropriate Riemannian metric tensor at w_t, Equation 2.7 becomes the update equation for a natural gradient algorithm [Amari and Douglas, 1998b]. As previously discussed, natural gradients have been applied to yield improvements in performance in a variety of optimization settings, including the use of policy gradients for RL [Amari, 2000, Kakade, 2001].

When these matrix-inversion methods for step sizes have access to the true Hessian of the loss function, in the case of Newton's method, or the true metric tensor, in the case of Natural Gradients, they vastly outperform other methods discussed below. However, when these matrices must be estimated from experience, or imperfect estimates are used, it has been shown that much faster adaptive step-size methods can actually outperform some of them [Sutton, 1992a]. An additional drawback to the matrix-valued step-size methods are that they all require at least quadratic time and space complexity per step to compute the adaptive step size.

The closely related *AdaGrad* algorithm provides an adaptive proximal function for subgradient methods [Duchi et al., 2011b]. The full matrix version of AdaGrad produces an α_t equal to the un-normalized matrix square root of that produced by natural gradients. A vector-valued version of AdaGrad results from assuming this matrix is diagonal and, given scalar step size $\tilde{\alpha}_t$, yields the step size at time t for each dimension i:

$$\alpha_{t,i} = \frac{\tilde{\alpha}_t}{\sqrt{\sum_{k=0}^{t} \triangle w_{k,i}^2}}. \tag{2.8}$$

It bears repeating that the matrix-valued adaptive algorithms discussed above generally require an additional tunable scalar step size. The reason is that matrix and vector-valued adaptive step-size methods change both the direction and magnitude of updates given some assumptions about the loss function. However, such methods are often applied in settings where the assumptions do not hold exactly and an additional scalar (step size) is used to limit the magnitude of updates. For example, Newton's method gives a matrix-valued step size for deterministic optimization problems. When the loss function is quadratic both the direction and magnitude provided by Newton's method are optimal, but when the loss function is not quadratic a tunable step size must be used to rescale the magnitude of the update. The differences between adjusting the update direction and rescaling its magnitude are conflated by matrix and vector-valued step-size methods that do not use an additional scalar step size. These differences will become particularly relevant in Chapter 4.

2.2.2 Vector-Valued Step Sizes

This section deals with vector-valued adaptive step-size methods, that is, those in which α_t is a diagonal matrix. We begin by covering the RProp algorithm for training Artificial Neural Networks with Backpropagation [Riedmiller and Braun, 1993], and then cover several algorithms that are all special cases of a general class of adaptive step-size methods based upon the use of stochastic gradient descent.

The RProp algorithm is a heuristic method for choosing between two fixed step sizes, $\eta^+ > 1$ and $\eta^- < 1$. For each weight update, if the derivative with respect to that weight has changed sign from the last iteration to the current iteration then the update is performed with η^-, otherwise the update is performed with η^+. Thus, if $\alpha_{t,i}$ gives the ith component of the step-size vector at step t, then the RProp algorithm produces:

$$\alpha_{t,i} = \begin{cases} \eta^+ & \text{if } sign(\triangle w_{t-1,i}) = sign(\triangle w_{t,i}) \\ \eta^- & \text{if } sign(\triangle w_{t-1,i}) \neq sign(\triangle w_{t,i}). \end{cases} \tag{2.9}$$

While very simple, RProp performs well in practice for training feedforward neural networks [Riedmiller, 1994]. The intuition behind RProp, shared by many adaptive step-size algorithms, is that if the update direction changes sign then the previous update has overshot the local optimum and a smaller step size should be used.

Next, we turn to a class of methods that take a meta-optimization approach to incrementally find the best step size. The general approach is to take the derivative of $\mathcal{J}(w_t)$ with respect to the step sizes $\alpha_{t,i}$ and to do so in such a way as to take into account the incremental dependencies between steps of gradient descent. Schraudolph [1999] gives the generalized algorithm for this class of methods as

$$\alpha_{t+1} = \alpha_t e^{\mu \triangle w_t h_t} \tag{2.10}$$

$$h_{t+1} = h_t + \alpha_{t+1}(\triangle w_t - H(w_t)h_t). \tag{2.11}$$

When the Hessian is replaced by a diagonal approximation, $H(w_t) \approx Diag(\triangle w_t^2)$, the result is the Incremental Delta-Bar-Delta (IDBD) algorithm [Sutton, 1992b, Jacobs, 1988]. With some additional modification to prevent divergence this also produces the Autostep algorithm [Mahmood et al., 2012b]. Another variant, ELK1, is obtained by treating the optimization problem as a normalized LMS problem, taking the same approximation of the Hessian, and wrapping the predictions in a sigmoid function [Schraudolph, 1999]. Among these variants Autostep has been shown to perform the best in two idealized supervised learning tasks, and to have outperformed recursive least squares on the same problems [Mahmood et al., 2012b]. Autostep differs from

IDBD in two ways. First, the exponents in Equation 2.10 are normalized by a running estimate of their maximum value for each dimension in order to prevent updates from getting too large. This is done by replacing h_t in Equation 2.10 with $\frac{h_t}{v_t}$, where $x_{t,i}$ is the input value in the ith dimension of the linear function approximation at step t and:

$$v_{t+1,i} \leftarrow max(|\triangle w_{t,i} h_{t,i}|, v_{t,i} + (1/\tau)\alpha_i x_{t,i}^2 (|\triangle w_{t,i} h_{t,i}| - v_{t,i})).$$

Second, after Equation 2.10 is computed, each step size is divided by the maximum between unity and the step-size weighted square of the inputs,

$$max(\alpha_{t+1}^\top x_t^2, 1).$$

This is to prevent the step sizes from getting large enough to cause function approximation divergence. Also, notice that the momentum term used to train multilayer feed-forward neural networks with back propagation can be considered a type of adaptive step size related to the above algorithms that retain information from past descent directions to adapt the current update.

Finally, the variance-based stochastic gradient descent (vSGD) algorithm is a recently published vector-valued adaptive step-size algorithm for stochastic gradient descent (SGD) on noisy quadratic loss functions [Schaul et al., 2012]. The derivation of vSGD assumes the loss function is given by an expectation over noisy quadratic instantaneous loss functions:

$$\mathcal{J}(w_t) = \mathbb{E}_{j \sim \mathcal{P}} \left[\mathcal{J}_j(w_t) \right]. \tag{2.12}$$

Here, $\mathcal{P}$ denotes the sampling distribution over training examples. Additionally, the Hessian of $\mathcal{J}_j(w_t)$ and the covariance matrix Σ of the mean-zero noise are both assumed to be diagonal matrices, with $h_i = \mathbb{E}[H^j_{i,i}]$ where H^j is the Hessian of $\mathcal{J}_j$. Under these assumptions the vSGD adaptive step size is:

$$\alpha_{t,i} = \frac{1}{h_i} \frac{\mathbb{E}[\triangle w_i]^2}{\mathbb{E}[\triangle w_i^2]}. \tag{2.13}$$

2.2.3 Scalar Step Sizes

Now, consider adaptive step-size methods in which $\alpha_t = \tilde{\alpha}_t I$, where $\tilde{\alpha}_t \in \mathbb{R}^+$ and I is the identity matrix. There has been more research into step-size methods of this form, in part because it is trivial to guarantee that the resulting update is a descent direction and because convergence analysis of these algorithms is more tractable. Additionally, the results of these methods and the analysis of them leads to interesting questions about what it means to be an optimal step size. We start by reviewing the conditions used in convergence analysis of step-size algorithms. With that foundation in place we consider deterministic back off strategies that are not adaptive, then a variety of exact and approximate line search methods, and finally meta-optimization step-size methods. Meta-optimization methods are distinguished from line search in that they do not find the optimal step size at each iteration but instead apply a meta-optimization algorithm to incrementally move towards the optimal step size.

2.2.3.1 Convergence Conditions

Given standard regularity assumptions it has been shown that Equation 2.7 [Kushner and Yin, 2003], with scalar step sizes, will converge with probability one (w.p.1) when $\alpha_t \geq 0$ and

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty. \tag{2.14}$$

These conditions are often used to prove a step-size algorithm's convergence, or cited to indicate the required form of the step sizes used by an algorithm. However, convergence in the limit can still be very slow, and convergence rates for some algorithms will be discussed below. Many line search methods use the following inequalities as a means to indicate when a *sufficient decrease* in the loss function will be satisfied. The Wolfe conditions are satisfied by α_t whenever:

$$\mathcal{J}(w_t + \alpha_t \triangle w_t) \leq \mathcal{J}(w_t) + c_1 \alpha_t \triangle w_t^T \nabla \mathcal{J}(w_t) \tag{2.15}$$

$$\triangle w_t^T \nabla \mathcal{J}(w_t + \alpha_t \triangle w_t) \geq c_2 \triangle w_t^T \nabla \mathcal{J}(w_t), \tag{2.16}$$

where $0 < c_1 < c_2 < 1$ [Wolfe, 1969]. Equation 2.15 is also referred to as the Armijo-Goldstein inequality. An improvement to the Wolfe conditions are given by the strong Wolfe conditions [Nocedal and Wright, 2006], which replace Equation 2.16 with

$$|\triangle w_t^T \nabla \mathcal{J}(w_t + \alpha_t \triangle w_t)| \leq -c_2 |\triangle w_t^T \nabla \mathcal{J}(w_t)|. \tag{2.17}$$

2.2.3.2 Deterministic Step-Size Schedules

A deterministic step-size schedule is a step-size algorithm that does not adapt but instead generates a predetermined sequence of decreasing step sizes. By far the most common step-size method used in practice is to choose a fixed step size, $\alpha_t = \alpha \in (0, 1],\ \forall t \geq 0$. This can be viewed as an adaptive step-size algorithm with one parameter, α, no assumptions placed on the loss function, but correspondingly no convergence guarantees. In fact, one can construct problems for which any fixed step size will perform arbitrarily poorly. For example, Simao and Powell [2009] studied an inventory management problem where the occurrence of rare events causes a fixed step size to be either too large for common events or too small for rare events.

The step-size schedule $\alpha_t = 1/t$ has been proven to be the best linear unbiased estimator of the true mean $\sum_{i=0}^{t-1} \triangle w_i / t$ [George and Powell, 2006b], in the sense that it provides a minimum variance unbiased estimator for stationary data. While this is an interesting theoretical result it does not apply to the RL context because changes to the policy cause the data to come from a non-stationary distribution. It does however meet the convergence conditions from Equation 2.14. This step size is the simplest case of the Generalized Harmonic Step-size (GHS) [George and Powell, 2006b]:

$$\alpha_t = \alpha_0 \frac{a}{a+t-1}, \tag{2.18}$$

where $\alpha_0 \in (0,1]$ and $a \in \mathbb{R}^+$ are meta parameters. While the performance of GHS is very sensitive to these parameter values, with tuning it can perform comparably with much more sophisticated methods, even out performing the bias adjusted Kalman filter [Ryzhov et al., 2012]. McClain's formula can be seen as a combination of constant and $1/t$ decaying step sizes:

$$\alpha_t = \begin{cases} \alpha_0 & \text{if } t = 0 \\ \frac{\alpha_{t-1}}{1+\alpha_{t-1}-a} & \text{if } t \geq 1. \end{cases} \tag{2.19}$$

Initially it will perform similarly to $1/t$, but will eventually converge to the constant step size a [George and Powell, 2006b].

The last deterministic step size we consider is the Search Then Converge (STC) algorithm [Darken and Moody, 1990],

$$\alpha_t = \alpha_0 \frac{1 + \frac{c}{\alpha_0}\frac{t}{N}}{1 + \frac{c}{\alpha_0}\frac{t}{N} + N\frac{t^2}{N^2}}, \tag{2.20}$$

where $\alpha_0 \in (0,1]$ is an initial step size, $N \in \mathbb{N}$ is the threshold past which the step sizes begin to converge, and $c \in \mathbb{R}^+$ scales the rate of convergence. The STC algorithm behaves as its name suggests, early on step sizes are fairly large, and as t becomes larger than N the algorithm begins to converge with c/t.

2.2.3.3 Line Search Methods

Line search methods attempt to find the optimal step size at each step by searching along the line $\{w_t + \alpha_t \triangle w_t | \alpha_t \in \mathbb{R}^+\}$ [Boyd and Vandenberghe, 2004]. Exact line search methods attempt to solve the minimization problem:

$$\alpha_t = \arg\min_{\alpha \geq 0} \mathcal{J}(w_t + \alpha \triangle w_t).$$

Inexact line search methods try to find an approximate solution to this problem. All of the methods considered in this section assume that $\mathcal{J}$ is convex and differentiable.

Backtracking line search [Boyd and Vandenberghe, 2004], Algorithm 3, starts with $\alpha_t = 1$ and then multiplicatively reduces it by a factor β until some convergence conditions are satisfied. Typically the Wolfe or strong Wolfe conditions are used to ensure *sufficient decrease* in $\mathcal{J}(w_t)$.

Algorithm 3 Backtracking line search

```
$c_1 \in (0, 0.5)$, $\beta \in (0, 1)$
$\alpha_t = 1$
while $\mathcal{J}(w_t + \alpha_t \triangle w_t) > \mathcal{J}(w_t) + \alpha_t c_1 \nabla \mathcal{J}(w_t)^T \triangle w_t$ do
    $\alpha_t = \beta \alpha_t$
end while
```

We can also perform an exact line search using Newton's method with respect to α_t [Wen et al., 2012], given by Algorithm 4. This approach performs well compared with other line search methods on a variety of deterministic multivariate functions [Wen et al., 2012].

Algorithm 4 Newton-like exact line search

```
$i = 1$
$a_i = a_{i-1} - \frac{\partial \mathcal{J}(w_t)/\partial \alpha}{\partial^2 \mathcal{J}(w_t)/\partial^2 \alpha}$
while $\|a_i - a_{i-1}\| \geq \epsilon$ do
    $i = i + 1$
    $a_i = a_{i-1} - \frac{\partial \mathcal{J}(w_t)/\partial \alpha}{\partial^2 \mathcal{J}(w_t)/\partial^2 \alpha}$
end while
$\alpha_t = a_i$
```

When the loss function is quadratic the exact line search problem has a closed form solution:

$$\alpha_t = \frac{\triangle w_t^\intercal \triangle w_t}{\triangle w_t^\intercal H(w_t) \triangle w_t}. \tag{2.21}$$

Many researchers have noticed that this "locally optimal" step size tends to overshoot the best value in terms of minimizing sample complexity resulting in slower convergence rates [Yuan, 2008]. Barzilai and Borwein [1988] found a closed form solution to the exact line search problem over a modified loss function . Specifically, they seek to minimize $\|s_{t-1} - \alpha_t y_{t-1}\|_2$, where $s_{k-1} = w_t - w_{t-1}$ and $y_{t-1} = \triangle w_{t-1} - \triangle w_t$, and to make α_t quasi-Newton-like with $\alpha_t^{-1} s_{t-1} = y_{t-1}$. This yields the BB step size

$$\alpha_t = \frac{s_{t-1}^T y_{t-1}}{\|y_{t-1}\|_2^2}.$$

For a quadratic loss function over two variables the BB step size with $\triangle w_t = -\nabla \mathcal{J}(w_t)$ was shown to converge R-superlinearly with R-order of $\sqrt{2}$ [Barzilai and Borwein, 1988, Yuan, 2008]. For more than two variables only linear convergence rates have been proven, but in practice the BB step size often converges superlinearly [Yuan, 2008].

These results are surprising because the optimal step size found by exact line search methods has only been proven to converge linearly, and this matches the results in practice. An explanation is given by Akaike [1959], who proved that the iterates w_t converge into an alternating pattern of directions which are in a two-dimensional subspace, even when $\triangle w_t$ is high dimensional. This gives a theoretical explanation for the often observed zig-zagging convergence pattern observed with these algorithms. The BB step size is able to achieve its super linear convergence rates by preventing this degeneration of the gradient updates into a two-dimensional subspace. Yuan [2008] presents a new step-size based upon the BB step size and gives an empirical method for estimating what Yuan [2008] calls the *decreasing together* property which measures how well the algorithm stays outside of the degenerate two-dimensional subspace. The method given by Yuan [2008] has the same convergence guarantees, and like the BB step size is able to avoid the descent directions from sinking into the lower dimensional subspace:

$$\alpha_t = \begin{cases} \alpha_t^* & \text{if } mod(t,3) \neq 0 \\ \alpha_t^Y & \text{if } mod(t,3) = 0, \text{ where} \end{cases}$$

$$\alpha_t^* = \arg\min_{\alpha} \mathcal{J}(w_t + \alpha \triangle w_t), \text{ using some line search method}$$

$$\alpha_t^Y = \frac{2}{[\sqrt{(1/\alpha_{t-1}^* - 1/\alpha_t^*)^2 + 4\|\triangle w_t\|_2^2 / \|s_{t-1}\|_2^2}] + 1/\alpha_{t-1}^* + 1/\alpha_t^*}.$$

2.2.3.4 Meta Optimization of Step Size

We now turn to methods that apply a meta-optimization algorithm to find the optimal step size, and in this sense they can be viewed as incremental alternatives to the line search methods just discussed. Many algorithms in this section are based on measuring the correlation between successive descent directions (gradient vectors). The simplest such method, RProp, has already been discussed as a vector-valued adaptive step size. RProp examines the correlation of each dimension independently. Kesten's rule is a closely related meta-optimization method, given by:

$$\alpha_t = \alpha_0 \frac{a}{b + K_t}, \text{ where}$$

$$K_t = \begin{cases} t & \text{if } t = 1, 2 \\ K_{t-1} + 1 & \text{if } n > 2 \text{ and } \triangle w_t^T \triangle w_{t-1} < 0 \\ K_{t-1} & \text{if } n > 2 \text{ and } \triangle w_t^T \triangle w_{t-1} \geq 0, \end{cases}$$

where $\alpha_0 \in (0, 1]$ and $a, b \in \mathbb{R}^+$ are positive constant parameters [Kesten, 1958]. Kesten's rule is tunable with parameters (α_0, a, b), and decreases the step size whenever the dot product between the current update and the previous update is negative. This fits with the general intuition, shared among these algorithms, that the correlation of the updates becoming negative indicates that the step size should be smaller because either the process is near the optimal solution or the process is diverging. However, note that Kesten's rule is a non-increasing step-size algorithm.

Magoulas et al. [2001] propose a related step-size algorithm based on the correlation of the updates:

$$\alpha_{t+1} = \alpha_t + K \triangle w_t^T \triangle w_{t-1},$$

where $K \in (0, 1]$ is the meta step-size parameter, with $K = 1$ used for experiments by the authors. This algorithm performed well compared with other adaptive and batch methods for training a feedforward neural network, and converged faster than other methods considered [Magoulas et al., 2001]. However, it is clear that the choice of initial step size, α_0, plays a large role in this algorithm's performance.

Mirozahmedov and Uryasev's (1983) rule [George and Powell, 2006b], takes this same concept but uses an exponential update which allows the step size to increase and decrease based upon the correlation of the updates. For constant parameters $a, \delta \in \mathbb{R}^+$, Mirozahmedov and Uryasev's (1983) rule is given by:

$$\alpha_t = \alpha_{t-1} e^{(a \triangle w_t^T \triangle w_{t-1} - \delta)\alpha_{t-1}}.$$

Next, we look at a straightforward application of stochastic gradient descent as a meta-optimization method for the step size. Kushner and Yin [2003], give the stochastic gradient adaptive (SGA) algorithm for step sizes assuming that $\triangle w_t = \phi_t(y_t - \phi_t^T w_t)$,

$$\alpha_{t+1} = \Pi[\alpha_t + \mu \triangle w_t^T V_t], \tag{2.22}$$

$$V_{t+1} = V_t - \alpha_t \phi_t \phi_t^T V_t + \triangle w_t, \tag{2.23}$$

$$V_0 = 0, \tag{2.24}$$

where $\mu \geq 0$ is a meta step-size parameter and $\Pi[\cdot]$ is some projection onto the acceptable set of step sizes. The projection used by Kushner and Yin [2003] was a truncation to the range $[\alpha_-, \alpha_+]$, where $0 < \mu \ll \alpha_-$.

LeCun et al. claim that the optimal learning rate is the inverse of the largest eigenvalue of the Hessian [Lecun et al., 1993]. Such an approach is also tightly connected to the variations on Equation 2.11. They present an iterative method for estimating the largest eigenvalue of the Hessian in linear time per step given parameters $\gamma, \alpha \in (0, 1)$:

$$\Psi_{-1} = \text{random normalized vector in } \mathbb{R}^n, \tag{2.25}$$

$$\Psi_t = (1-\gamma)\Psi_{t-1} + \frac{\gamma}{\alpha}(\nabla\mathcal{J}(w_t + \alpha\frac{\Psi}{\|\Psi\|}) - \nabla\mathcal{J}(w_t)), \tag{2.26}$$

$$\alpha_t = \frac{1}{\|\Psi_t\|}. \tag{2.27}$$

When used to train a 2-layer feedforward neural network on the NIST database of handwritten digits the adaptive step-size algorithm converged to a step size that was equal to the fixed step size that resulted in the lowest mean squared error [Lecun et al., 1993].

2.3 Adaptive Step-Sizes in Reinforcement Learning

The adaptive step-size problem for RL presents additional challenges not typically present in stochastic optimization. Online reinforcement learning of policies is a fundamentally non-stationary problem. As the policy changes, so too does both the action-value function and the distribution over states visited by the agent. Moreover, the rate at which the problem changes is inherently tied to the learning process itself. Early on, the policy, action-values, and state visitation distribution will all change rapidly, presenting a highly non-stationary problem. As the policy improves the problem will generally become more stationary. Another complication comes from the loss function minimized by RL algorithms. For example, Sarsa(λ) does not minimize any stationary loss function, and thus we must take care when attempting to apply or derive adaptive step-size methods to such algorithms.

Existing research into adaptive step-sizes for RL is peculiarly limited. The existing methods can be divided into two classes: adaptive step-sizes for policy evaluation (stationary policy) and adaptive step-sizes for online control learning. The first is a simpler problem because the policy is held fixed, eliminating the problem of non-stationarity. The second is the focus of this dissertation and provides an adaptive step-size for algorithms such as Sarsa(λ) or policy gradient in which the policy itself is changing throughout the learning process.

2.3.1 Adaptive Step-Sizes for Policy Evaluation

Both IDBD and Autostep were derived for policy evaluation with a fixed policy [Sutton, 1992b, Mahmood et al., 2012b]. The remaining two methods were derived explicitly for approximate dynamic programming and approximate value iteration respectively. First, the Optimal Step-size Algorithm (OSA) [George and Powell, 2006b], also known as the bias-adjusted Kalman filter, begins by using McClain's formula to give a meta-step-size for estimating the mean error and mean-squared error:

$$v_t = \frac{v_{t-1}}{1 + v_{t-1} - \bar{v}} \tag{2.28}$$

$$\bar{\beta}_t = (1 - v_t)\bar{\beta}_{t-1} + v_t(\triangle w_t - w_t) \tag{2.29}$$

$$\bar{\delta}_t = (1 - v_t)\bar{\delta}_{t-1} + v_t(\triangle w_t - w_t)^2. \tag{2.30}$$

These estimates are then used to compute the step size for step t:

$$(\bar{\sigma}_t)^2 = \frac{\bar{\delta}_t - (\bar{\beta}_t)^2}{1 + \bar{\lambda}_{t-1}}, \tag{2.31}$$

$$\alpha_t = \begin{cases} \alpha_0 & \text{if } t = 0 \\ 1 - \frac{(\bar{\sigma}_t)^2}{\bar{\delta}_t} & \text{if } t > 0. \end{cases} \tag{2.32}$$

Then the variance estimate $\bar{\lambda}_t$ and value-function weights w_t are updated by:

$$\bar{\lambda}_t = \begin{cases} (\alpha_t)^2 & \text{if } t = 0 \\ (1-\alpha_t)^2\bar{\lambda}_{t-1} + (\alpha_t)^2 & \text{if } t > 0, \end{cases} \tag{2.33}$$

$$w_t = (1-\alpha_t)w_{t-1} + \alpha_t \triangle w_t. \tag{2.34}$$

This algorithm uses v_t as a meta-step size, with v_{-1} its initial value and $\bar{v}$ the target meta-step-size value, which is updated following McClain's rule. George and Powell [2006b], give results comparing error percentages of OSA with the STC and $1/t$ step-size methods. These results are for estimating value functions in two RL domains and show that OSA is able to lower prediction error faster and to smaller asymptotic values than the other methods. The first domain is a simplified *batch replenishment* problem in which the agent must order products to meet varying demands. The second domain is called *nomadic trucker*, which is very similar to the classic RL Taxi domain [Dietterich, 1998].

The Optimal Step-size for Approximate Value Iteration (OSAVI) algorithm extends the ideas of OSA to the approximate value iteration setting by removing the assumption of sequential independence [Ryzhov et al., 2012]. OSAVI has no tunable parameters, and outperformed McClain's rule, the Harmonic step size, and OSA in its ability to minimize value function approximation error. The only exception was a highly tuned Harmonic step size that was able to perform equivalently well on one of the simpler domains considered.

The OSA and OSAVI algorithms follow an approach taken by vSGD, and used in Chapter 4, in which the step size is derived with respect to the expected value of some unknown variables and standard stochastic approximation methods are used to estimate the value of these variables.

2.3.2 Adaptive Step-Sizes for Online Control

Recently there have been efforts towards an adaptive step-size for policy gradient algorithms [Matsubara et al., 2010, Pirotta et al., 2013]. The adaptive natural policy gradient (aNPG) algorithm normalizes natural policy gradient updates by the squared Mahalanobis norm of the policy gradient with respect to the Riemannian metric tensor, but it requires an additional tunable scalar step size [Matsubara et al., 2010]. This research is asking the same important questions as addressed in this dissertation. However, our focus is on the optimistic approximate policy iteration class of algorithms, specifically Sarsa(λ), while their work is on policy gradient algorithms.

Perhaps the only existing work on this particular topic is Hutter and Legg's (2008) HL(λ) algorithm. HL(λ) is a parameter-free adaptive step-size algorithm for temporal difference (TD) learning, with a similarly named version for online control learning with Sarsa(λ). The intuition behind the algorithm comes from observing that TD-learning is essentially propagating information backward through recently visited state-actions. The algorithm maintains a visitation count for every state-action pair and the authors derived the optimal step size in terms of these visitation counts and the eligibility traces [Hutter and Legg, 2008]. HL(λ) is only applicable to finite MDPs, and the derivation uses the squared λ-return error loss function. Recall that this is the same loss function which motivates the derivation of Sarsa(λ). The HL(λ) step size is computed by:

$$\delta_{i,j} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases} \tag{2.35}$$

$$N_s^{t+1} = \lambda N_s^t + \delta_{s_{t+1},s} \tag{2.36}$$

$$E_s^{t+1} = \lambda\gamma E_s^t + \delta_{s_{t+1},s} \tag{2.37}$$

$$R_s^{t+1} = \lambda R_s^t + \lambda E_s^t r_t \tag{2.38}$$

$$\alpha_t = \frac{1}{N_{s_{t+1}}^t - \gamma E_{s_{t+1}}^t} \frac{N_{s_{t+1}}^t}{N_s^t}. \tag{2.39}$$

HL(λ) was able to outperform TD(λ) with a hand-tuned fixed step size on a random 50 state MDP and a 21 state non-stationary MDP. HLS(λ), the extension of HL(λ) to action-value functions, was able to outperform Sarsa(λ) with a hand-tuned fixed step size on the Windy Gridworld domain [Hutter and Legg, 2008]. When using HL(λ) in experiments we will exclusively use HLS(λ). HL(λ) is one of the few parameter-free adaptive step-size algorithms, but it should be noted that the optimal values for the λ parameter tend to differ substantially between it and the non-adaptive algorithms upon which it is based.

2.4 Summary

While there is an extensive body of research on adaptive step-size algorithms, the vast majority of the work focuses on the stationary and unconstrained optimization context and is not directly applicable to RL. The research on adaptive step-sizes for RL has clear areas that demand further study. Specifically, adaptive step sizes derived in terms of expected values of measurable values are a promising direction that has not been very well explored outside of finite MDPs. On the other hand, there has been an abundance of research into adaptive step-size algorithms that rely primarily on estimating the diagonal of the Hessian matrix and using this in some form as the step size. Unfortunately, these methods have the drawback of squaring the condition number of the meta-optimization problem and thus require careful parameter tuning or special heuristics to prevent divergence.

CHAPTER 3

EVALUATION METHODS FOR REINFORCEMENT LEARNING

In studying the adaptive step-size problem for RL we seek to improve empirical performance of RL algorithms while eliminating a tunable parameter. However, current methods for evaluating RL algorithms give such an occluded perspective on the performance of algorithms that the benefits of eliminating tunable parameters cannot be empirically verified. Therefore, as a prerequisite to any study of adaptive step sizes one must address the issue of empirical methods in RL and how to go about evaluating RL algorithms. Our contributions in this chapter are an illustration of the limitations of current practices and a proposed set of methods for conducting and presenting experiments that provide more informative measures of the performance of RL algorithms.

3.1 Introduction

RL is studied and used in a variety of ways and for a variety of purposes. Arguably one of the most fundamental objectives in the study of RL is that of *optimal control*, learning to control an unknown system in order to maximize a measure of long term reward, where the reward function encodes aspects of the agent's desired performance. In this chapter, we are concerned with the empirical evaluation and comparison of RL algorithms for optimal control based entirely on their observed behavior on suites of problems. That is, we take an empirical view of evaluating RL algorithms and thus do not concern ourselves with internal (to the agent) measures of learning progress, but instead rely only on what is observable externally for any online RL algorithm.

Not all researchers or users of RL will care about this topic. For those whose uses for RL are restricted to the minimization of the value function error, perhaps for the purposes of learning predictive features [Modayil et al., 2012], these methods are not for you. Similarly, this chapter is not intended for the many researchers for whom the most relevant evaluation criterion for RL algorithms is fidelity to biology. However, for any reader using or studying RL algorithms who would prefer an algorithm that achieves higher reward rather than lower, and for whom optimal control is primary among objectives, this chapter proposes a new set of evaluation methods and shows why current standard practices can lead to spurious conclusions.

Due to the continued growth of the field of RL the number of algorithms has increased to the point where an individual researcher cannot experiment with all of them. Similarly, a non-researcher is likely to be overwhelmed by the number of potential methods available for use. To facilitate the decision of which algorithms to invest time in, the field needs methods that thoroughly and accurately describe the performance of algorithms. Unfortunately, the methods currently in widespread use for evaluating RL algorithms do not accomplish these goals.

The field of supervised learning has moved through the same problem of having common evaluation practices that distorted, instead of elucidating, the performance characteristics of learning algorithms. Cohen [1995], for instance, observed that out of a survey of 150 papers published in the *Proceedings of the Eighth National Conference on Artificial Intelligence* (1990), "only 42 percent had suggested a program had run on multiple examples; just 30 percent demonstrated their performance in some way." Such limited empirical studies limit the progress of a research field and make applying a particular algorithm a frustrating and uncertain process.

By way of comparison, we have done a survey of recently published RL research papers in top conferences (NIPS, ICML, and AAAI 2013) and found that, out of 40 papers on RL, 23% contained no empirical results or evaluations. Although this in itself is only evidence of a strong theoretical component to the RL research community, we found that among those papers that reported empirical results a small minority of them did so with rigorous experimental methods. Empirical studies in these papers tended to be fairly disjointed from one another and difficult to compare due to a lack of hypothesis testing and standardized benchmarking domains or procedures.

3.1.1 Demonstration vs. Experimentation

Empirical methods are questions posed to and answered by nature. The questions one seeks to answer must necessarily inform the methods one uses. The first question about any new algorithm will often be "Does it work?" The answer to this question can be given by a demonstration of the algorithm. A demonstration does not require statistical significance, nor a large number of trials. Instead, a demonstration is practical proof that an algorithm does work in a particular example. Demonstrations answer this question with broad strokes, "yes", "no", "not very well". A demonstration does not tell us how well an algorithm works on average for a particular domain, nor does it provide any information about how well the algorithm will work on different domains. Those questions are answered, or at least informed, by experiments.

In this chapter we study the question "How well does an algorithm work?", and later the question "Does algorithm A work better than algorithm B?", by analyzing the failure rate, policy percentile, and difficulty/impact of parameter tuning on some set of domains. These empirical methods are designed for experimental use to make the questions clearer and to make arriving at answers by interpreting the data less error prone. Some interesting domains are currently too computationally intensive to use for experimental purposes in short timeframes. Such domains may still provide interesting demonstrations, but cannot reasonably be used to answer our quantitative questions.

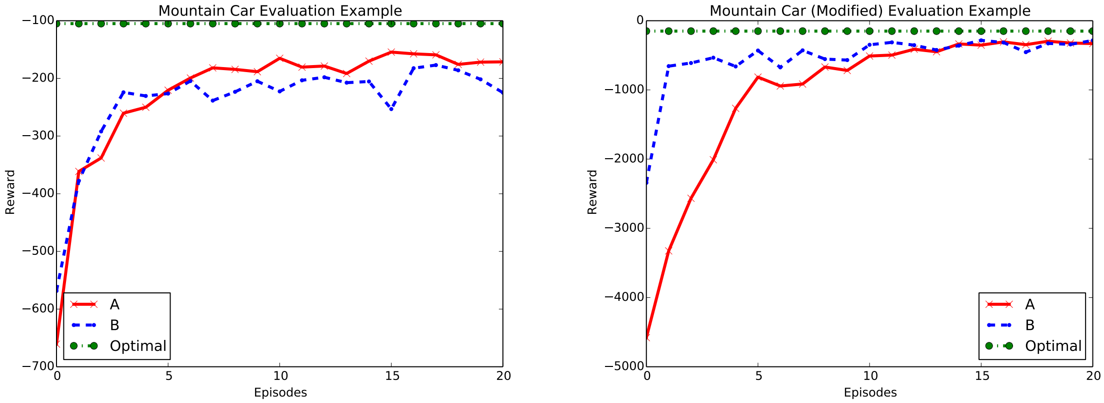
If the goal is to study RL algorithms and their behavior, then one must be equiped with the tools of scientific inquiry and empirical observation. We first review current standard practices in evaluating RL algorithms, then discuss their limitations, and give examples that illustrate how these limitations can lead to spurious results. Next, we propose a set of empirical methods for evaluating and comparing RL algorithms and explain the motivation for each. Finally, we discuss how these methods can be used in practice and demonstrate their use in two case studies.

3.2 Current Methods of Evaluating Reinforcement Learning Algorithms

3.2.1 Learning Curves

At present the standard way that empirical results are presented for RL algorithms is through the use of learning curves. Typically, figures show the total discounted, undiscounted, or average reward versus number of time steps, episodes, or CPU time used for training. These curves attempt to capture two important aspects of the learning algorithm's behavior: (1) how quickly (measured by time steps, episodes or CPU time) the algorithm improves performance, and (2) the total discounted, undiscounted, or average reward achieved by the final policy after the algorithm has been fully trained.

One well known improvement to the standard learning curve is the use of error bars to indicate confidence intervals, or alternately the standard error of the sample returns. Out of the surveyed papers containing empirical studies, only 42% included any form of error bars or otherwise gave information about the variance of an algorithm's performance. Another related and well known method that is rarely used in practice (7%) is hypothesis testing on the results shown in the learning curves. These two methods allow a reader to gauge the significance of the experimental results presented. Without them learning curves themselves are much less informative.



(a) Mountain Car

(b) Modified Mountain Car

Figure 3.1: Comparing two algorithms (A and B), on Mountain Car and a modified Mountain Car, illustrating possible ceiling effects obscuring results. Learning curves for each algorithm are averages over 30 runs.

These methods may help determine when the differences in performance are significant, but when the algorithms being compared all learn near-optimal policies *ceiling effects* can further hamper effective comparisons. A ceiling effect is when algorithms come close to reaching the maximum expected performance possible for some domain, making it impossible to determine if two algorithms perform significantly differently [Cohen, 1995]. The existence of ceiling effects becomes much harder to identify when the performance of an optimal policy is unknown. Only 10% of the papers surveyed compared the performance of algorithms with an estimate of an optimal policy's performance.

For example, consider Figure 3.1a which shows a comparison of two RL algorithms on the standard Mountain Car domain (used in 5% of papers surveyed). Here we have included a line indicating the expected discounted reward of the optimal policy, but such a line is rarely shown in practice. Observe that both algorithms learn policies that are very close to optimal, but that algorithm A appears to perform better than algorithm B. As is, this would often be presented as evidence in support of algorithm

A's superiority. However, the difference is not significant at any reasonable threshold ($p > 0.3$). Results like this have been used as evidence for A's superiority, but in fact the current experiment provides no such evidence in favor of either algorithm. It fundamentally tells us nothing.

That said, considering how close both algorithms' performances are to optimal it is possible that there are ceiling effects at work. The *theoretical performance* of an optimal policy is the performance (e.g. discounted return) when an optimal action is taken at every step, and as such gives a least upper bound on the performance of any policy. However, this upper bound may be overly optimistic and unachievable in practice when an agent uses function approximation and/or exploratory actions. When either of these are used the best achievable performance of an RL agent may be much lower than the theoretical upper bound, making ceiling effects more challenging to identify.

Consider what happens if we modify the Mountain Car domain in order to make it more challenging. Many changes to the domain could have this effect, but for this example we lower the maximum velocity in the direction of the goal and add noise to the transition and reward functions. Figure 3.1b shows the results on this modified domain. There is now a significant difference in performance on average between the algorithms (p $<$ 0.001), and interestingly algorithm B turns out to yield superior performance.

What this example illustrates is that standard practices for using learning curves can obscure the relative performance of algorithms and lead to incorrect conclusions. Even going with the extra step of hypothesis testing, this methodology can be handicapped by ceiling effects due to the domain choice.

3.2.2 Parameter Tuning

Parameter tuning is the dirty secret of RL research. Often the process used for parameter tuning is never mentioned. 87% of the papers with empirical results in our survey gave no mention to how parameters were chosen. Unfortunately, the way parameter tuning is performed and reported in RL publications is almost as uninformative as providing no information to begin with. Two methodologies are commonly used for parameter tuning and for presenting the results of tuning in RL research papers.

The first approach is to manually test each algorithm with a relatively small collection of parameter values and report the best results found.[1] This can introduce an unintended bias in an algorithm's favor because researchers have more insight when selecting parameter values for methods with which they are familiar. Additionally, this manual optimization does not provide information about the behavior of the algorithm for any but a small number of parameter values. It is unlikely that the results from this small number of samples reveals an algorithm's true performance. This method is generally referred to as *hand tuning* of the parameters and was seen in 10% of the surveyed papers.

The second approach is to perform a large parameter optimization procedure for each algorithm and to report the best results found. This approach does not accurately capture the difficulty of finding good parameter values. For instance, it could be that finding good parameter values is more difficult than finding an optimal policy. It also provides no information about the performance of the algorithm using

[1] For our purposes parameter values refer to the values that control the behavior of the algorithm, and not to the values the algorithm controls such as the weights of a function approximator. For example, the step size, eligibility trace decay rate, and exploration rate are common parameters whose values control the behavior of the RL algorithm. These are sometimes referred to as hyper-parameters in other contexts.

any but the best parameter values. We refer to this method as *meta-optimization* of the parameters, and it was seen in 7% of the surveyed papers.[2]

The fundamental problem with both approaches is that the sensitivity of the algorithm to its parameter values is ignored. In the first approach this results in biased evaluations. In the second approach it causes only the combination of the RL algorithm and parameter optimization to be evaluated, which allows the parameter optimization to compensate for weaknesses in the RL algorithm.

To make the point clearer consider the following example: We have developed a new RL algorithm, *evolutionary Sarsa*, which is identical to Sarsa(λ), with one exception. The initial action-values are made to be tunable hyper-parameters so as to simulate some bias provided by evolution. Using the meta-optimization approach to parameter tuning the difficulty of finding these initial values is entirely hidden. Given that the parameter optimization is not treated as part of the algorithm, evolutionary Sarsa appears to consistently outperform Sarsa(λ) and often appears to find a near-optimal policy within a couple of episodes. With the current methods for performing and presenting experiments on RL algorithms one might incorrectly conclude that evolutionary Sarsa is an incredible breakthrough. This is an extreme case, but to a lesser degree this is precisely what happens with current methods in RL. Some researchers provide the parameter values used, but this does not change the fact that the view of the performance of the algorithms is still biased and occluded.

Recall the experimental results in Figure 3.1b comparing algorithms A and B on a modified version of Mountain Car. We concluded that algorithm B is the preferred algorithm, but like many researchers we did not present information about the parameter tuning process used for each algorithm. In fact, the results shown for algorithm A are for the best parameters found out of 10 configurations tried by hand, and the

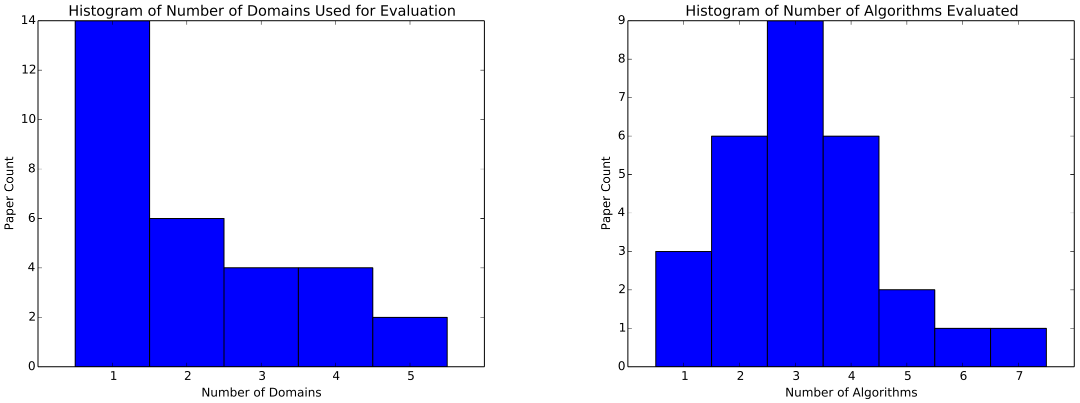
[2] One paper used hand tuning for one algorithm and a meta-optimization method for another.

results shown for algorithm B are for the best parameters found out of a randomized parameter search that considered $100,000$ different configurations. With this new information, are we still confident that algorithm B is superior to algorithm A?

3.2.3 Multiple Domains

Of the research papers surveyed that contain empirical results, almost half contain only a single domain (45%). In general, across RL research papers the number of domains considered in each paper is far too low to make real claims about the generalization of the reported behavior (see Figure 3.2a). Additionally, we observed that there is no commonly used set of benchmarks that are present in most papers, making direct comparison of results in different papers nearly impossible. As one can infer from Figure 3.3, which shows how many papers in which the most frequently used domains appeared, the likelihood of any two papers containing empirical results for the same domain is very small. On the one hand, it is promising to see a variety of different RL domains being used in research. But on the other hand, without some overlap in domains comparing research results requires independently implementing each algorithm in question.

The importance of evaluating algorithms on multiple domains may be made clearer by adopting the language of supervised learning. When a trained classifier is evaluated on a test set we assume that the test set is a representative sample from an underlying population, and we assume that the accuracy we care about is with respect to that population. In RL, experimental domains comprises the test set upon which we evaluate the learning algorithm. Consider how much information is obtained with a test set containing three examples. No matter how interesting these few examples may be, it is very unlikely that they are a representative sample of the underlying population of interest. Similarly, few would argue that any three domains in Figure 3.3 provide a representative sample of the population of control tasks for which we want



(a) Number of domains used

(b) Number of algorithms compared

Figure 3.2: Frequency histograms over empirical studies contained by RL research papers (NIPS, ICML, and AAAI 2013).

RL to be effective. Another way to view this is in the framework of predictive power. If one observes superior performance of an algorithm on three domains (e.g., Mountain Car, Cart Pole, and GridWorld) how well can one predict the performance on a fourth domain drawn randomly from the remaining domains in the table? How about for some new previously unstudied control problem? To what degree does this vary depending on the set of domains used for evaluation? The use of multiple domains in RL experiments is absolutely vital to accurately evaluate an algorithm's performance. Moreover, there must be a variety of domains in every experiment for empirical results to be predictive of performance in general.

3.3 Meaningful Empirical Methods

3.3.1 Background and Desired Properties

Here we propose empirical methods for use in RL research that address the issues discussed above. We approach this by asking two questions. First, what aspects of the algorithm's behavior should be measured? Second, how should the measurements be compared across different algorithms?

Domain	#	Domain	#
Grid World	3	7-Link Reaching	1
Cart Pole	3	BlackJack	1
Chain MDP	3	DAS1	1
Box Pushing (POMDP)	3	Frogger	1
Tiger (POMDP)	2	Helicopter Hover	1
Recycling (POMDP)	2	Hidden Fork (POMDP)	1
Planar Swimming	2	HIV Treatment	1
Mountain Car	2	Pinball	1
Elevator Control	2	Sys-Admin	1
Inventory Control	2	Tag	1

Figure 3.3: Selection of domains used in recent RL research papers

Dependent	Independent
State visitation	Time step
Actions selected	Episodes
Reward received	CPU Time
Memory used	Parameters evaluated

Table 3.1: Observations about the behavior of an RL algorithm

As mentioned earlier, this chapter focuses on empirically evaluating the behavior of an algorithm, and not on any intermediate aspect such as the value function error. With this restriction, the set of possible observations that can be made about the algorithm's behavior with respect to a domain is limited to those listed in Table 3.1. These are simply the raw observable events and how they are measured is what really matters. The dependent (observable) variables in Table 3.1 are affected by the agent's behavior and can be measured with respect to variables that are independent of the agent's behavior. For example, one measurement of state visitation is the number of states visited per episode, which in a shortest-path problem measures the quality of the learned policy.

An experiment performed for exploratory purposes might consider measurements such as the state-visitation trace. Because our focus is on choosing which algorithm to use, or which algorithm to invest time in implementing, we focus on measuring how well an algorithm achieves the principle RL objective, that of accumulating reward. Thus, we primarily focus on the dependent variable of reward received with respect to the different independent variables. Even with this restriction (to only one of the four dependent variables), there are many possible measurements. To motivate this choice we propose some high-level aspects of performance that we would like to capture.

1. *Failure rate.* When, and how often, does the algorithm perform no better than random? (worst case behaviors)

2. *Optimized learning speed.* When using optimized parameters, how quickly does the algorithm improve the rate at which it accrues reward?

3. *Asymptotic policy performance.* What is the policy performance after the algorithm has received ample training and parameter optimization? (best case behaviors)

4. *Difficulty of parameter tuning.* How hard is it to find parameters for the algorithm that achieve good performance?

5. *Impact of parameter tuning.* How much does performance improve when parameter tuning is used? (best vs. average case)

6. *Generalization and variation over domains.* How well does the algorithm's performance generalize across several domains?

Additionally, methods should not be easily biased by researchers' familiarity with their own algorithms (e.g., knowing a tight range for the parameter values for their own algorithms, but not others). That is, measurements should be *objective.* In the next section we present methods for measuring these aspects of an algorithm's performance. We then present methods for comparing two or more RL algorithms that aid in making predictions about which algorithm will perform better on untested domains.

3.3.2 Failure Rate: Quantifying when things fall apart

Judging by research publications, one could easily come to the conclusion that studying when and how algorithms fail to perform well has no research value. However, this would be a terribly mistaken conclusion. One of the principle pursuits of scientific research is to explain variance in observation [Cohen, 1995]. Failure of a learning algorithm dramatically increases variance in reward received, and thus provides an important measurement for any empirical study.

We consider a run of an algorithm as *failed* if the algorithm receives, on average, no more reward than the uniform random policy, or if the algorithm crashes (e.g., due to function approximation divergence). In such cases the learning algorithm fails to provide any advantage over a random policy.

Then the *failure rate* of an algorithm on a domain is the probability of the algorithm failing on that domain when algorithm parameters are drawn uniformly at random from the predefined ranges.

Definition 3.1. *The parameter space, Φ, is a possibly infinite set of parameter vectors of length d.*

Definition 3.2. *For some MDP M, a* ***history*** *of length T is a sequence of state, action, reward tuples, $(s_t, a_t, r_t)\}_{t=0}^{T-1}$ to which M assigns a non-zero probability. The set of all possible histories on M is denoted $\mathcal{H}(M)$.*

Definition 3.3. *Let M be an MDP with action set $\mathcal{A}$. An* ***algorithm*** *is a function mapping histories to actions in M and is denoted by the tuple (A, Φ), where Φ is a parameter space and A is a function such that for any $\phi \in \Phi$, $A_\phi : \mathcal{H}(M) \to \mathcal{A}$.*

Definition 3.4. *Let U indicate the uniform random policy that selects actions with equal probability. For any algorithm (A, Φ) and MDP M, denote by $R(A_\phi, M)$ the* *random variable given by the discounted return achieved by A_ϕ on M. The* ***failure threshold*** *for M is the expected discounted return of the uniform random policy:*

$$\mathbb{F}_M = \mathbb{E}[R(U, M)].$$

The ***failure rate****, $Pr_{\phi \in \Phi}(R(A_\phi, M) \leq \mathbb{F}_M)$, is the probability that the algorithm, with parameters drawn randomly from Φ, achieves a discounted return on M that is no greater than the expected discounted return of the uniform random policy.*

Analyzing the variation of the failure rate across multiple domains may provide additional insight into an RL algorithm's behavior. As an illustrative example, Table 3.2 gives the failure threshold for some common RL domains. Computing the failure rate of an algorithm requires running a large randomized parameter search and counting the number of times the discounted return falls below the failure threshold. In practice the failure threshold can also be used during parameter tuning to cut short trials that are unsalvageable.

3.3.3 Reward Received

Measuring the reward received using learning curves, which express information about both learning speed and asymptotic policy performance with tuned parameters, is currently the most common method used to report on RL experiments. However, as we showed previously, the way learning curves are frequently used makes them substantially less informative. In this section, we propose the *policy percentile* transformation of the reward received which produces more informative learning curves with intuitive interpretations and is compatible with the long established practice of using error bars and hypothesis testing.

We first turn to the issue of whether to show discounted or undiscounted return. The discount factor is generally considered to be part of the domain specification, as

Domain	Failure Threshold	Standard Error
50-Chain	0.180775	±0.0022
Acrobot	-2095	±25
BicycleRiding	1.38858×10^{-3}	$\pm 6.3939 \times 10^{-9}$
BlackJack	-2.79	±0.02
BlocksWorld	0.9374	±0.0006
CliffWalk	-1.004	±0.0007
Cart Pole (Balance, 1-pole)	33.46	±0.12
Cart Pole (Balance, 2-poles)	14.97	±0.065
Cart Pole (Swing Up)	3.593	±0.004
GridWorld 10x10	0.3189	±0.004
HIV Treatment	1.9×10^{6}	$\pm 1.4912 \times 10^{4}$
MountainCar	-41875	±1198
PuddleWorld	-9483	±119
Planar Swimmer	-1636.77	±0.545
SysAdmin	71.78	±0.35

Table 3.2: Failure thresholds on discounted return (estimated over 3000 trials)

it determines which policies are optimal. However, it is quite common to see different discount factors used for the same domain, and this value is sometimes treated as an additional parameter of the algorithm. This confuses the problem of maximizing discounted return with maximizing undiscounted return using a discount factor to bootstrap the process. Smaller discount factors are known to accelerate convergence in some situations [Bertsekas and Tsitsiklis, 1996]. Therefore, an algorithm could conceivably use a smaller discount factor to bootstrap the learning process or as an approximation for solving the harder (larger discount factor) problem. This is akin to making simplifying assumptions that may not hold in practice. However, performance should be reported with respect to the original problem and not the approximation used in its solution.

Like many other issues, this confusion makes reproducing and comparing research results much more challenging. We stick with the view that the discount factor is a parameter of the domain, and thus should be seen as fixed from the agent's or meta-optimization's perspective. Therefore, in the domain specification one should endeavor to always provide the fixed discount factor used, and to be consistent with that value over all experiments.

We have chosen to focus on the objective of maximizing discounted return, but some algorithms instead attempt to maximize the average reward. This is completely consistent with the arguments we present, but the average reward objective presents a different optimization problem than the discounted return objective. The two problems can be related through the concept of *Blackwell optimality*. A policy π is *Blackwell optimal* if there exists $\gamma_0 \in (0, 1)$ such that π is optimal for all discount factors $\gamma \in (\gamma_0, 1)$, and every Blackwell optimal policy is also optimal for the average reward objective [Blackwell, 1965, Hordijk and Yushkevich, 2002].

With this in mind, the only consistent choice for reporting reward received is to do so with respect to the discounted return of the MDP because doing otherwise reports performance on a problem different from the one being used for learning. However, for discount factors less than one, the difference between policies may become most apparent only after many time steps, and the use of discounted return makes these differences much harder to discern. For example, if the goal is t steps away then the difference between an optimal policy and a pessimal one shrinks with γ^{t-1}.

One way to view this is to consider the set of all possible stationary policies for an MDP M. Let Π_M denote this set. RL can be viewed as an optimization problem over this set of policies with respect to the discounted return. If all policies result in the same discounted return, then the optimization problem is trivial because every policy is optimal. In contrast, a very challenging problem is one in which the vast majority of policies offer very low discounted return and very few policies achieve near-optimal discounted return. Thinking along these lines one may start to think of an MDP in terms of the distribution of discounted returns over the set of possible policies. The shape of this distribution would reveal, to some degree, the difficulty of the MDP. There are of course still other aspects that affect the difficulty of the learning problem, such as the state and action space dimensionality, the smoothness of the optimal value function, and the sensitivity of the learning problem to changes in the MDP.

Figure 3.4 shows histograms of undiscounted and discounted ($\gamma = 0.9999$) returns for the uniform random policy for the Mountain Car domain. Undiscounted return is simply the sum of rewards. The difference in discount factors has a substantial impact on the distribution of returns and similarly affects the mean of that distribution. In Mountain Car the undiscounted returns form an approximately log-normal distribution, and the use of a discount factor causes all episodes of length greater than 1000 to appear to be of similar value. This is simply the nature of a discount factor, which affects the RL problem being solved. However, in both cases what one looks for in an RL algorithm is the ability to move the policy towards the far right end of the distribution, to find a policy in the top percentile of all possible policies. We can make this concept more concrete with the following definition:

Definition 3.5. *Let M be an MDP with a set of possible policies Π_M. Then, the* ***policy percentile*** *of a policy $\pi \in \Pi_M$ is the probability that a policy sampled from Π_M will obtain lower expected discounted return than achieved by π. Let F_{Π_M} be the cumulative distribution function of discounted returns over policies Π_M, and let $R(\pi, M)$ be the expected discounted return of the fixed policy π on M. Then the policy percentile of π on M is given by:*

$$\rho(\pi, M) = F_{\Pi_M}(R(\pi, M)). \tag{3.1}$$

We propose the use of an algorithm's expected policy percentile as the dependent variable, replacing discounted or undiscounted return, in learning curves. The first advantage, which we have motivated thus far, is that the policy percentile is easier to

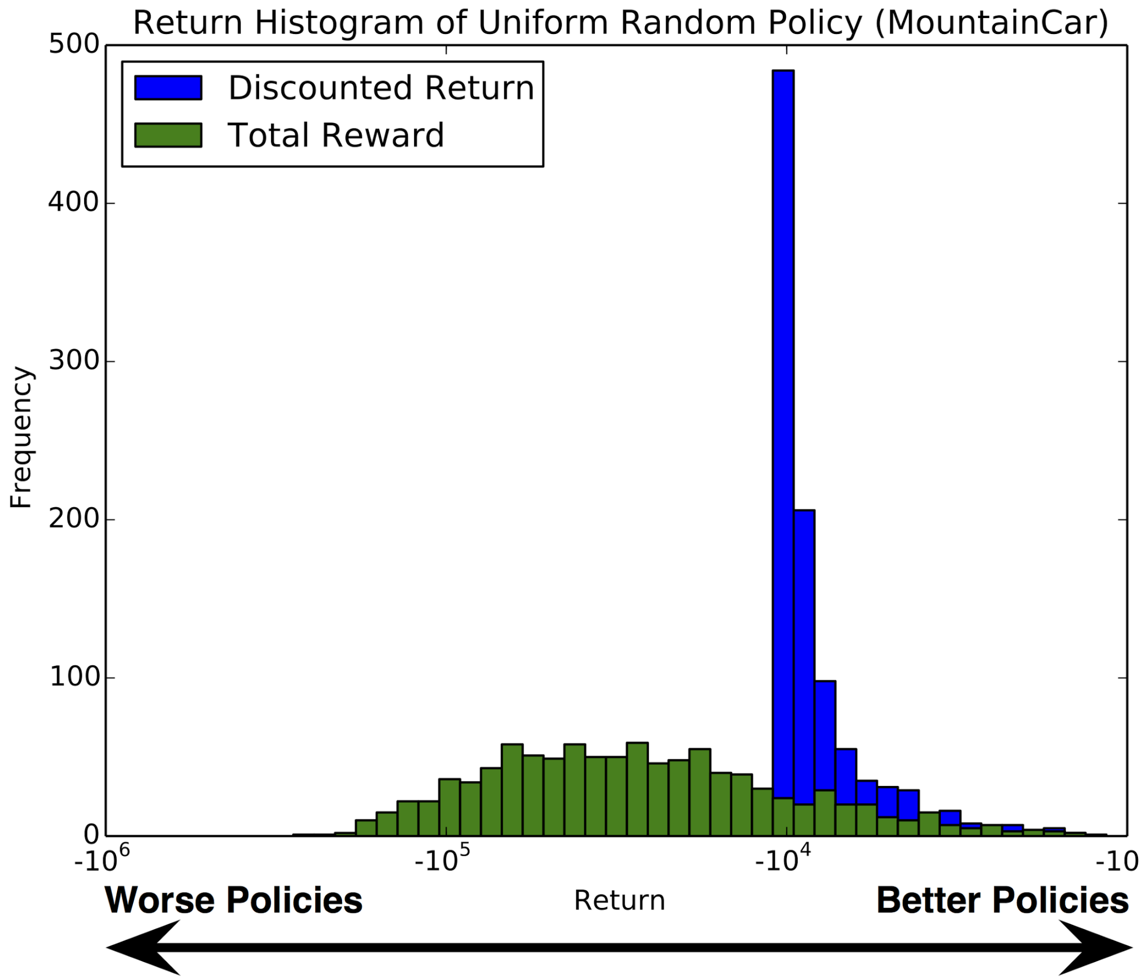


Figure 3.4: Histogram of undiscounted return (Total Reward) compared with discounted return ($\gamma = 0.9999$) on Mountain Car. Optimal policies achieve a total reward of about -120, and an undiscounted return of about -119.

interpret as a measure of a policy's performance, and the differences between policies are not obscured when the actual returns differ by small amounts due to a smaller discount factor. Another benefit is that policy percentile implicitly conveys the same information as when an optimal policy and a baseline policy are shown for comparison purposes. This has the effect of making ceiling effects more apparent and producing a measure of performance relative to the difficulty of the domain. The policy percentile provides a measure whose scale is independent of the domain and that inherently takes into account one of the difficulties of learning the MDP. One way to think about this is that this measure normalizes performance with respect to a baseline algorithm, which is random policy search.

Figure 3.5 gives an example showing the policy percentile of Sarsa(λ) and Q-Learning on Mountain Car ($\gamma = 1.0$). Compare this example with Figure 3.6, which shows the learning curve for discounted return for the same experiment. Standard error, shown with error bars, can be computed easily because the policy percentile is monotonically increasing with discounted return. Notice that for the first episode the difference in discounted returns corresponds to a difference of 0.005 in policy percentile, whereas the much smaller gap in discounted returns at episode 8 results in an even larger difference in policy percentile. This is because it is much harder to find policies that are near optimal than those that are far from optimal. Thus, for policies far from optimal the policy percentile contracts the distance between two learning curves and for near-optimal policies it expands this distance. This is controlled by the empirical cumulative distribution over discounted returns for each domain so that the policy percentile transforms discounted returns based on the difficulty of the domain.

3.3.4 Difficulty and Impact of Parameter Tuning

Parameter tuning is the meta-optimization of the parameter vector of an algorithm with respect to some objective function. Reports of empirical results, such as those shown in the previous section, are really presenting results for the combination of some meta-optimization process and the learning algorithm. The pervasive problem throughout RL research is that the meta-optimization step is rarely acknowledged or discussed in any way. This fundamentally discredits the experimental results because the unspecified meta-optimization step can mask differences in the RL algorithms.

If the meta-optimization method is not specified then it becomes much harder to use an algorithm for new domains or for variations on existing experiments. The

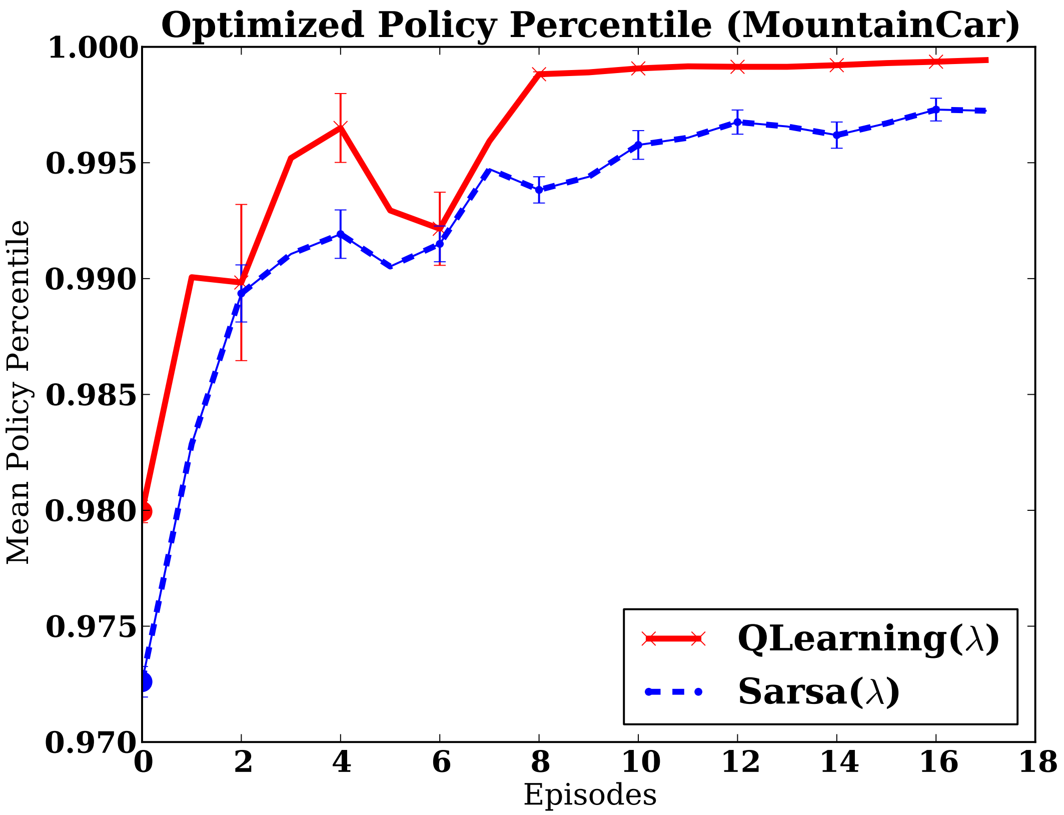


Figure 3.5: Average policy percentile as dependent variable for Sarsa(λ) and Q-Learning on Mountain Car.

method for meta-optimization must be precisely defined. If hand tuning, or any other interactive human-guided method, is used as the meta-optimization method then the results will be biased by the researcher's familiarity with, and understanding of, the algorithms under consideration. The meta-optimization method must be automatic and objective. If only the final result of the meta-optimization process is reported then no information about the difficulty of the process, the impact of parameter tuning, the sensitivity of the algorithm with respect to its parameters, nor the average-case performance is communicated. Finally, if only a small number of parameter vectors are considered then neither the researcher nor the audience is given any information about the expected performance of the algorithm over the parameter

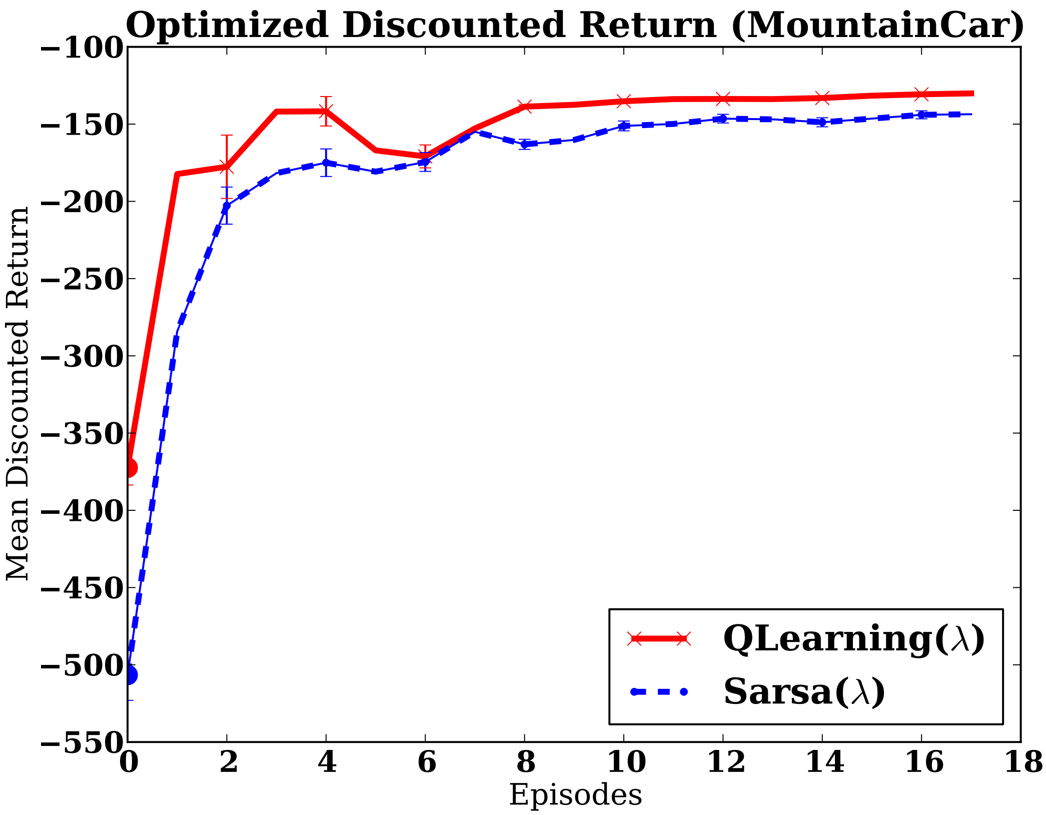


Figure 3.6: Discounted return as dependent variable for Sarsa(λ) and Q-Learning on Mountain Car

space. Current standard practices provide high variance and highly-biased estimates of the performance of algorithms.

We propose the following methods for measuring the difficulty and impact of parameter tuning:

1. Precisely define the learning algorithm, including an explicit definition of the parameter space.

2. Describe the meta-optimization algorithm used and the objective function it attempts to maximize.

3. Run the meta-optimization procedure K times, recording the objective function value for every parameter vector evaluated. Each meta-optimization procedure

evaluates N parameter vectors, thus producing $N \times K$ total parameter evaluations.

4. Report the average and standard error of the objective function over the K runs of the meta-optimization for the best parameter vector found so far against the number of parameter vectors evaluated in the run. That is, for the independent variable *Parameters Evaluated*, $0 \leq n \leq N - 1$, report μ_n and σ_n given by

$$\mu_n = \frac{1}{K} \sum_{k=1}^{K} \max_{0 \leq i \leq n} f_k(i), \tag{3.2}$$

$$\sigma_n = \frac{1}{K} \sqrt{\sum_{k=1}^{K} (\max_{0 \leq i \leq n} f_k(i) - \mu_n)^2}, \tag{3.3}$$

   where $f_k(i)$ is the objective function value for the i^{th} parameter vector evaluated during the k^{th} run of the meta-optimization procedure.

The results from the above procedure can be reported easily in a table, or by a learning curve with error bars. For example, we used a randomized parameter search meta-optimization for Sarsa(λ) and Q-Learning on Mountain Car[3], with $K = 50$ runs of length $N = 120$. This means we ran fifty independent randomized parameter searches for each algorithm on the Mountain Car domain. Each randomized parameter search sampled 120 different parameter vectors and evaluated the algorithm with each using the domain.

In Figure 3.7 we give the results of applying the above procedure. In addition to those already discussed, other useful details can be read from this figure. The y-intercept of each curve, the starting value for average policy percentile, gives an estimate of the average policy percentile of each algorithm if parameter vectors were

[3] Sarsa(λ) and Q-Learning share the same parameter space. We used ϵ-greedy policies and the following parameter space: $\alpha \in (0, 1]$, $\lambda \in [0, 1]$, and $\epsilon \in [0, 1)$. Both algorithms used a Fourier basis order 3 for function approximation. Mountain Car's discount factor was $\gamma = 1.0$.

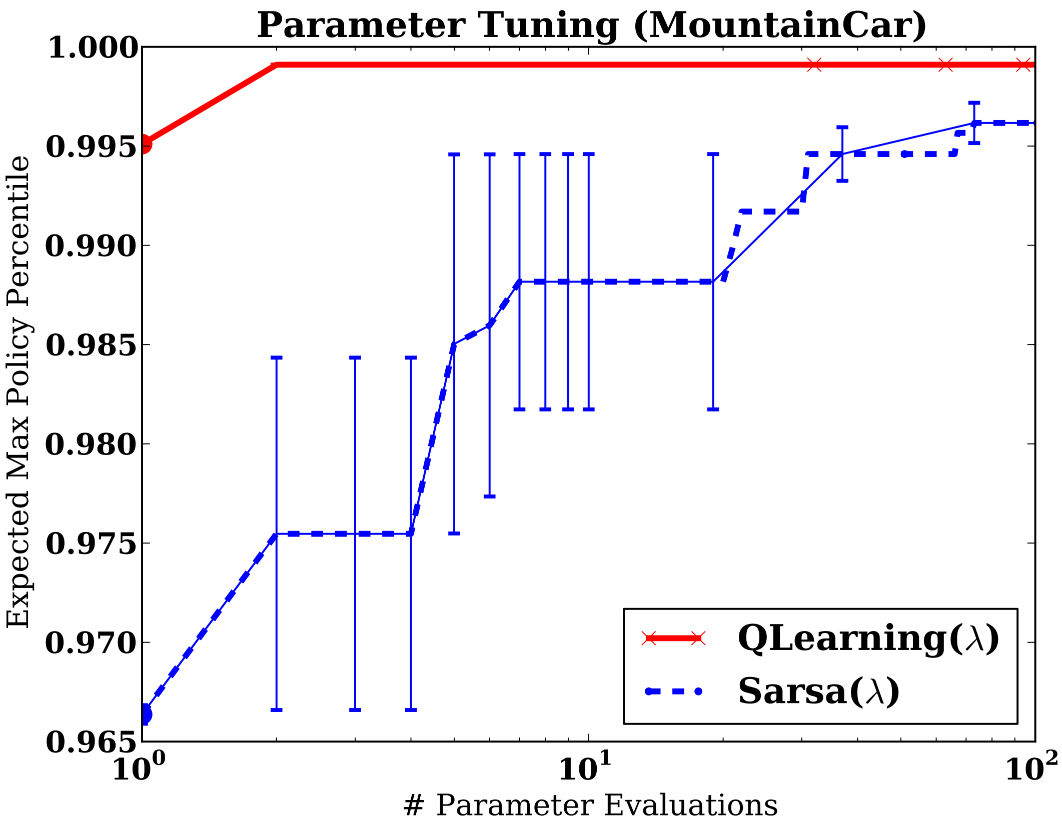


Figure 3.7: Difficulty and impact of parameter tuning for Sarsa(λ) and Q-Learning on Mountain Car. Expected max policy percentile (μ_n) shown by lines and standard error (σ_n) shown by error bars.

chosen randomly. This provides a measure of the average case performance of each algorithm. The difference between the final policy percentile and this initial value is the *impact of parameter tuning*, which shows how much an algorithm benefits from parameter tuning. The *difficulty of parameter tuning* is shown by the rate at which each curve increases toward its maximum. The difficulty and impact of parameter tuning curves provide a more complete picture of the performance characteristics of an algorithm.

The above procedure requires a scalar valued objective function, but running an RL algorithm produces a sequence of discounted returns. If each evaluation runs for E episodes, then the sequence is of length E. Any choice of objective function must transform this sequence of values into a single scalar, and in doing so expresses some trade-off between final policy performance and learning speed. On the one extreme of this spectrum is an objective function that averages over all episodes and on the other extreme is an objective function that returns only the discounted return of the last episode. We propose to break the sequence of discounted returns into two phases: the *learning phase* and the *evaluation phase*. The learning phase is characterized by an approximately increasing discounted return and the evaluation phase by an approximately constant discounted return.

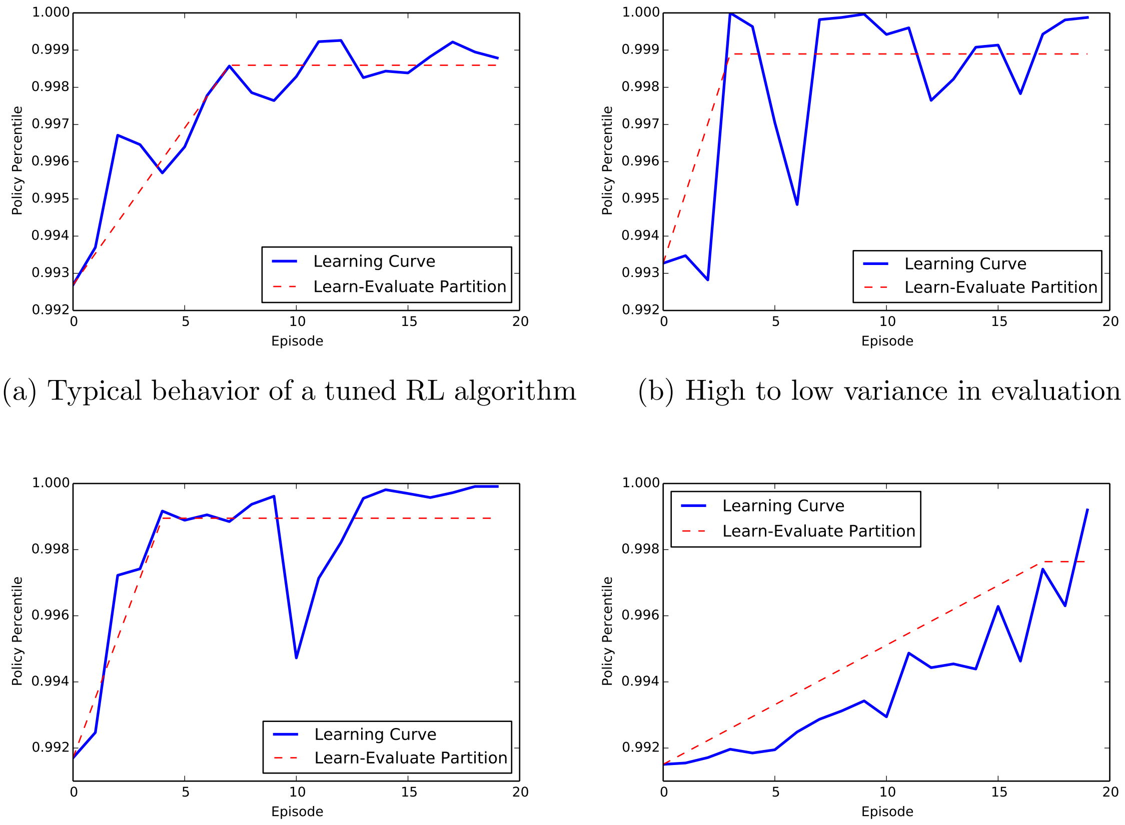
The objective function we propose for meta-optimization of RL algorithms: (1) converts the sequence of discounted returns into a sequence of policy percentiles, (2) automatically partitions the sequence into a learning phase and an evaluation phase, and (3) returns the average policy percentile of the evaluation phase. The assumption that makes this automatic partitioning possible is that the learning phase is best approximated by a line through the first episode's policy percentile and that the evaluation phase is best approximated by its mean. Therefore, finding such a partitioning is equivalent to minimizing the squared error of the two approximations.

Let ρ_i denote the policy percentile for episode i. Then, the learning-evaluation partition is defined by the episode number n^* that produces the lowest sum-squared error:

$$n^* = \operatorname*{arg\,min}_{2 \leq n < E-1} \left[\min_{\beta \in \mathbb{R}} \sum_{i=0}^{n} (\rho_i - \rho_0 - \beta i)^2 + (E-n) Var(\{\rho_i\}_{i=n}^{E}) \right]. \tag{3.4}$$

The error term in red gives the sum-squared error of approximating the learning-phase with a line through the first episode's value and the error term in blue gives the sum-squared error of approximating the evaluation-phase with its mean. Given this partitioning procedure, the objective function we propose is given by:

$$f_k(i) = \frac{1}{E - n^*} \sum_{j=n^*}^{E} \rho_j. \tag{3.5}$$



(a) Typical behavior of a tuned RL algorithm

(b) High to low variance in evaluation

(c) High variance in evaluation, possibly due to numerical instabilities or too large a step size.

(d) Slow learning behavior, likely due to too small a step size or too few episodes for this domain.

Figure 3.8: Learning curves and Learn-Evaluate partition generated by Equation 3.4

This objective function gives the mean policy percentile over the last $(E - n^*)$ episodes, where n^* is selected so that these remaining episodes are better approximated by a constant mean than by inclusion in a linear approximation of the initial learning speed. This formulation captures the essential separation of an RL algorithm's performance into an initial phase dominated by the learning speed (represented by the slope of the line between ρ_0 and ρ_{n^*}), and the behavior of the algorithm once it has leveled out. Figure 3.8 gives some representative examples of the partitioning this method finds between the learning phase and the evaluation phase of a RL algorithm's execution.

3.3.5 Importance of Multiple Domains

Two main reasons motivate studying the behavior of RL agents on an MDP M. The first reason is to solve M itself and the second reason is because M is a representative MDP from a class of interest. In other words, one may study the behavior of an algorithm on M because controlling M is the end goal, or one may study an algorithm's behavior on M as a means to measure some other variable.

Typically it is this second purpose for which RL research papers study behavior on an MDP, and the variable they seek to measure is the performance of an algorithm on a class or a subset of MDPs. When an algorithm is demonstrated on a single problem the only thing one knows is that "it works" (to a certain extent) on that particular problem. There is little information about how effective the algorithm will be if applied to any other MDP. This is why reporting empirical results over multiple domains is important.

Imagine each MDP as a point in an infinite dimensional space. Each MDP on which an algorithm is studied provides a measurement at one point in that space. There is no hope of testing an algorithm on every MDP, nor even on a spanning subset, but the more numerous and varied the set of MDPs used for experiments the more meaningful results will be and the more confident one can be in drawing conclusions.

Already a large number of MDPs have been studied in RL publications, and while designing interesting and challenging new domains should be encouraged, there already are enough to begin to construct sets of benchmark MDPs. Individually these MDPs are not interesting for the purposes of demonstration, but used together they begin to provide a detailed image of the performance characteristics of an RL algorithm. The use of benchmark datasets has become commonplace in the field of supervised learning, making comparisons between different algorithms much more

Domain	State Space	Discount (γ)	Source
50-Chain	Discrete	0.9	[Lagoudakis and Parr, 2001]
BlocksWorld	Discrete	1.0	[Geramifard et al., 2011]
BlackJack	Discrete	1.0	[Sutton and Barto, 1998a]
GridWorld 10x10	Discrete	1.0	[Sutton and Barto, 1998a]
SysAdmin	Discrete	0.95	[Guestrin et al., 2003]
CliffWalk	Discrete	1.0	[Sutton and Barto, 1998a]
Acrobot	Continuous	1.0	[Sutton and Barto, 1998a]
Bicycle (with shaping rewards)	Continuous	0.8	[Lagoudakis and Parr, 2001]
Cart Pole (Balance, 1-pole)	Continuous	0.95	[Lagoudakis and Parr, 2001]
Cart Pole (Balance, 2-poles)	Continuous	0.99	[Wieland, 1991]
Cart Pole (Swing Up)	Continuous	0.95	
HIV Treatment	Continuous	0.98	[Ernst et al., 2006]
Mountain Car	Continuous	1.0	[Sutton and Barto, 1998a]
Planar Swimming	Continuous	0.98	[Tassa et al., 2007]
Puddle World	Continuous	1.0	[Sutton, 1996]

Table 3.3: RL Benchmark: Our proposed set of benchmark MDPs, with (6) discrete state problems and (9) continuous state problems.

straightforward. By contrast, it is rarely possible to directly compare published RL results of different papers because the sets of domains are unlikely to overlap.

Table 3.3 gives our proposed set of benchmark MDP domains. The benchmark domains contain some of the most frequently used MDPs in the RL literature, all have state sets with fairly modest dimensionality, and there is a range of difficulties (from very easy to moderately hard). An important property of all these domains is that they are not computationally intensive to simulate, which allows for more thorough empirical studies. That is not to say that this list is comprehensive and that research would not benefit from additional domains as well as from real world applications. The list is heavily biased toward the types of problems considered in this thesis. Additional benchmark sets should be devised with the aim of measuring performance on the class of continuous action MDPs, skill learning domains, and high-dimensional continuous-state MDPs. Each such benchmark set would provide useful insight into the behavior of the applicable RL algorithms.

3.3.6 Comparing Reinforcement Learning Algorithms

Of course multiple learning curves corresponding to different algorithms can be included in the same figure for visual comparison. However, when the number of domains becomes larger doing this for every domain individually becomes problematic. This can still be done for a few particularly informative domains, but in the case of our proposed set of benchmark MDPs a better approach is to use a single visualization by giving learning curves for each algorithm formed by averaging over all domains in the benchmark set. This may be supplemented by paired hypothesis testing to compare algorithms over multiple domains.

We use Cohen's (1995) randomized paired sample hypothesis test, given by Algorithm 5. The test is applied to two samples, each sample corresponding to a treatment of one of the two algorithms. For our purposes each treatment sample is a scalar evaluation of an algorithm on a particular MDP. We use three such evaluations in this dissertation. The first two apply to the learning curves with optimized parameters and are the learning speed and evaluation policy percentile given by the learn-evaluate partitioning of a learning curve. The third is applied to the parameter tuning curves and is a quadratically weighted average of the means computed in Equation 3.2:

$$\frac{\sum_{i=0}^{N}(N-i)^2\mu_i}{\sum_{i=0}^{N}(N-i)^2}. \tag{3.6}$$

This expresses our stated bias in favor of minimal parameter tuning and strong average case performance.

We use two different statistics with the hypothesis test and these three evaluations. The first, $f_\mu(A,B) = \sum_{i=1}^{N} \frac{a_i - b_i}{N}$ where $a_i \in A, b_i \in B$, corresponds to a randomized version of the paired sample t test. The second, $f_\sigma(A,B) = \frac{var(A)}{var(B)}$ where $var(A)$ and

$var(B)$ are the variances of samples A and B, tests if the two algorithms have equal variance in performance over multiple MDPs. More precisely, the null hypothesis for f_μ is that there is no difference in the mean performance between algorithms A and B over the set of MDPs. The null hypothesis for f_σ is that the algorithms are equally variable. Notice that for f_σ one needs to subtract the mean over samples for A and B before applying the hypothesis test so that each set of paired samples are mean zero. Additionally, we are testing if the variance of A is greater than the variance of B, and it is for this reason that $var(A)$ appears in the numerator. If the test is to be reversed the numerator and denominator must also be switched [Cohen, 1995].

Algorithm 5 Randomized Paired Samples Hypothesis Test

```
Let $f(A, B)$ be a statistic on paired samples $A$ and $B$,
and let $F_{\theta^*}$ be the empirical cumulative distribution function over $\theta^*$.

input: Paired samples $A$ and $B$ of size $N$.

$\theta = f(A, B)$                    ▷ Compute statistic for original paired sets of samples
for $k = 1, \ldots, K$ do
    for $i = 1, \ldots, N$ do
        if $random() \leq 0.5$ then                ▷ Randomly swap set membership
            $A_i^*, B_i^* = B_i, A_i$
        else
            $A_i^*, B_i^* = A_i, B_i$
        end if
    end for
    $\theta^*[k] = f(A^*, B^*)$            ▷ Compute statistic for randomized paired samples
end for
return $F_{\theta^*}(\theta)$
```

A final detail remains before these methods can be put into practice. The above hypothesis tests are based on samples of the algorithms' performances on a set of MDPs. An obvious choice for the set of MDPs is a benchmark set such as those previously discussed (Table 3.3). As for the sample of performance we could use discounted return, but in addition to the reasons already discussed there is another reason that policy percentile is particularly well suited for use in this case. Unlike discounted return, policy percentiles are on a uniform scale across all domains. This is particularly useful here because the range of returns among MDPs can vary substantially, making comparisons such as those discussed in this section less meaningful because we would be comparing samples that come from completely different ranges.

3.4 Experiments with Proposed Methods

3.4.1 Use In Practice (In the Presence of Page Limits and Deadlines)

There are two limitations with these proposed procedures. First, the parameter meta-optimization procedure is inherently time consuming. Second, the procedure produces enough experimental results to fill many pages with just learning curve figures. However, the computational burden can be somewhat reduced, and because of the use of policy percentiles so too can the number of figures. The following suggestions may allow the proposed meta-optimization procedure to be run despite constraints on resources:

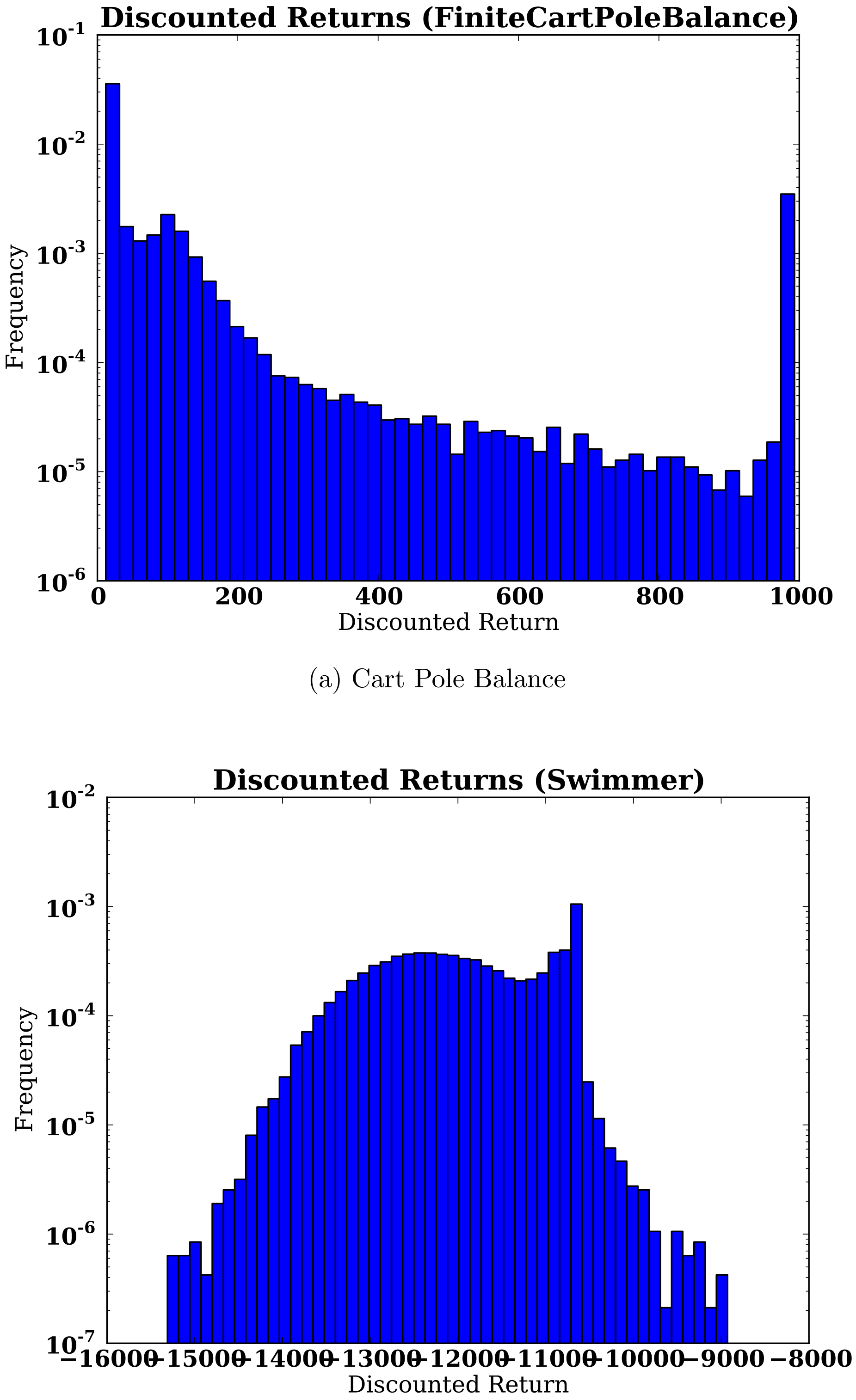
- Cut off all trials once the failure threshold is reached.

- The empirical cumulative distribution of discounted return need only be computed once per domain.

- Define the parameter space wisely, and bias sampling appropriately (e.g. step sizes sampled from a log-normal distribution with negative mean).

- Keep the number of parameters evaluated per meta-optimization, N, within reason. If one algorithm is particularly sensitive to parameter values consider running only that algorithm with large values of N.

- The learn-evaluate partition objective function is robust to different episode increments and can be used when the discounted return is stored for only a spanning subset of the episodes. If disk space is a limiting factor this can be used to overcome the limitation.



(a) Cart Pole Balance

(b) Swimmer

Figure 3.9: Empirical discounted return distributions for randomly sampled fixed-policies.

Additionally, the empirical distribution of discounted returns is usually well behaved outside of the failure threshold (see Figure 3.9), but is rarely normal. This means that when sampling policy performance is expensive an estimated model of the distribution may provide a suitable solution.

One important aspect not covered is the need for modular, reusable, software libraries of RL domains and algorithms. To this end, we use the domain implementations from the RLPy project [Geramifard et al., 2013].

The RL Benchmark set contains 15 MDPs and so produces twice as many figures when used with the proposed procedure (one for learning curves and one for parameter-tuning performance). For each domain we may give the learning curve showing the average policy percentile with optimized parameters as well as the parameter tuning figure that shows the difficulty and impact of the meta-optimization procedure. However, as previously mentioned, the use of policy percentile provides a measure whose scale is independent of the domain. To reduce the number of figures one can report the average over the entire RL Benchmark set instead of for each domain. This can be done for both the learning curve and parameter tuning figures, and shows how well an algorithm performs averaged over the entire set of domains. Finally, individual domains may be featured to discuss interesting cases in which an algorithm does particularly well or poorly.

3.4.2 Case Study: Sarsa(λ) With and Without an Adaptive Step Size

In this case study we compare Sarsa(λ) with and without an adaptive step size. The adaptive step-size algorithm used is not particularly important for our purposes, but we use the PARL2 algorithm introduced in Chapter 4. For this study we used $K = 4$ independent randomized searches of length $N = 100$ over the parameter values for both algorithms. The two algorithms have the same parameter spaces except that PARL2 does not have a tunable step size, and thus has one fewer parameter.

Figure 3.10a shows the learning curves of the two algorithms using optimized parameters, and is an average over 30 runs. The error bars show that the differences are not statistically significant, but the adaptive step-size algorithm appears to learn slightly faster. Both figures are averages over the RL Benchmark set. Figure 3.10b shows the results of the meta-optimization procedure. The adaptive step-size algorithm has fewer parameters to tune and gives better performance on average than Sarsa(λ) when parameter values are chosen randomly. However, after parameter tuning both algorithms reach similar objective function values.

Although the average performance of the two algorithms is very similar they each perform differently for some individual domains. On the BlackJack domain (Figure 3.11a) Sarsa(λ) out performs the adaptive step-size method both during parameter tuning and with optimized parameters. The discounted returns for BlackJack are high variance, which might explain why this particular adaptive algorithm does poorly in this case. However, in the HIV Treatment domain (Figure 3.11b) the adaptive step-size algorithm is far superior to Sarsa(λ).

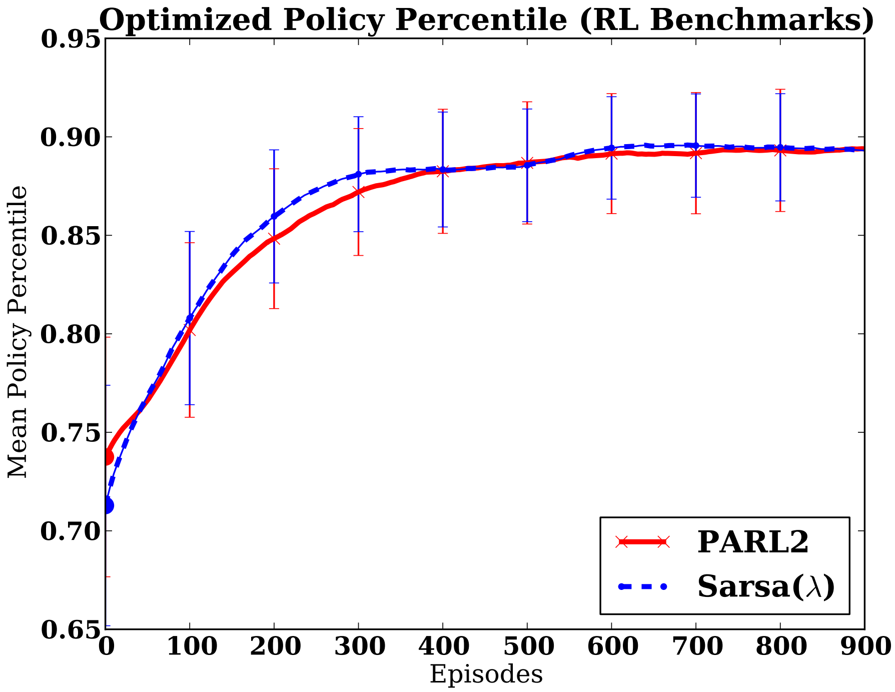
3.4.3 Case Study: LSPI, NAC, and Sarsa(λ)

We now turn to a comparison of three RL algorithms: Sarsa(λ), Natural Actor-Critic (NAC), and Least-Squares Policy Iteration (LSPI). We introduced all three of these algorithms in Chapter 2, but did not discuss their parameters in depth. NAC, as we have implemented it, takes six parameters, three of which control the frequency of updates and at what rate past information is discarded. LSPI has the advantage of not requiring a step-size parameter, but takes five parameters. Sarsa(λ) has only three parameters. Thus, NAC and LSPI have much larger parameter spaces than Sarsa(λ). Figure 3.12a shows the parameter tuning curve for these three algorithms and clearly illustrates the consequences of such a large parameter space. If we ran the meta-optimization procedure for much larger number of evaluations, such as $N = 10000$, then we expect that both LSPI and NAC would come to out-perform Sarsa(λ) on most domains.

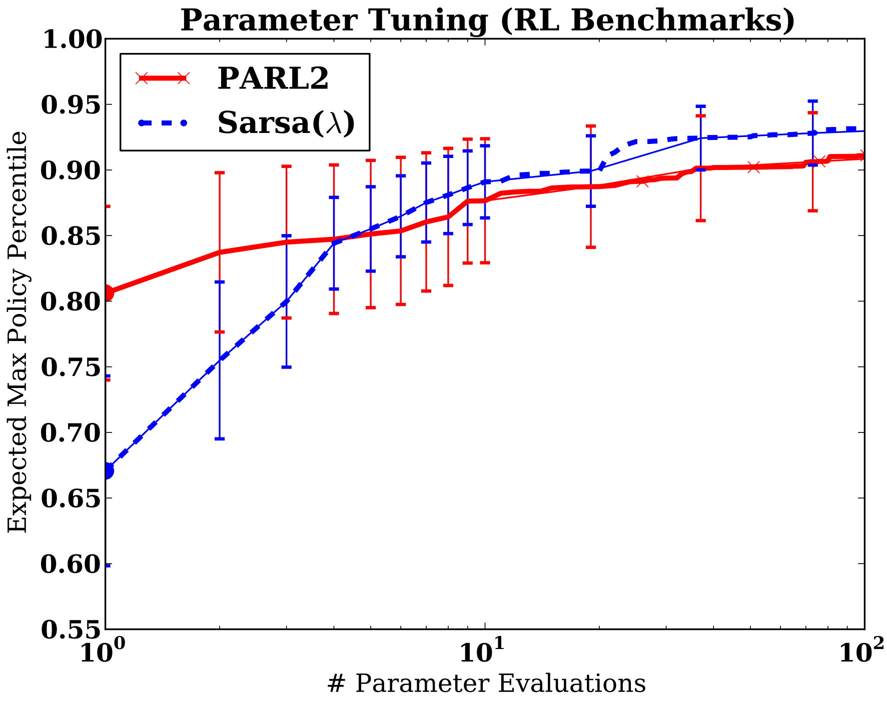
However, as Figure 3.12b shows, given only modest amounts of parameter tuning Sarsa(λ) significantly out-performs both algorithms. Specifically, we can say that over the RL Benchmark set Sarsa(λ)'s mean policy percentile after parameter tuning is greater than that of NAC's ($p > 0.005$) and LSPI's ($p > 0.01$). LSPI is known to perform particularly well on the Bicycle Riding task [Lagoudakis and Parr, 2003], but is also known to be somewhat sensitive to the parameter values. Although we used the same discount factor as Lagoudakis and Parr [2003] none of the parameter values found during meta-optimization performed nearly as well as previously reported results. If instead we hand tune parameter values for NAC or LSPI we observe the same level of performance reported by others. We must conclude that these two algorithms are much more sensitive to parameter values than Sarsa(λ) and are only competitive when extensive parameter tuning is performed or the practitioner has expert knowledge of the algorithms in question.

3.5 Conclusion

The empirical methods regularly used in RL impede research progress by masking the performance of the RL algorithms with the unreported parameter tuning process used to achieve the reported results. We have shown, both in general and with concrete examples, the negative effects of current practices and how they might lead to erroneous conclusions. We further motivated the work with the results of a survey of empirical methods used in RL conference papers during the preceding year which support our claim that current practices are insufficient. We then proposed a set of experimental methods for performing studies on RL algorithms as well as reporting the results of those studies. Finally, we demonstrated their use in two case studies comparing some well known RL algorithms.

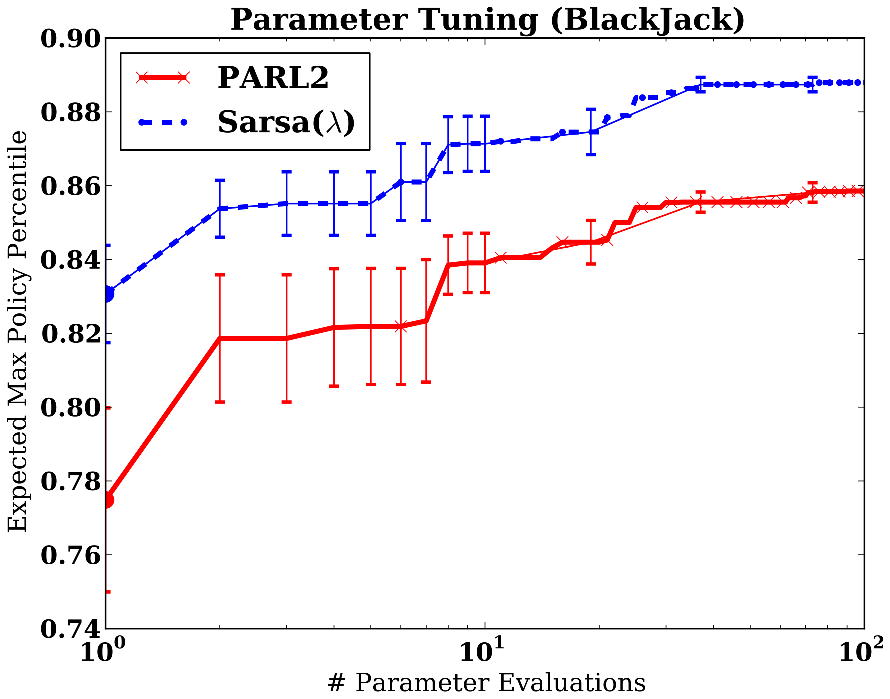


(a) Performance with optimized parameters averaged over RL Benchmark set of domains.

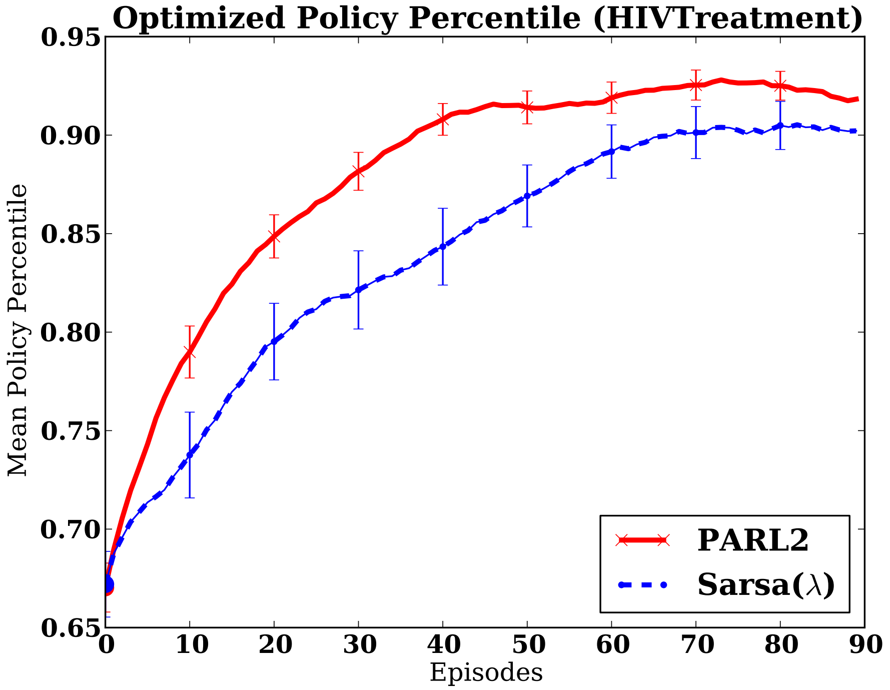


(b) Difficulty and Impact of parameter tuning.

Figure 3.10: Sarsa(λ) with and without an adaptive step size.

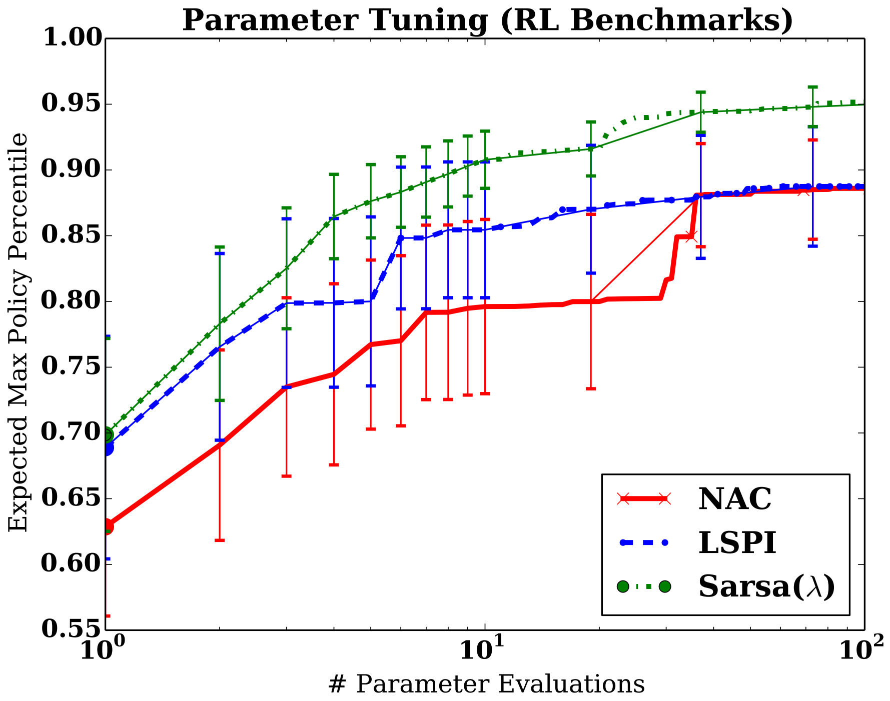


(a) Parameter tuning curves on the BlackJack domain

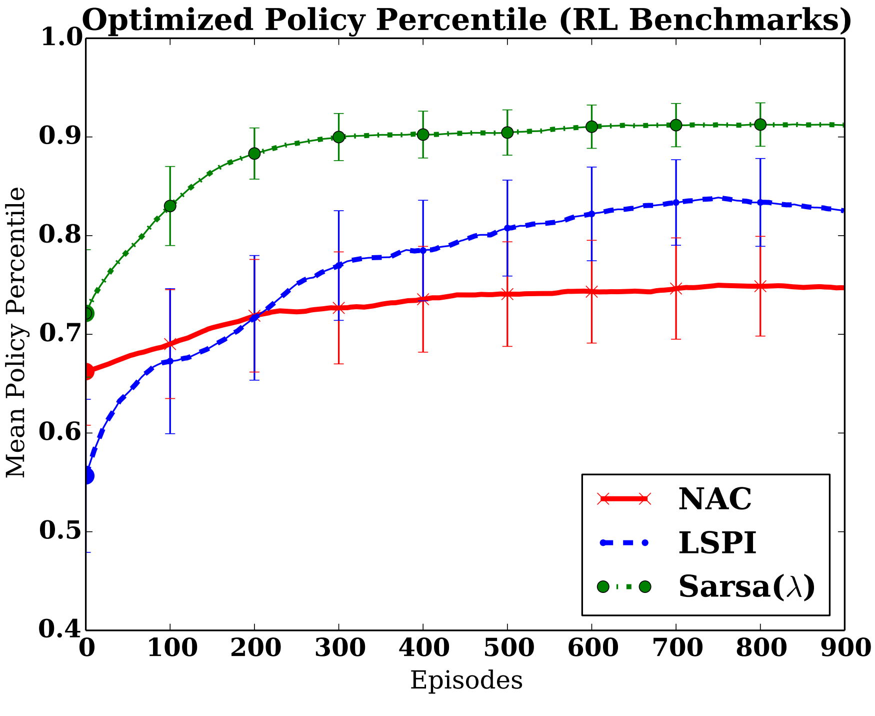


(b) Learning curves on the HIV Treatment domain.

Figure 3.11: Sarsa(λ) with and without an adaptive step size on individual domains.



(a) Difficulty and Impact of parameter tuning.



(b) Performance with optimized parameters averaged over RL Benchmark set of domains.

Figure 3.12: Case study of Sarsa(λ), LSPI, and NAC averaged over RL Benchmark.

CHAPTER 4

ADAPTIVE SCALAR STEP SIZES FOR REINFORCEMENT LEARNING

This chapter focuses on the adaptive scalar step-size problem for RL. Our primary contributions the derivations of three parameter-free adaptive scalar step-size algorithms for RL. Additionally, we perform an empirical study of deterministic step-size schedules and adaptive step-size algorithms in RL that is far more comprehensive than any other in the field.

4.1 Introduction

Online reinforcement learning (RL) algorithms, such as Sarsa(λ), maintain an approximate action-value function and at each time step update the approximation toward some locally optimal solution. How far the approximation weights are moved in this direction is determined by the step size. In this way, the step size directly controls how quickly or slowly an algorithm incorporates new information.

Most RL algorithms are highly sensitive to the choice of step-size value because the RL problem is inherently non-stationary—as policies improve the distributions over states and rewards change—and because effective values can vary dramatically across domains and function approximation methods.

Let $\mathcal{J} : \mathbb{R}^n \rightarrow \mathbb{R}$ be a loss function over weight vectors $w \in \mathbb{R}^n$, and A be an incremental algorithm. Given a sequence of step sizes $\{\alpha_t\}_{t=0}^{\infty}$, A produces update directions $\triangle w_t \in \mathbb{R}^n$ intended to minimize $\mathcal{J}(w_t)$ and produce the sequence of parameter vectors defined by:

$$w_{t+1} = w_t - \alpha_t \triangle w_t.$$

The adaptive step-size problem is to generate a sequence of step sizes $\{\alpha_t\}_{t=0}^{\infty}$, where $\alpha_t > 0 \ \forall t$, which minimize the loss incurred by the learning algorithm. The adaptive step-size problem for RL restricts the problem to the set of RL algorithms and their associated loss functions. Ideally RL algorithms would minimize a loss function representing the policy's distance to a locally optimal policy. Policy gradient algorithms can be viewed as minimizing such a loss function, but these methods can only be applied to differentiable policies. Another approach is to minimize a loss function that indirectly leads to a locally optimal policy. For example, a commonly used loss function in RL is the mean-squared Bellman error (MSBE):

$$\mathcal{J}_{MSBE}(w_t) = \frac{1}{2}\mathbb{E}_{s \sim d^{\pi}, a \sim \pi}\left[r + \gamma Q_{w_t}(s', a') - Q_{w_t}(s, a)\right]^2.$$

For finite MDPs the Bellman optimality principle says that if a policy achieves zero MSBE then the policy is optimal [Sutton and Barto, 1998a]. Thus, the MSBE loss function can be used to indirectly optimize the policy with respect to an approximate value-function.

A number of choices can be made within this framework that result in different adaptive step sizes, but each choice comes with a particular set of assumptions placed on the sequence of step sizes, the loss function, and the algorithm. For example, if the assumptions are that the optimal sequence of step sizes is constant over time, $\alpha_t \equiv \alpha_0^*$ $\forall t \in \mathbb{N}$, that the loss function is MSBE, and if the agent's algorithm uses linear function approximation, then stochastic gradient descent may be used to generate a sequence of step sizes which converges to α_0^*. In this chapter we explore some of the possible avenues for deriving an adaptive scalar step size for action-value based RL, each motivated by some set of assumptions about the problem.

Chapter 2 reviewed a variety of the adaptive step-size methods available for stochastic gradient descent, and noted the fact that few methods have been explicitly designed for use in RL. Although the methods in this chapter may be extended to other algorithms, it is assumed that the RL algorithm is Sarsa(λ) and that the step sizes are positive scalars. Sarsa(λ) is used because it is a simple algorithm that often performs competitively with state-of-the-art methods after suitable parameter tuning.

We consider three fundamentally different approaches for the derivation of an adaptive step size. The first, which is most closely related to Incremental Delta-Bar-Delta (IDBD) [Sutton, 1992b], assumes that there is some unknown, fixed, locally optimal step size and uses stochastic gradient descent (SGD) to improve the current step size incrementally in that direction. The second, which is a generalization of the vSGD adaptive step-size algorithm [Schaul et al., 2012], solves the optimal step-size problem with respect to estimated expected values that take into account the variance of the Sarsa(λ) updates. The final approach was inspired by the passive aggressive algorithm for online learning [Crammer et al., 2006]. Our approach starts by finding the step size that aggressively minimizes the squared Bellman error of the current transition and uses this to create a passive aggressive adaptive step size upper bounded by this aggressive step size.

While this chapter focuses on scalar step sizes, some methods can be extended to the case of vector-valued step sizes. We argue in favor of the separability of the adaptive scalar step size from the correction of the update direction. We then derive adaptive scalar step-size algorithms for RL and provide an empirical study of the three adaptive step-size methods compared with each other and with a variety of existing step-size schedules and adaptive algorithms.

4.2 Update Whitening and Adaptive Scalar Step Sizes

The step size is classically assumed to be a positive-real-scalar value, but recent research has also focused on vector-valued step sizes [Duchi et al., 2011b, Schaul et al., 2012, Ross et al., 2013]. Vector-valued step sizes do more than simply scale the magnitude of an update, they also change the direction of the update in the same way as positive definite matrix-valued step sizes. Although these developments in adaptive vector-valued step sizes are recent, the concept of matrix-valued step sizes is far from being a recent development and is central to work on quasi-Newton methods in convex optimization. However, adaptive vector-valued step-size methods attempt to modify the update direction without the need for an additional scalar step size.

We claim that these two problems, despite often being treated simultaneously, can be solved sequentially very effectively. The two problems are the *adaptive scalar step-size problem* and the *update whitening problem*, which attempts to correct for differences of scale, variability and higher order effects of the loss function. The main result of this section derives the optimal step size for any given whitening matrix. We now proceed to define the update whitening problem.

Definition 4.1. *For matrix $G \in \mathbb{R}^{n\times n}$, $G \succ 0$ indicates that G is a positive definite matrix.*

Assumption 4.1. *The loss function $\mathcal{J} : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable with respect to parameters $w \in \mathbb{R}^n$.*

Definition 4.2. *Let ξ be either a deterministic approximation error vector or a mean-zero noise vector with bounded variance, then for loss function $\mathcal{J}$ satisfying Assumption 4.1 a **descent direction** at w is given by:*

$$\triangle w = -\nabla \mathcal{J}(w_)) + \xi.$$

Definition 4.3. *Let $A \in \mathbb{R}^{n\times n}$ be a square matrix. Then the matrix norm $\|A\|$ is assumed to be the operator norm:*

$$\|A\| = \sup\{\frac{\|Ax\|}{\|x\|} \mid x \in \mathbb{R}^n\}. \tag{4.1}$$

If A is invertible then the ***condition number*** *of A is given by:*

$$\kappa(A) = \|A\|\|A^{-1}\|. \tag{4.2}$$

Definition 4.4. *Let $\mathcal{J} : \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function over parameters $w \in \mathbb{R}^n$. Taylor's theorem gives the following equivalence [Sun and Yuan, 2006]:*

$$\begin{aligned} \mathcal{J}(w + \triangle w) &= \mathcal{J}(w) + \nabla\mathcal{J}(w)^\top \triangle w + R_2(w + \triangle w), \ \textit{where} & (4.3)\\ R_2(w + \triangle w) &= \triangle w^\top D^2\mathcal{J}(w + \triangle w)\triangle w, \ \textit{with} & (4.4)\\ D^2\mathcal{J}(w + \triangle w) &= \frac{1}{2}\int_0^1 (1-t)\nabla^2\mathcal{J}(w + t\triangle w)dt. & (4.5) \end{aligned}$$

Definition 4.5. *Given a loss function satisfying Assumption 4.1 and descent direction $\triangle w$, the* ***update whitening problem****, for $\epsilon > 0$, is to find $G^{-1} \in \mathbb{R}^{n\times n}$, with $G^{-1} \succ 0$ and $\|G^{-1}\| \leq 1$, solving:*

$$\begin{aligned} &\min_{G^{-1}} \mathcal{J}(w + G^{-1}\triangle w),\\ &\textit{such that} \quad R_2(w + G^{-1}\triangle w) = \epsilon. \end{aligned}$$

We write the whitened update of $\triangle w$ as $\hat{\triangle} w = G^{-1}\triangle w$.

The update whitening problem is motivated by considering the derivation for stochastic gradient descent, which minimizes the Taylor expansion of $\mathcal{J}(w + \triangle w)$

around w. The error between $\mathcal{J}(w + \triangle w)$ and this approximation is given by Equation 4.4. When $\mathcal{J}$ is continuously differentiable, Taylor's Theorem gives an exact equation for R_2 in terms of the average curvature of the function, but upper bounds on the approximation error can be achieved under milder assumptions.

The idea behind the update whitening problem is to produce an update direction, $\hat{\triangle} w = G^{-1} \triangle w$, that minimizes the loss function while measuring the size of the update in terms of the approximation error from Equation 4.4. This has the result of producing updates that are in a sense more *efficient*, in that they minimize the loss as quickly as possible relative to the local uncertainty in the gradient direction, where this uncertainty may be due to a deterministic approximation error or additive mean-zero noise.

Doing so requires making smoothness assumptions on the loss function. The simplest case is observed when $\mathcal{J}$ is quadratic and noise-free. Then, the Hessian of $\mathcal{J}$ is constant and Newton's Method gives an exact solution to the update whitening problem. This is the ideal case of reducing the condition number to unity, but in general an exact solution will not be available, and some error due to higher order effects and variance in the gradient estimates will remain. In this case, the full update $w + \hat{\triangle} w$ is not guaranteed to minimize the loss function, and a scalar step size α must be introduced to keep $w + \alpha \hat{\triangle} w$ within a neighborhood containing small approximation errors.

Many existing methods can be seen as solving the update whitening problem under varying assumptions. Whitening data refers to the common practice used to produce uncorrelated data with variance one. Preconditioning of optimization problems [Boyd and Vandenberghe, 2004], and whitening the data before applying a machine learning algorithm, are a commonly used techniques for improving the performance of existing methods. These can be seen as the analogous problem in the batch learning setting. Slow feature analysis is closely related, but adds the additional requirement of producing zero mean updates [Wiskott and Sejnowski, 2002]. AdaGrad is an adaptive vector-valued step-size algorithm which approximately solves the update whitening problem by assuming that the update directions are already uncorrelated [Duchi et al., 2011a].

Yang and Laaksonen [2008] explored the connections between the whitened update and the natural gradient. In doing so they also proved that the whitened update maximizes the local change in information. Natural gradients, introduced by Amari [1998], are another approximate solution to the update whitening problem which avoid the use of matrix square roots used by Yang and Laaksonen [2008].

Le Roux et al. [2007] have shown that the natural gradient is the update direction which minimizes the probability that the generalization error will increase. Aside from the theoretical guarantees of natural gradients, they have also been shown to substantially improve performance of online stochastic gradient descent algorithms.

A full discussion of the benefits and procedures for the update whitening problem is beyond the scope of this chapter. However, we now establish some particularly useful aspects of the problem that will aid in the proof of Theorem 4.1. Using the above motivation for the use of a scalar step size, Theorem 4.1 gives conditions on the scalar step size required for convergence when update whitening is used to change the descent direction.

Assumption 4.2. *The matrix $G_t^{-1} \in \mathbb{R}^{n \times n}$ gives an approximate solution to the update whitening problem at time t such that:*

(a) G_t^{-1} exists and is the inverse of the matrix G_t (invertible)

(b) $G_t^{-1} \succ 0$ (positive definite)

(c) $\|G_t^{-1}\| \leq 1$ (contraction)

(d) $\forall t : \kappa(G_t^{-1}) \leq \frac{1}{c_1}$ for some positive constant $c_1 > 0$ (bounded condition number)

Theorem 4.1. *Let $\mathcal{J}$ be a loss function satisfying Assumption 4.1 with parameter vector $w_t \in \mathbb{R}^n$, descent direction $\triangle w_t$, and $\hat{\triangle} w_t = G_t^{-1} \triangle w_t$ for matrix G_t^{-1} satisfying Assumption 4.2.*

Then the optimal scalar step size is given by:

$$\alpha_t^* = -\frac{\nabla \mathcal{J}(w_t)^\top \hat{\triangle} w_t}{R_2(w_t + \hat{\triangle} w_t)}, \quad \forall t \in \{t : t \in \mathbb{N}, \|\nabla \mathcal{J}(w_t)\| > 0\}. \tag{4.6}$$

Furthermore, the sequence $\{w_t\}_{t=0}^{\infty}$ converges with probability one to a local optimum w^* when one exists and if the step-size sequence satisfies:

$$\alpha_t < 2\alpha_t^*, \quad \forall t \in \{t : t \in \mathbb{N}, \|\nabla \mathcal{J}(w_t)\| > 0\}. \tag{4.7}$$

Proof. Begin with the Taylor expansion of $\mathcal{J}(w_t + \hat{\triangle} w_t)$ around w_t and step size α_t minimizing $\mathcal{J}(w_t + \hat{\triangle} w_t)$,

$$\begin{aligned}
\mathcal{J}(w_t + \alpha_t \hat{\triangle} w_t) &= \mathcal{J}(w) + \alpha_t \nabla \mathcal{J}(w)^\top \hat{\triangle} w + \frac{1}{2}\alpha_t^2 R_2(w + \hat{\triangle} w), \\
\mathcal{J}(w_t + \alpha_t \hat{\triangle} w_t) - \mathcal{J}(w) &= \alpha_t \nabla \mathcal{J}(w)^\top \hat{\triangle} w + \frac{1}{2}\alpha_t^2 R_2(w + \hat{\triangle} w), \qquad (4.8) \\
\alpha_t \nabla \mathcal{J}(w)^\top \hat{\triangle} w + \frac{1}{2}\alpha_t^2 R_2(w + \hat{\triangle} w) &< 0, \\
\frac{1}{2}\alpha_t R_2(w + \hat{\triangle} w) &< -\nabla \mathcal{J}(w)^\top \hat{\triangle} w_t, \\
\alpha_t &< -2\frac{\nabla \mathcal{J}(w_t)^\top \hat{\triangle} w_t}{R_2(w + \hat{\triangle} w)}.
\end{aligned}$$

This proves the upper bound. Then, taking the derivative of Equation 4.8 with respect to α_t and solving for α_t, gives the result for the optimal scalar step size. What remains is to prove that satisfying the above bound guarantees convergence.

Bertsekas and Tsitsiklis [2000] established convergence of stochastic gradient descent with modified update directions under general conditions. Specifically, Bertsekas and Tsitsiklis's (2000) proof establishes in our case that the sequence of parameter vectors will converge with probability one if the following hold:

(a) $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$.

(b) For positive scalars c_1 and c_2:

$$c_1 \|\nabla \mathcal{J}(w_t)\|^2 \leq -\nabla \mathcal{J}(w_t)^\top G_t^{-1} \nabla \mathcal{J}(w_t), \tag{4.9}$$

$$\|G_t^{-1} \nabla \mathcal{J}(w_t)\| \leq c_2 \|\nabla \mathcal{J}(w_t)\|. \tag{4.10}$$

(c) ξ_t is either a deterministic error satisfying for positive scalars p and q,

$$\|G_t^{-1} \xi_t\| \leq \alpha_t (q + p \|\nabla \mathcal{J}(w_t)\|), \ \forall t, \tag{4.11}$$

or is a stochastic error satisfying for positive scalar A,

$$\mathbb{E}[\xi_t] = 0, \quad \mathbb{E}[\|\xi_t\|^2] \leq A(1 + \|\nabla \mathcal{J}(w_t)\|^2). \tag{4.12}$$

Condition (a) is the standard assumption on step sizes, required for convergence in stochastic approximation algorithms. Theorem 2.2.4 of Sun and Yuan [2006] proves this condition for our form of step size on quadratic functions. In our case the quadratic term is $R_2(w + \hat{\triangle} w)$ so that their proof is applicable and condition (a) holds. Condition (b) is implied by our assumption that G_t^{-1} is a contraction operator ($\|G_t^{-1}\| \leq 1$) and that the condition number of G_t^{-1} is bounded from above:

$$\begin{aligned}
1 &\geq \|G_t^{-1}\|, \\
&= \sup\{\frac{\|G_t^{-1} x\|}{\|x\|} \mid x \in \mathbb{R}^n\}, \\
&\geq \frac{\|G_t^{-1} \nabla \mathcal{J}(w_t)\|}{\|\nabla \mathcal{J}(w_t)\|}, \\
\implies \nabla \mathcal{J}(w_t) &\geq \|G_t^{-1} \nabla \mathcal{J}(w_t)\|, \\
\therefore c_2 &= 1.
\end{aligned}$$

By the Min-Max theorem, where λ_{min} and λ_{max} are the minimum and maximum eigenvalues of G_t^{-1} respectively,

$$\lambda_{min} \leq \frac{x^\top G_t^{-1} x}{\|x\|^2} \leq \lambda_{max}, \ \forall x \in \mathbb{R}^n \setminus \{0\}. \tag{4.13}$$

Therefore, let $c_1 = \lambda_{min}$ which is bounded away from zero by our assumption that G_t^{-1} is a contraction operator with bounded condition number.

Finally, it is easy to see that condition (c) will hold for the whitened update $\hat{\triangle} w_t$ as long as the condition holds for the original update $\triangle w_t$ with error ξ_t. That is, if ξ_t conforms to the assumptions required for convergence in the usual case of stochastic gradient descent, then ξ_t also satisfies condition (c) in the case of whitened updates. If ξ_t is a deterministic error then this holds trivially because G_t^{-1} is a contraction operator and therefore $\|G_t^{-1}\xi_t\| \leq \|\xi_t\|$. The same property can also be used for stochastic errors: $\mathbb{E}[\|G_t^{-1}\xi_t\|^2] \leq \mathbb{E}[\|\xi_t\|^2]$. Thus, the conditions for gradient convergence are satisfied, completing our proof. □

Theorem 4.1 has previously been proven under more restrictive assumptions, but a better understanding of the role of scalar step sizes is obtained by considering the general case. Without additional assumptions placed upon the loss function, $R_2(w + \triangle w)$ is the average rate at which the gradient may change between the points w and $\triangle w$. Take, for example, the Cauchy step size:

$$\alpha_t = \frac{\|\nabla \mathcal{J}(w_t)\|^2}{\nabla \mathcal{J}(w_t)^\top H(w_t) \nabla \mathcal{J}(w_t)}.$$

The Cauchy step size is, in some sense, optimal. Specifically, it is locally optimal if there is no noise and the Hessian at w_t is assumed to be a good approximation to the average Hessian between w_t and $w_t - \nabla \mathcal{J}(w_t)$. However, although this step size is always below the upper bound required to ensure convergence it is not always exact causing the Cauchy step size to frequently over-shoot the true optimal step size. This is due to the error in the approximation formed by the assumptions on the loss function and the Taylor expansion.

4.3 Adaptive Scalar Step Sizes for Sarsa(λ)

Recall from Chapter 2, the algorithm for Sarsa(λ) with function approximation Q_w and parameterized policy π. Modifying this algorithm to allow for a generic adaptive step size, with initialization (*InitStepSize*) and implementation (*StepSize*) functions gives Algorithm 6. All of the adaptive step-size algorithms in this chapter are presented with pseudo-code implementations of the two functions referenced by Algorithm 6. This pseudo-code does not show the details of how internal variables are maintained for each algorithm, but the reader may interpret local variable assignments within a function as persistent variables internal to a particular adaptive step-size algorithm. Any tunable parameters required by an adaptive step-size algorithm will be specified in the initialization function. We use a tabular representation for action-value functions on all finite MDPs and linear function approximation with a Fourier basis on all continuous MDPs [Konidaris et al., 2012]. Table A.1 in the appendix gives the basis order used by all algorithms for each domain.

4.3.1 Stochastic Gradient Descent Methods

In this section we derive adaptive step-size algorithms SID (Algorithm 7) and NOSID (Algorithm 8) using stochastic gradient descent (SGD). The first step in applying SGD to the scalar adaptive step-size problem for Sarsa(λ) is to determine what loss function Sarsa(λ) minimizes. When $\lambda = 1$, Sarsa(λ) minimizes the squared error from the Monte Carlo returns, in which case it becomes a stochastic gradient descent algorithm. However, when $\lambda < 1$, Sarsa(λ) no longer follows the stochastic gradient of any stationary loss function.

Algorithm 6 Sarsa(λ) with Adaptive Step Size

```
Given MDP $M = (S, \mathcal{A}, P, p, R, \gamma)$, and parameterized policy $\pi$
Initialize $\lambda \in [0, 1)$, $w_0 = 0$
InitStepSize($\cdots$)
$s_0 \sim p(\cdot)$,    $a_0 \sim \pi(s_0; w_0)$
for $t = 0, 1, 2 \ldots$ do
    $\pi_t = \pi(w_t)$
    $r_{t+1} \sim R(\cdot \mid s_t, a_t)$,    $s_{t+1} \sim P(\cdot \mid s_t, a_t)$,    $a_{t+1} \sim \pi_t(s_{t+1})$

    ▷ Sarsa($\lambda$) Update
    $\delta_t = r_t + \gamma Q_{w_t}(s_{t+1}, a_{t+1}) - Q_{w_t}(s_t, a_t)$
    $e_t = \gamma \lambda e_{t-1} + \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w}$
    $\alpha_t = $ StepSize($\cdots$)
    $w_{t+1} = w_t + \alpha_t \delta_t e_t$
end for
```

Following the derivation for TD(λ), recall that the corrected n-step discounted return at time step t is given by the random variable

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n Q_{t+n-1}(s_{t+n}, a_{t+n}).$$

From this, the λ-return is given by:

$$R_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}.$$

We can derive the Sarsa(λ) update using the sum-squared error between the λ-return, and the current action-value function, $Q_{t-1}(s_t, a_t)$ [Sutton and Barto, 1998a]. First, we simplify the λ-return error,

$$\begin{aligned} R_t^{\lambda} - Q_{t-1}(s_t, a_t) &= -Q_{t-1}(s_t, a_t) && +(1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}, \qquad (4.14) \\ &= -Q_{t-1}(s_t, a_t) && + (1 - \lambda)\lambda^0 [r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1})] \\ & && + (1 - \lambda)\lambda^1 [r_{t+1} + \gamma r_{t+2} + \gamma^2 Q_{t+1}(s_{t+2}, a_{t+2})] \\ & && + \cdots . \end{aligned}$$

This summation can be decomposed by separating out the individual rewards. For example, pulling out the first column of rewards gives:

$$(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}r_{t+1} = r_{t+1}(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1} = r_{t+1}(1-\lambda)\frac{1}{1-\lambda} = r_{t+1}.$$

This can be repeated for each reward column, although each time it is repeated an additional power of γ is included so that in general, for column $i \geq 1$,

$$(1-\lambda)\sum_{n=i}^{\infty}\lambda^{n-1}\gamma^{i-1}r_{t+i} = (\gamma\lambda)^{i-1}r_{t+i}(1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1} = (\gamma\lambda)^{i-1}r_{t+i}.$$

Summing over all reward columns gives

$$\sum_{n=1}^{\infty}(\gamma\lambda)^{n-1}r_{t+n}. \tag{4.15}$$

The remaining column in Equation 4.14 consists of action-value functions and can be simplified as

$$\begin{aligned}
&-Q_{t-1}(s_t,a_t) + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^{n}Q_{t+n-1}(s_{t+n},a_{t+n}) \\
&= -Q_{t-1}(s_t,a_t) + \sum_{n=1}^{\infty}\left[\lambda^{n-1}\gamma^{n}Q_{t+n-1}(s_{t+n},a_{t+n}) - (\gamma\lambda)^{n}Q_{t+n-1}(s_{t+n},a_{t+n})\right], \\
&= -Q_{t-1}(s_t,a_t) + \sum_{n=1}^{\infty}(\gamma\lambda)^{n-1}\left[\gamma Q_{t+n-1}(s_{t+n},a_{t+n}) - Q_{t+n-1}(s_{t+n},a_{t+n})\right], \\
&= \sum_{n=1}^{\infty}(\gamma\lambda)^{n-1}\left[\gamma Q_{t+n-1}(s_{t+n},a_{t+n}) - Q_{t+n-2}(s_{t+n-1},a_{t+n-1})\right]. \qquad (4.16)
\end{aligned}$$

Combining Equations 4.15 & 4.16 gives:

$$\begin{aligned}
\sum_{n=1}^{\infty}(\gamma\lambda)^{n-1}\left[r_{t+n} + \gamma Q_{t+n-1}(s_{t+n},a_{t+n}) - Q_{t+n-2}(s_{t+n-1},a_{t+n-1})\right] &= \sum_{n=1}^{\infty}(\gamma\lambda)^{n-1}\delta_{t+n-1}, \\
&= \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k.
\end{aligned}$$

This establishes an equivalence for the λ-return error that we can now use. Next, we take the derivative of the sum-squared λ-return error, $\mathcal{J}_\lambda(w)$, with respect to the action-value function weights w,

$$\begin{aligned}
\mathcal{J}_\lambda(w) &= \frac{1}{2}\sum_{t=0}^{\infty}[R_t^\lambda - Q_w(s_t,a_t)]^2, \qquad (4.17)\\
\frac{\partial}{\partial w}\mathcal{J}_\lambda(w) &= \sum_{t=0}^{\infty}\frac{\partial[R_t^\lambda - Q_{t-1}(s_t,a_t)]}{\partial w}[R_t^\lambda - Q_{t-1}(s_t,a_t)],\\
&= -\sum_{t=0}^{\infty}\frac{\partial Q_{t-1}(s_t,a_t)}{\partial w}\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k,\\
&= -\sum_{t=0}^{\infty}\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta_k\frac{\partial Q_{t-1}(s_t,a_t}{\partial w},\\
&= -\sum_{k=0}^{\infty}\sum_{t=0}^{k}(\gamma\lambda)^{k-t}\delta_k\frac{\partial Q_{t-1}(s_t,a_t)}{\partial w},\\
&= -\sum_{t=0}^{\infty}\delta_t\sum_{k=0}^{t}(\gamma\lambda)^{t-k}\frac{\partial Q_{k-1}(s_k,a_k)}{\partial w},\\
&= -\sum_{t=0}^{\infty}\delta_t e_t.
\end{aligned}$$

This produces the Sarsa(λ) update with eligibility traces (see Algorithm 6). The stochastic gradient methods for the scalar adaptive step-size problem take Equation 4.17 as the loss function and assume the existence of a locally optimal constant step-size sequence. Additionally, we use an exponential form for the step size, e^{α_t}, for both algorithms (SID and NOSID). This is fundamentally the same approach as taken by IDBD and Autostep [Mahmood et al., 2012a], except that IDBD and Autostep produce vector-valued step sizes and do not take into account the effects of eligibility traces. All of these SGD based adaptive step-size algorithms are extensions of, and approximations to, algorithms discussed in Chapter 2. They can be seen as extensions of the stochastic gradient adaptive (SGA) algorithm and as approximations to the general meta-optimization approach of Schraudolph [1999].

We derive the scalar IDBD step size for Sarsa(λ) (SID, Algorithm 7) by building upon the derivation of Sarsa(λ), and assuming that the optimal sequence of step sizes is constant or varies slowly over time. Such an assumption allows the use of stochastic gradient descent to incrementally move the current step size toward an unknown local optimum. Begin by taking the derivative of the loss function in Equation 4.17, at time step t, with respect to the step size:

$$\begin{aligned} \frac{\partial}{\partial \alpha} \mathcal{J}_\lambda(w_t) &= \frac{\partial}{\partial w_t} \mathcal{J}_\lambda(w_t) \frac{\partial w_t}{\partial \alpha}, \\ &= -\sum_{t=0}^{\infty} \delta_t e_t^T \frac{\partial w_t}{\partial \alpha}, \\ &= -\sum_{t=0}^{\infty} \delta_t e_t^T h(t), \end{aligned} \tag{4.18}$$

where $h(t) = \frac{\partial w_t}{\partial \alpha}$ is found by:

$$\begin{aligned} h(t+1) &= \frac{\partial}{\partial \alpha} \left[w_t + e^\alpha \delta_t e_t \right], \\ &= \frac{\partial w_t}{\partial \alpha} + e^\alpha \delta_t e_t + e^\alpha e_t^T \frac{\partial \delta_t}{\partial \alpha}, \\ &= h(t) + e^\alpha \delta_t e_t + e^\alpha e_t \left[\frac{\partial \delta_t}{\partial w_t}^T h(t) \right], \\ &= h(t) + e^\alpha \delta_t e_t + e^\alpha e_t \left[\gamma \frac{\partial Q_{w_t}(s_{t+1}, a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w} \right]^T h(t), \\ &= (1 + e^\alpha e_t \triangle\phi_t^T) h(t) + e^\alpha \delta_t e_t, \text{ where} \qquad (4.19) \\ \triangle\phi_t &= \left[\gamma \frac{\partial Q_{w_t}(s_{t+1}, a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w} \right]. \end{aligned}$$

Equations 4.18 and 4.19 together yield SID, given in Algorithm 7. The benefits of this approach are that the step size is fully adaptable and is able to increase or decrease as needed to minimize the squared λ-return error. Such methods are particularly robust in the presence of noise, however this comes at the cost of introducing a parameter—the meta-step size, $\beta_0 \in (0, 1]$, is the step size used by stochastic gradient

Algorithm 7 Scalar Incremental Delta-Bar-Delta for Sarsa(λ) (SID)

```
function InitSID($\alpha_0$, $\beta_0$, $d = size(w_0)$)
    $h = zeros(d)$
    $\alpha = \alpha_0$
    $\beta = \beta_0$
end function
function SID($\delta_t, e_t, \triangle\phi_t = \left[\gamma \frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    $\alpha = \alpha + \beta \delta_t e_t^T h$
    $h = (1 + e^\alpha e_t \triangle\phi_t^T)h + e^\alpha \delta_t e_t$
    return $e^\alpha$
end function
```

descent to optimize the step size e^{α_t}. Aside from this, there is another caveat to consider. Using stochastic gradient descent to optimize the step size requires the assumption of the existence of constant-valued locally optimal step-size sequences. That is, a sequence in which all entries share the same value. This can be extended to allow α^* to vary slowly over time, but does not allow for substantially different step sizes from one time step to the next. This assumption may not be true in general, and when violated can lead to worsening performance and even function approximation divergence.

Autostep extends the original IDBD algorithm to reduce its sensitivity to the meta-step-size parameter [Mahmood et al., 2012a]. Their approach is an attempt to dynamically normalize the updates to the step size, but also introduces a second new parameter which adjusts the horizon of the normalization. A more principled solution can be found by using the online normalization approach of Ross et al. [2013]. Thus, the Normalized Scalar IDBD (NOSID, Algorithm 8) algorithm is an extension to SID inspired by Autostep, but using the normalized online learning algorithm adapted for a scalar step size. NOSID also requires a meta-step-size parameter, $\beta_0 \in (0, 1]$.

4.3.2 Variance Estimating Step Sizes for Sarsa(λ) (VES)

The Variance Estimating Step-Size algorithm (VES) builds upon the derivation of the variance-based SGD (vSGD) algorithm [Schaul et al., 2012]. Recently, Ranganath

Algorithm 8 Normalized Scalar Incremental Delta-Bar-Delta for Sarsa(λ) (NOSID)

```
function InitNOSID($\alpha_0$, $\beta_0$, $d = size(w_0)$)
    $h, s = zeros(d)$
    $\alpha = \alpha_0$
    $\beta = \beta_0$
    $N = 0$
end function
function NOSID($\delta_t, e_t, \triangle\phi_t = \left[\gamma \frac{\partial Q_{w_t}(s_{t+1}, a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w}\right]$)
    for $i = 0, 1, \ldots, size(s)$ do
        if $|\triangle\phi_t[i]| > s[i]$ then
            $s[i] = |\triangle\phi_t[i]|$
        end if
    end for
    $N = N + \frac{\|\triangle\phi_t\|^2}{\|s\|^2}$
    $\alpha = \alpha + \beta \frac{t}{Ns^2} \delta_t e_t^T h$
    $\alpha = \textsc{Min}(\alpha,\ 0)$
    $M = \textsc{Max}(-exp(\alpha) \triangle\phi_t^{\top} e_t,\ 1.0)$
    $\alpha = \alpha - \log M$
    $h = (1 + e^{\alpha} e_t \triangle\phi_t^T) h + e^{\alpha} \delta_t e_t$
    return $e^{\alpha}$
end function
```

et al. [2013] derived an adaptive scalar step size for stochastic variational inference. These two methods are actually identical with respect to the adaptive step size, but provide different methods for adaptively correcting the update direction of the stochastic gradient update. vSGD assumes the loss function is an expectation over quadratic losses:

$$\mathcal{J}(w) = \mathbb{E}_{j \sim \mathcal{P}}[\frac{1}{2}(w - \hat{w}_j)^T H_j (w - \hat{w}_j)], \tag{4.20}$$

where both the instantaneous Hessian, H_j, and the covariance matrix of the per sample optima, $Cov(\hat{w}_j, \hat{w}_j)$, are diagonal matrices. By comparison, Ranganath et al. [2013] assume that they are taking the natural gradient, and the loss function is quadratic with a Hessian equal to the identity matrix. Thus, without making the distinction explicit, Ranganath et al. [2013] have done as we propose and split their algorithm into an adaptive scalar step size and an update whitening procedure, through the use of natural gradients.

Although the derivations are substantially different and VES includes improvements that allow it to be parameter-free, VES and vSGD use fundamentally the same approach. Thus, VES should be considered as vSGD applied to reinforcement learning with both practical and theoretical improvements. vSGD uses a *slow start* heuristic to initialize the expected value estimates. If this procedure is run for too short a time or the overestimation factor C is not chosen well, then the memory size estimation, τ, will sometimes collapse toward unity and be unable to change once that value is reached.[1] This has a catastrophic effect on the algorithm overall and tends to result in function approximation divergence. VES eliminates the need for this heuristic and its accompanying parameter completely, and as a result the memory size estimation is more robust. The result is a parameter-free adaptive step-size algorithm, although the other Sarsa(λ) parameters remain.

We derive VES under the assumption that the optimal step-size sequence, $\{\alpha_t\}_{t=0}^{\infty}$, as well as the loss function being minimized may be non-stationary. To approach this much harder version of the adaptive step-size problem requires assuming that the loss function can be expressed as a sum of noisy quadratic loss functions. For this reason, we use the loss function given by the error from the LSTD(λ) solution. Finally, the step size is derived in terms of expected values that are never fully observed, and must instead be estimated.

We begin by considering the derivation of the Least Squares Temporal Difference learning (LSTD) algorithm [Bradtke and Barto, 1996, Boyan, 1999, Geramifard et al., 2007]. Let $\mu_t(w_t)$ be the expected update for Sarsa(λ) at time step t, and assume the action-value function approximation is linear with $\frac{\partial Q_w(s_t,a_t)}{\partial w} = \phi_t$,

[1] Schaul et al. [2012] suggest using $C = d/10$ as a rule of thumb and while this works well in supervised learning, experiments suggest that the setting is not particularly robust in the RL setting.

$$
\begin{aligned}
\mu_t(w_t) &= \mathbb{E}_t[\delta_j e_j], \\
&= \mathbb{E}_t[e_j(r_{j+1} + \gamma Q_{w_t}(s_{j+1}, a_{j+1}) - Q_{w_t}(s_j, a_j))], \\
&= \mathbb{E}_t[e_j(r_{j+1} + [\gamma \phi_{j+1} - \phi_j]^T w_t)], \\
&= \mathbb{E}_t[e_j(r_{j+1} - [\phi_j - \gamma \phi_{j+1}]^T w_t)], \\
&= \mathbb{E}_t[e_j r_{j+1}] - \mathbb{E}_t[e_j(\phi_j - \gamma \phi_{j+1})^T] w_t, \\
&= b_t - A_t w_t,
\end{aligned}
$$

where $b_t = \mathbb{E}_t[e_j r_{j+1}]$ and $A_t = \mathbb{E}_t[e_j(\phi_j - \gamma\phi_{j+1})^T]$. The LSTD solution is given by

$$w_t^* = A_t^{-1} b_t. \tag{4.21}$$

Then, notice the error between the current weight vector and the LSTD solution is

$$
\begin{aligned}
w_t^* - w_t &= A_t^{-1} b_t - w_t, \\
&= A_t^{-1} \mu_t(w_t), \\
=> \mu_t(w_t) &= A(w_t^* - w_t), \\
&= \mathbb{E}_t[w_t - \hat{w}_t],
\end{aligned}
$$

where $w_t - \hat{w}_t$ is some per step noisy estimate of the update direction towards the LSTD solution. Then the loss function for LSTD can be expressed by:

$$\mathcal{J}_{LSTD}(w_t) = (w_t^* - w_t)^T A_t^2 (w_t^* - w_t). \tag{4.22}$$

With this form of loss function and the incorrect assumption that A_t^2 is diagonal, the same derivation as Schaul et al. [2012] may be used. Instead, we work under the assumption that the adaptive scalar step-size and update whitening problems can be performed sequentially. Assume that the whitened update is used, then the loss function to be minimized by the adaptive step-size algorithm becomes $\mathcal{J}(w_{t+1}) = \|w_{t+1} - w_t^*\|^2$, where $w_{t+1} = w_t + \alpha_t \hat{\triangle} w_t$ and $\hat{\triangle} w_t$ is the whitened update. This approach can be viewed as choosing the step size such that the resulting update minimizes the distance from the LSTD solution. First, take the expected value of the next step loss, $\mathcal{J}(w_{t+1})$,

$$\begin{aligned}
\mathbb{E}[\mathcal{J}(w_{t+1})|w_t, \alpha_t] &= \mathbb{E}[\|w_{t+1} - w_t^*\|^2 | w_t, \alpha_t], \\
&= \mathbb{E}[\|(1+\alpha_t)w_t - (\alpha_t \hat{w}_t + w_t^*)\|^2], \\
&= (1+\alpha_t)^2\|w_t\|^2 - 2(1+\alpha_t)^2 w_t^T w_t^* \\
&\qquad + (1+\alpha_t)^2 \|w_t^*\|^2 + \alpha_t^2 \mathbb{E}[\|\hat{w}_t - w_t^*\|^2], \\
&= (1+\alpha_t)^2 \|w_t - w_t^*\|^2 + \alpha_t^2 \mathbb{E}[\|\hat{w}_t - w_t^*\|^2], \\
&= (1-\alpha_t)^2 \|w_t^* - w_t\|^2 + \alpha_t^2 Var(\hat{w}_t).
\end{aligned} \tag{4.23}$$

Setting the derivative with respect to α_t of Equation 4.23 equal to zero and solving for α_t gives the adaptive step-size solution,

$$\begin{aligned}
0 &= \frac{\partial}{\partial \alpha_t} \mathbb{E}[\mathcal{J}(w_{t+1})|w_t, \alpha_t], \\
&= 2(1-\alpha_t)\|w_t^* - w_t\|^2 + 2\alpha_t Var(\hat{w}_t), \\
\implies \alpha_t &= \frac{\|w_t^* - w_t\|^2}{\|w_t^* - w_t\|^2 + Var(\hat{w}_t)}.
\end{aligned} \tag{4.24}$$

Finally, we substitute in the expected value of the whitened update to get the adaptive scalar step size in terms of estimated expected values,

$$\alpha_t = \frac{\|\mathbb{E}_j[\hat{\triangle} w_j]\|^2}{\mathbb{E}_j[\|\hat{\triangle} w_j\|^2]}. \tag{4.25}$$

This adaptive step size still leaves two questions to be answered before a practical implementation is reached. First, how are the expected values estimated? This is particularly important in RL because the distribution over states and actions varies as the agent moves towards an increasingly optimal policy. Second, what update whitening method is used? The latter is the subject of the next chapter, which studies the Natural Temporal Difference Learning (NaTD) class of algorithms. However, for now we simply use a scalar approximation, $G_t^{-1} \approx \mathbb{E}[\|\triangle\phi_t\|^2]$. Estimating the expected values is done with an extension to the adaptive sliding window method proposed by Schaul et al. [2012]. Together these give Algorithm 9.

Algorithm 9 Variance Estimating Step Sizes for Sarsa(λ) (VES)

```
function InitVES($d = size(w_0)$)
    $v = zeros(d)$
    $g, h, \tau, \tau_0 = 1.0, 1.0, 2.0, 1.0$
end function
function VES($\delta_t, e_t, \triangle\phi_t = \left[\gamma \frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    $v = v + \frac{1}{\tau}(\|\delta_t e_t\|^2 - v)$
    $g = g + \frac{1}{\tau}(\delta_t e_t - g)$
    $h = h + \frac{1}{\tau}(\|\triangle\phi_t\|^2 - h)$
    $\alpha = \textsc{Min}(\frac{\|g\|^2}{vh}, 1.0)$
    $\tau_0 = \tau_0 + \alpha^2$
    $\tau = (1 - \frac{\|g\|^2}{v})\tau + \tau_0$
    return $\alpha$
end function
```

4.3.3 Passive-Aggressive Updates for Reinforcement Learning (PARL)

The final approach we study in this chapter is found by first deriving a step size that minimizes the one-step squared Bellman error. Such a step size *aggressively* minimizes the loss (squared Bellman error) of the current transition without regard for how this affects the loss on past or future transitions. By contrast, a *passive* update would have a step size at or near zero. The idea of switching or scaling between passive and aggressive updates is inspired by the work of Crammer et al. [2006] on passive-aggressive algorithms for online learning.

The Passive-Aggressive Reinforcement Learning (PARL) algorithms share a common set of assumptions. The first assumption is that the loss function is the immediate squared Bellman error:

$$\mathcal{J}_\delta(w_t) = \frac{1}{2}\left[r_{t+1} + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)\right]^2 |_{w=w_t+\alpha_t\delta_t e_t}. \tag{4.26}$$

The second assumption is that the sequence of step sizes is unconstrained other than requiring $\alpha_t \in (0,1],\ \forall t$. The third assumption is that the function approximation method is linear. This approach finds the step size that aggressively minimizes Equation 4.26 by setting the derivative with respect to α to zero:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \alpha}\mathcal{J}_\delta(w_t), \\ &= \frac{\partial}{\partial \alpha}\left[\frac{1}{2}(r_{t+1} + \triangle\phi_t^T(w_t + \alpha\delta_t e_t))^2\right], \\ &= \frac{\partial}{\partial \alpha}\frac{1}{2}\delta_t^2(1 + \alpha\triangle\phi_t^T e_t)^2, \\ &= \delta_t^2(1 + \alpha\triangle\phi_t^T e_t)(\triangle\phi_t^T e_t), \\ \implies \alpha_t &= \frac{-1}{e_t^T\triangle\phi_t}. \end{aligned} \tag{4.27}$$

It is trivial to verify that if using this step size to update the weights, w_t, the Bellman error on the current transitions, $r_{t+1} + \triangle\phi_t^T w_{t+1}$, will vanish. However, in practice this may be undesirable for a number of reasons. The resulting step size may be negative when the eligibility traces happen to be aligned with the residual gradient, in which case the step size will go negative and the weights will move in the opposite direction from what Sarsa(λ)'s update requires.

The situation can be better understood by considering an alternate objective function, the ratio of the post-update squared Bellman error to the pre-update squared

Bellman error, and bounding this between zero and one. Let $\delta'_t = r_{t+1} + \gamma Q_{w_{t+1}}(s_{t+1}, a_{t+1}) - Q_{w_{t+1}}(s_t, a_t)$, then assume that:

$$0 \leq \left(\frac{\delta'_t}{\delta_t}\right)^2 < 1. \tag{4.28}$$

Then we expand this inequality by substituting in the definitions for δ_t and δ'_t:

$$\begin{aligned} &\left(\frac{\delta'_t}{\delta_t}\right)^2 = (1 + \alpha \triangle\phi_t^T e_t)^2, \\ \implies\ & 0 \leq 1 + 2\alpha e_t^T \triangle\phi_t + \alpha^2 (e_t^T \triangle\phi_t)^2 < 1. \end{aligned}$$

By considering the right inequality, $1 + 2\alpha e_t^T \triangle\phi_t + \alpha^2 (e_t^T \triangle\phi_t)^2 < 1$, as a quadratic equation in α with solutions

$$\alpha = \frac{-2}{e_t^T \triangle\phi_t}, \quad \text{or} \quad \alpha = 0.$$

It is clear that there exists a region depending on the value of $e_t^T \triangle\phi_t$ in which the step size will result in lower squared Bellman error on the current transition. These regions are depicted in Figure 4.1, which shows that when $e_t^T \triangle\phi_t < 0$ the region is $(0, 2/|e_t^T \triangle\phi_t|)$ and when $e_t^T \triangle\phi_t > 0$ the region is $(-2/|e_t^T \triangle\phi_t|, 0)$.

Each transition provides a different aggressive step-size region which contains only positive or only negative step sizes. The immediate application of this observation and step-size rule yields an algorithm that updates with the aggressive step size (Equation 4.27) if $e_t^T \triangle\phi_t < 0$, and otherwise makes no update (i.e., behaves passively). However, this seems particularly wasteful from the perspective of efficient use of experiences. Instead of making no update at all, one might choose to make a very small update. This leads to the first passive aggressive algorithm for RL (PARL1, Algorithm 10), which in place of making no update instead uses the smallest aggressive step size observed so far.

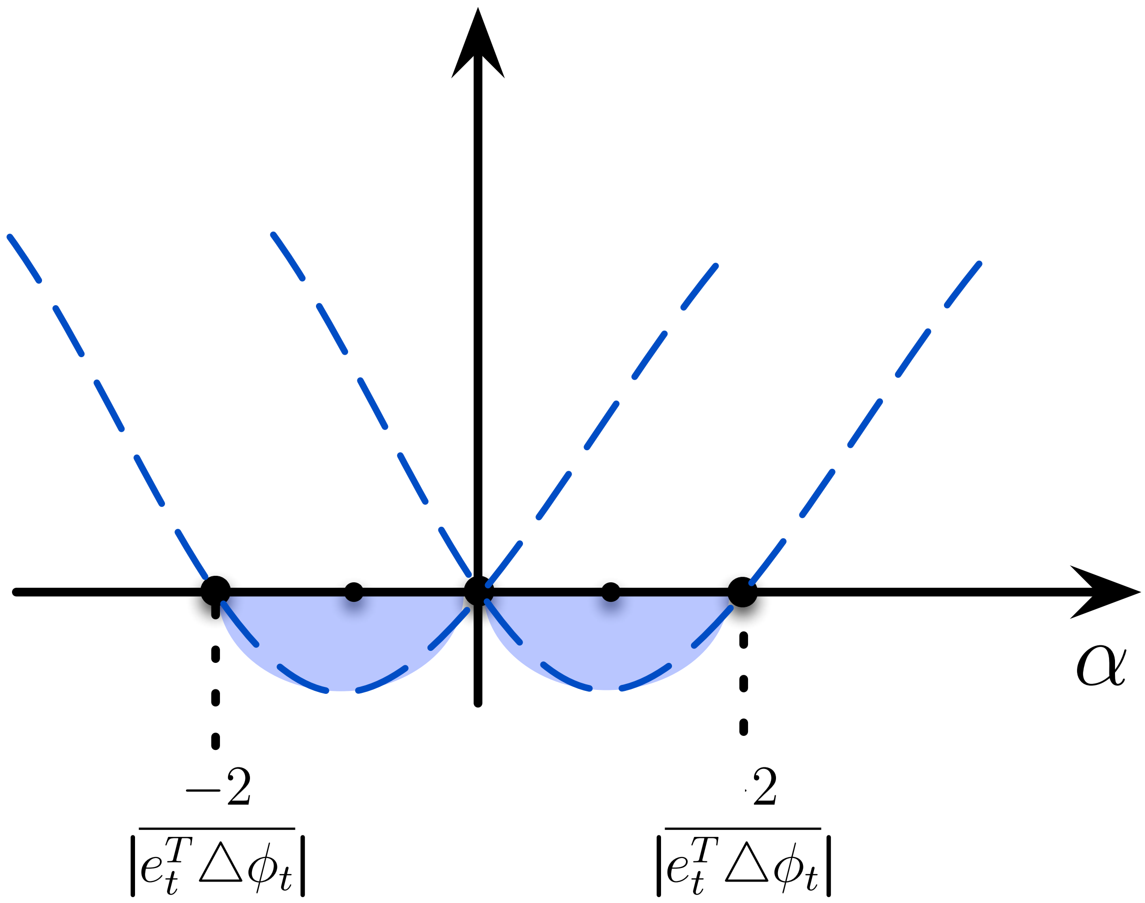

α
$\frac{-2}{|e_t^T \triangle \phi_t|}$
$\frac{2}{|e_t^T \triangle \phi_t|}$

Figure 4.1: Aggressive step-size regions.

However, sometimes the aggressive step size is absurdly large, and while it correctly minimizes the immediate loss it will sometimes do so too aggressively resulting in a phenomenon which may be described as *divergent over-fitting*. Divergent over-fitting is when each training example is fit perfectly, resulting in zero loss on that example, but also fitting any noise in the example. This tends to produce large weights and ever increasing errors on subsequent examples. When this happens repeatedly the magnitude of the weights continue to increase until eventually reaching the floating point precision limit and causing function approximation divergence. One solution is to require that the adaptive algorithm become increasingly passive over time. That is, as training continues and the current weights encode more informa-

Algorithm 10 Passive-Aggressive RL for Sarsa(λ) (PARL1)

```
function InitPARL1($d = size(w_0)$)
    $\alpha = 1.0$
    $\hat{\alpha} = 1.0$
end function
function PARL1($\delta_t, e_t, \triangle\phi_t = \left[\gamma\frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    if $e_t^T \triangle\phi_t < 0$ then
        $\alpha = \frac{-1}{e_t^T \triangle\phi_t}$
    else
        $\alpha = \hat{\alpha}$
    end if
    $\hat{\alpha} = $ Min($\hat{\alpha}$, $\alpha$)
    return $\alpha$
end function
```

tion, require that the step size is non-increasing while still ensuring that on any given update the step size is within the aggressive region. Doing so produces the novel adaptive step-size algorithm PARL2 (Algorithm 11).

Algorithm 11 Passive-Aggressive RL for Sarsa(λ) (PARL2)

```
function InitPARL2($d = size(w_0)$)
    $\alpha = 1.0$
end function
function PARL2($\delta_t, e_t, \triangle\phi_t = \left[\gamma\frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    if $e_t^T \triangle\phi_t < 0$ then
        $\alpha = $ Min($\alpha$, $\frac{-1}{e_t^T \triangle\phi_t}$)
    end if
    return $\alpha$
end function
```

PARL2 produces a sequence of non-increasing step sizes. However, in the presence of high variance rewards the step size could be driven down early on without any way to later increase. This could lead to slower learning speeds if the aggressive region is underestimated early-on in the agent's lifetime. Instead, an algorithm may dynamically shift between more aggressive or more passive updates.

This dynamic rescaling is motivated by considering the case where $0 < -e_t^\top \triangle\phi_t \leq 1$, which corresponds to aggressive step sizes greater than one. This is equivalent to when the angle between the the Sarsa(λ) update and the residual gradient is within $[0°, 90°)$. As the angle between the two updates goes to zero, it becomes increasingly appropriate to use an adaptive step size suited for the Residual Gradient update. The aggressive step size for Residual Gradient is $\frac{1}{\|\triangle\phi_t\|^2}$, which follows directly from the derivation of the online passive aggressive algorithm [Crammer et al., 2006]. However, even as the angle between the two updates may go to zero, the magnitudes of the two updates can still differ. Instead, averaging the two takes this into account and gives $2(\|\triangle\phi_t\|^2 + \|\triangle e_t\|^2)^{-1}$. These two approaches are combined to provide an adaptive step size that dynamically rescales the aggressive step size, assuming $e_t^\top \triangle\phi_t < 0$,

$$\begin{aligned}\alpha_t &= \frac{1}{-e_t^\top \phi_t + (1/2)\left[\|\triangle\phi_t\|^2 + \|\triangle e_t\|^2\right]},\\ &= \frac{2}{-2e_t^\top \phi_t + \|\triangle\phi_t\|^2 + \|\triangle e_t\|^2}.\end{aligned}$$

Simplifying the expression gives the adaptive step-size algorithm PARL3 (Algorithm 12),

$$\alpha_t = \frac{2}{\|e_t - \triangle\phi_t\|^2}. \tag{4.29}$$

Algorithm 12 Passive-Aggressive RL for Sarsa(λ) (PARL3)

```
function InitPARL3($d = size(w_0)$)
    $\alpha = 1.0$
    $\hat{\alpha} = 1.0$
end function
function PARL3($\delta_t, e_t, \triangle\phi_t = \left[\gamma\frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    if $e_t^T \triangle\phi_t < 0$ then
        $\alpha = \frac{2}{\|e_t - \triangle\phi_t\|^2}$
    else
        $\alpha = \hat{\alpha}$
    end if
    $\hat{\alpha} = $ Min($\hat{\alpha}$, $\alpha$)
    return $\alpha$
end function
```

All three passive aggressive algorithms have the advantage of not requiring any tunable parameters. However, as previously mentioned, PARL1 can produce divergent sequences. For this reason PARL1 is not used in the empirical studies and is included only for illustrative purposes.

4.4 Algorithms for Comparison

For all experiments the Sarsa(λ) algorithm was used while varying the way step sizes are generated. When a fixed scalar step-size parameter was used this is referred to as Sarsa(λ), while other variants are denoted by the name of the step-size algorithm. These methods are also compared to three deterministic step-size schedules, and two existing adaptive step-size algorithms.

4.4.1 Deterministic Step-Size Schedules

The first of the deterministic schedules is the Generalized Harmonic Step Size (GHC, Algorithm 13) [George and Powell, 2006a], and requires parameters $\alpha_0 \in (0, 1]$, an initial step size, and $\tau \in \mathbb{N}$, which affects the rate at which the step sizes decay. This is the general form of the often referenced $1/t$ step-size schedule.

Algorithm 13 Generalized Harmonic Step Size (GHC)

```
function InitGHC($\alpha_0$, $\tau$)
    $\alpha_0 = \alpha_0$
    $\tau = \tau$
end function
function GHC($t$)
    return $\alpha_0 \frac{\tau}{\tau+t-1}$
end function
```

The second step-size schedule is Search Then Converge (STC, Algorithm 14) [Darken and Moody, 1992]. STC takes parameters α_0, an initial step size, $c > 0$ and $\tau \in \mathbb{N}$. When t is large relative to τ the step size decreases at the rate of c/t, while using a larger step size when t is small relative to τ.

Algorithm 14 Search Then Converge (STC)

```
function InitSTC($\alpha_0$, $\tau$, $c$)
    $\alpha_0 = \alpha_0$
    $\tau = \tau$
    $c = c$
end function
function GHC($t$)
    return $\alpha_0 \frac{1+\frac{ct}{\alpha_0 \tau}}{1+\frac{ct}{\alpha_0 \tau}+\frac{t^2}{\tau}}$
end function
```

The third step-size schedule used for comparison, called *Boyan's step size*, was designed for RL algorithms and is due to Geramifard et al. [2007]. Boyan's step size is given by Algorithm 15 with parameters α_0, the initial step size, N_0 a parameter affecting the rate at which the step size decays, and where "Episode#" denotes which episode number the algorithm is on (starting with 1).

Algorithm 15 Boyan's Step Size (Boyan)

```
function InitBoyan($\alpha_0$, $N_0$)
    $\alpha_0 = \alpha_0$
    $N_0 = N_0$
end function
function Boyan($t$)
    return $\alpha_0 \frac{N_0+1}{N_0+(\text{Episode\#})^{1.1}}$
end function
```

4.4.2 Adaptive Step Sizes

Empirical comparisons are also given for two existing adaptive step-size algorithms for RL (Autostep and HL(λ)). Autostep [Mahmood et al., 2012a], as previously discussed, is an extension to the adaptive vector-valued step-size algorithm IDBD, and attempts to reduce parameter sensitivity by using normalized stochastic gradient descent to incrementally improve the step size.

The SGD adaptive scalar step-size algorithms presented earlier are the same fundamental approach as Autostep, but with differences in the loss function and how normalization is done. Both the SGD adaptive step sizes and Autostep require additional parameters in the form of a meta-step size, and in the case of Autostep a second new parameter which affects the normalization procedure. Mahmood et al. [2012a] find that the optimized values for these parameters are constant across the problems they consider ($\mu = 0.01$ and $\tau = 10000$). Autostep is given by Algorithm 16.[2] However, their research focused exclusively on prediction problems and these parameter values do not generalize well to the control learning setting studied in this chapter. Thus, these two parameters both required tuning for each domain shown.

Algorithm 16 Autostep

```
function INITAUTOSTEP($\alpha_0$, $\mu$, $\tau$, $d = size(w_0)$)
    $h, v = zeros(d)$
    $\alpha = ones(d) * \alpha_0$
    $\mu = \mu$
    $\tau = \tau$
end function
function AUTOSTEP($\delta_t, e_t, \triangle\phi_t = \left[\gamma\frac{\partial Q_{w_t}(s_{t+1},a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t,a_t)}{\partial w}\right]$)
    $v = \textsc{Max}(|\delta_t e_t h|,\ v + \frac{1}{\tau}\alpha * e_t^2(|\delta_t e_t| - v))$
    for $i = 0, 1, \ldots, d$ do
        if $v[i]! = 0$ then
            $\alpha[i] = \alpha[i]e^{\frac{\mu\delta_t e_t[i]}{v[i]}}$
        end if
    end for
    $M = \textsc{Max}(\alpha^\top(e_t)^2,\ 1.0)$
    $\alpha = \frac{\alpha}{M}$
    $h = h * (1.0 - \alpha * e_t^2) + \alpha\delta_t e_t$
    return $\alpha$
end function
```

HL(λ) [Hutter and Legg, 2007], previously discussed in the background chapter, was derived as a parameter-free optimal step size for Sarsa(λ) on finite state MDPs and is given by Algorithm 17. HL(λ) is only used for comparisons on finite state MDPs in the RL Benchmark.

[2] Autostep does not explicitly account for eligibility traces, but the parameter tuning was allowed to vary the value $\lambda \in [0, 1)$ and showed improved performance for non-zero values.

Algorithm 17 HL(λ)

```
function InitHL($d = size(w_0)$)
    $N = ones(d)$
end function
function HL($\delta_t, e_t, \triangle\phi_t = \left[\gamma \frac{\partial Q_{w_t}(s_{t+1}, a_{t+1})}{\partial w} - \frac{\partial Q_{w_t}(s_t, a_t)}{\partial w}\right]$)
    $\alpha(s, s_{t+1}) = \frac{1}{N^t_{s_{t+1}} - \gamma e^t_{s_{t+1}}} \frac{N^t_{s_{t+1}}}{N^t_s}$
    $N^{t+1}_s = \lambda N^t_s + \delta_{s_{t+1}, s}$, where $\delta_{s,s'}$ is the Kronecker delta.
    return $\alpha$
end function
```

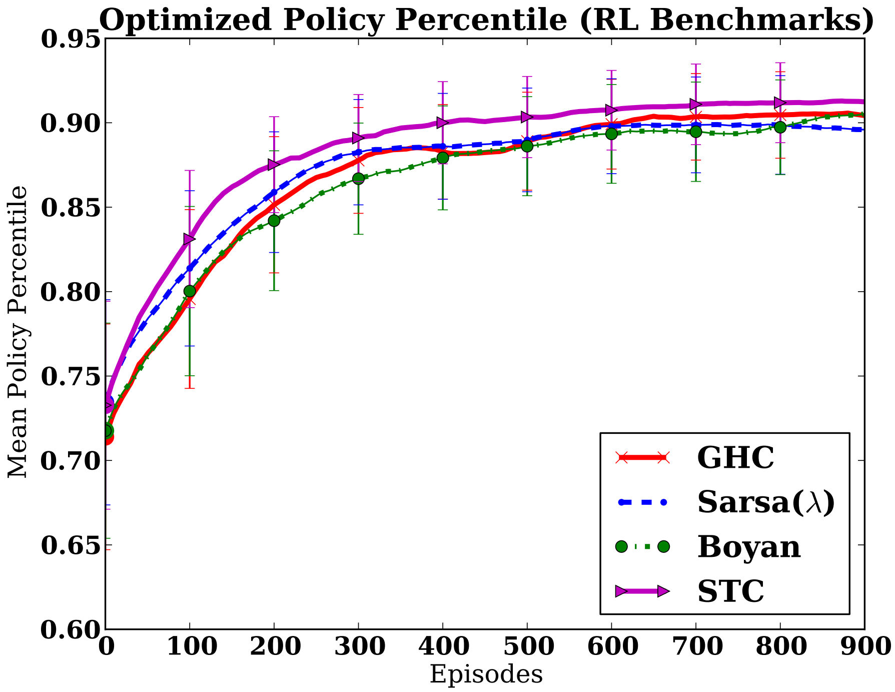
4.5 Empirical Results

We used the methods given in Chapter 3 to produce the following empirical results. Specifically, for each of the fifteen MDPs in the RL Benchmark set, we ran $K = 5$ independent randomized parameter optimizations of length $N = 100$. Each parameter evaluation was an average over 5 runs and, once found, optimized parameter values were then evaluated for a total of $M = 30$ runs.

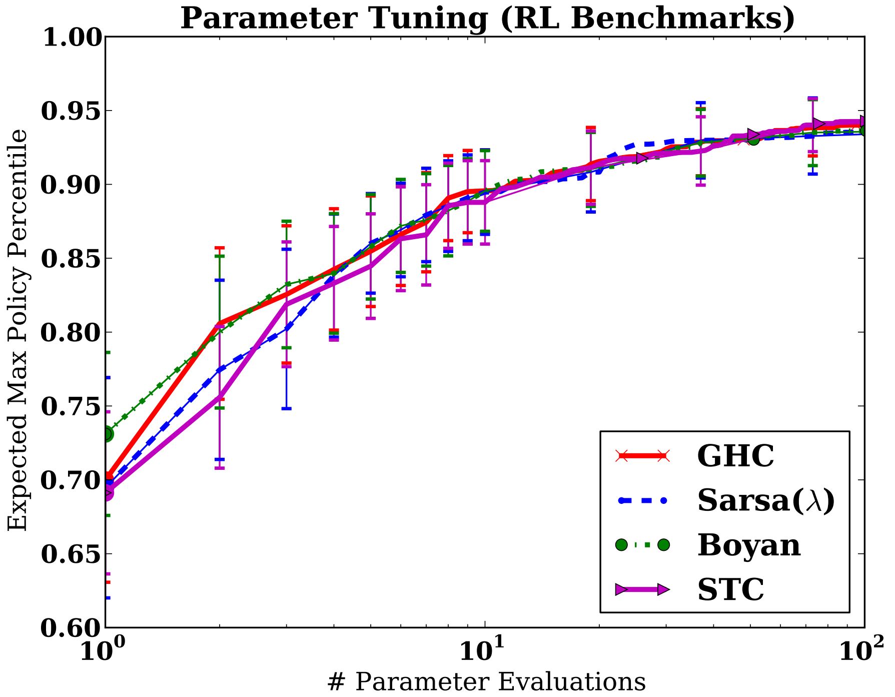
The figures provided show two things. First, the average policy percentile and standard error for each algorithm using optimized parameter values and averaged over M runs. These figures are titled with "Optimized Policy Percentile" followed by the set of MDPs the results are averaged over. Second, the average maximum policy percentile and standard error for each algorithm over the K parameter optimizations. These figures are titled with "Parameter Tuning", and are similarly followed by the set of MDPs over which the results are averaged. Generally this is the RL Benchmark set, although we also give results for individual domains when they provide interesting insights into the algorithms. Finally, due to the large number of algorithms involved in the study, we break the results into sections that are restricted to comparing related algorithms.

4.5.1 Deterministic Step-Size Schedules

We begin by comparing the deterministic step-size schedules for Sarsa(λ) with a fixed step-size. Figure 4.2a shows the performance with optimized parameter values and Figure 4.2b shows the difficulty and impact of the parameter tuning process. The Search-Then-Converge (STC) step-size schedule shows a large improvement, on average, over the other methods, but is simultaneously one of the most difficult to properly optimize.



(a) Performance with optimized parameters



(b) Difficulty and Impact of parameter tuning

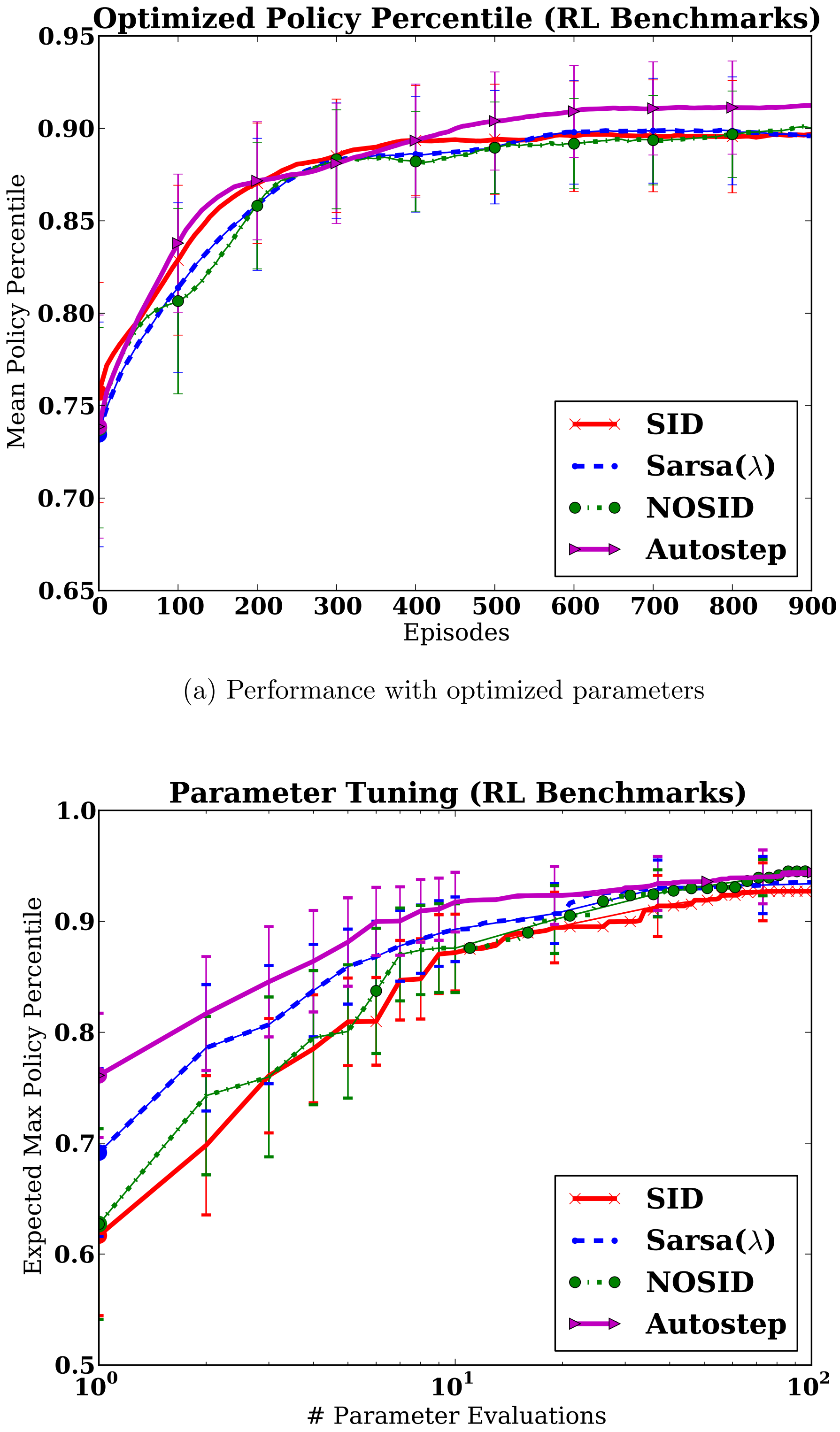
Figure 4.2: Deterministic step-size schedules and Sarsa(λ), averaged over the RL Benchmark set.

By comparison, Boyan's step-size schedule has the best average case performance when parameters are selected randomly, and provides the simplest parameter meta-optimization problem. Considering the MDPs separately, there were a handful in which one method or the other resulted in statistically significantly higher policy percentiles, but none had a statistically significant advantage when averaged over the entire RL Benchmark set.

4.5.2 Stochastic Gradient Descent Methods for Step Sizes

We next turn to the SGD based methods for adaptive step sizes, shown in Figure 4.3. The methods derived in this chapter are also compared with Autostep, which produces vector-valued step sizes, and is fairly consistently among the best performing algorithms in this set. It is interesting to notice that the normalized algorithm, NOSID, does not confer any improvement in performance when averaged over the entire RL Benchmark set. However, on a few of the individual MDPs NOSID is among the best, and on the HIV Treatment domain in particular it is able to overcome the high variance rewards while SID diverges. The HIV Treatment domain is useful for testing how algorithms react to especially large reward signals. SID uses an exponential form for the step size which often results in overflows on this domain when the updates are not normalized carefully.

While the more complex NOSID algorithm does confer significant advantages on some problems, the cost of the complexity outweighs the benefits when measured over the whole benchmark set. We can view these results as a baseline for adaptive step sizes on RL Benchmark. SID is due to the application of SGA to RL, NOSID is a normalized version of the same, and we include Autostep to show how these methods may be expected to perform when combined with an approximate update whitening algorithm.



(a) Performance with optimized parameters

(b) Difficulty and Impact of parameter tuning

Figure 4.3: SGD adaptive step sizes and Sarsa(λ), averaged over the RL Benchmark set.

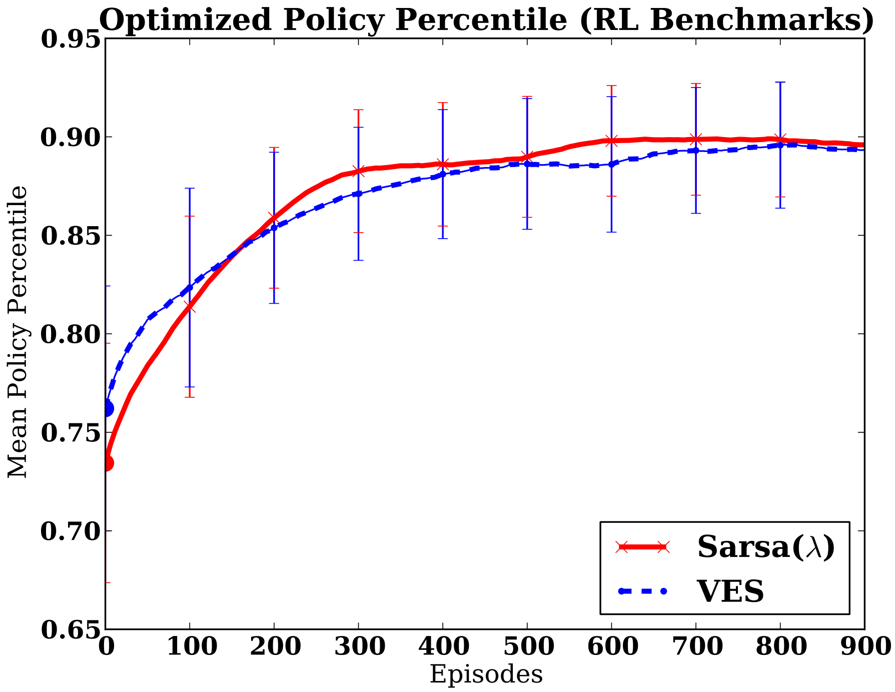
4.5.3 Variance Estimation Method for Step Sizes

In Figures 4.4 we compare the performance of Sarsa(λ) with a fixed step size with the performance of VES. VES has no tunable step-size parameter performs at least as well over the RL Benchmark as Sarsa(λ) with optimized parameter values. More importantly, Figure 4.4b shows that VES is much less difficult to parameter tune. Unlike in the case of SID and NOSID, which showed greater difficulty, VES eliminates the step-size parameter entirely, leaving only the other Sarsa(λ) parameters (λ and ϵ) as tunable parameters.

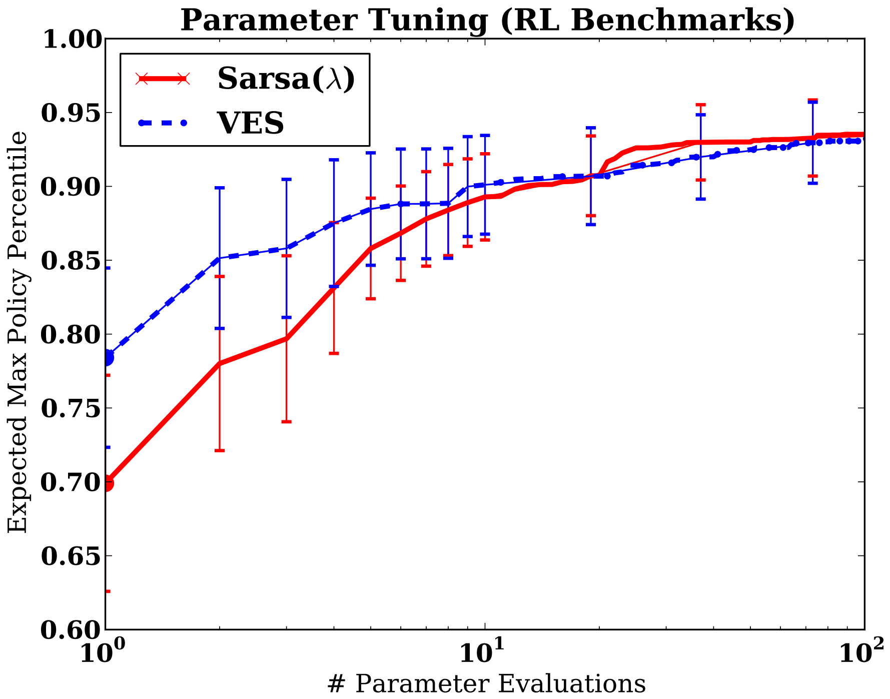
4.5.4 Passive-Aggressive Method for Step Sizes

Figure 4.5 shows two of the passive aggressive algorithms for RL (PARL2 and PARL3) compared with Sarsa(λ). Recall that PARL3 uses less aggressive step sizes in general, but generates a step-size sequence that can change freely within $(0, 1]$ whereas PARL2 can only generate non-increasing sequences of step sizes. Like VES both of these algorithms eliminate the step size as a tunable parameter. From the performance on the entire domain set we can say that PARL2 suffers no loss in performance compared with Sarsa(λ). However, this is not the full story.

PARL2, PARL3, and VES all completely eliminate the need for step-size tuning, but all three appear to under-perform on discrete state MDPs. Not that they do poorly, but Sarsa(λ) does better than would be expected from results on continuous MDPs. The reason is that in finite MDPs we do not use any function approximation in our experiments and each update can be made exactly. This means that much larger step sizes can be used without causing divergence, which in turn speeds up the learning process. These adaptive step size algorithms are, in essence, being too cautious in the finite MDP case while Sarsa(λ) can be tuned to use much larger step sizes. The effect in either direction is small but noticeable enough to be seen in



(a) Performance with optimized parameters



(b) Difficulty and Impact of parameter tuning

Figure 4.4: VES and Sarsa(λ), averaged over the RL Benchmark set.

Figure 4.6 which shows performance averaged over only the continuous state MDPs in RL Benchmark.

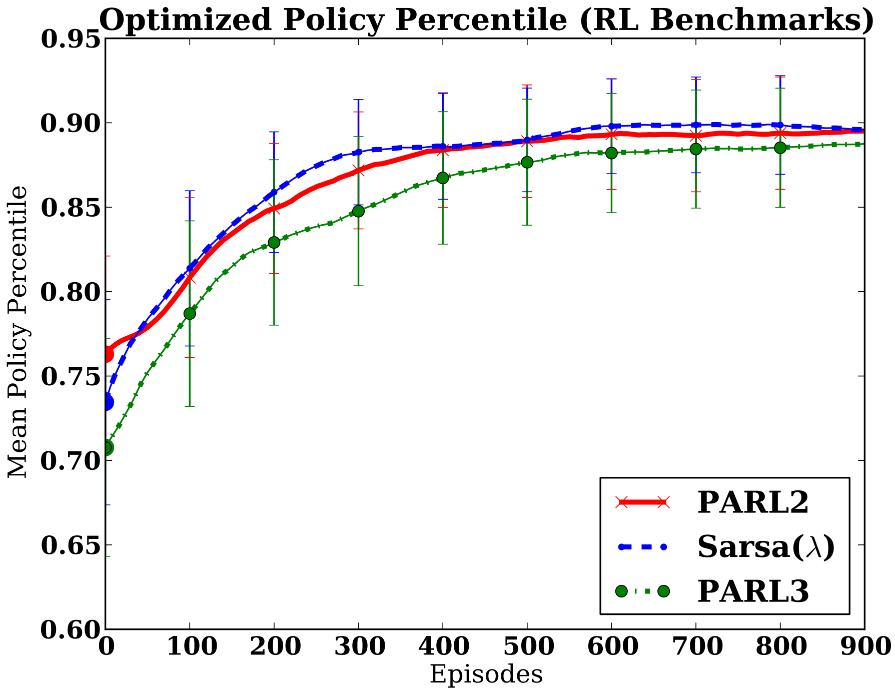
PARL2 appears to perform better that Sarsa(λ) throughout most of the RL Benchmark set, while PARL3 performs slightly worse on average. Both adaptive algorithms are significantly easier to parameter tune than Sarsa(λ) due to eliminating the tunable step size without introducing any other parameters.

4.5.5 Finite MDPs and HL(λ)

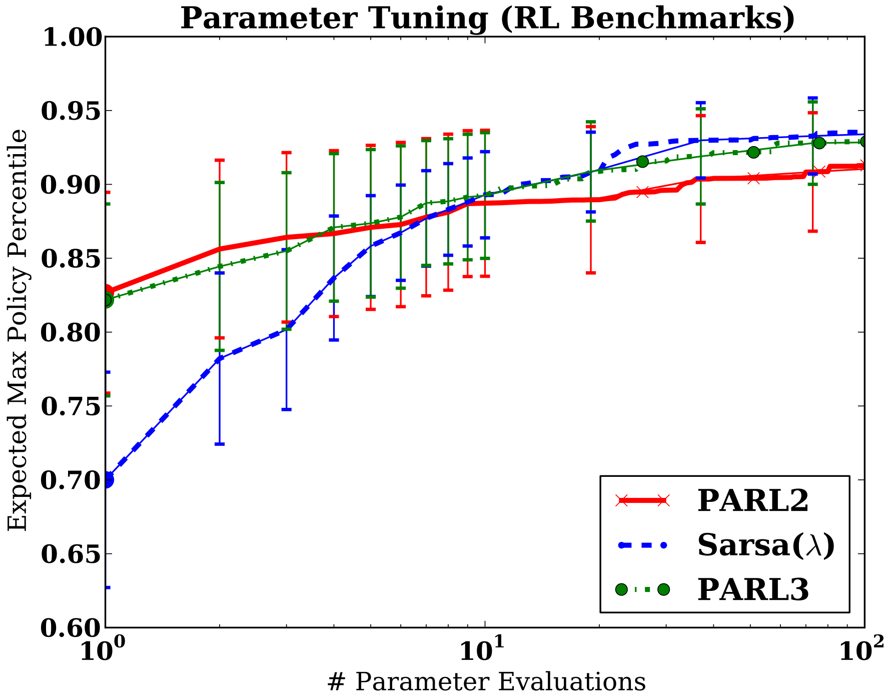
Figure 4.7a shows the performance for HL(λ), PARL2 and Sarsa(λ) after parameter tuning on the finite MDPs in RL Benchmark. We can see that Sarsa(λ) does appear to perform better relative to the adaptive step sizes than in Figure 4.6 where only continuous MDPs are used. However, we were surprised to find that HL(λ), which is an optimal step size derived for finite MDPs and Sarsa(λ) suffers the same performance gap as PARL2. We expected this to happen only for the adaptive algorithms that allow more general forms of function approximation. Note that these differences are well within margins for error and so are not statistically significant. However, these results lead us to believe that the observed differences may be characteristic of adaptive step sizes in general.

4.5.6 Overview

Figure 4.8 shows a comparison over the RL Benchmark set of Sarsa(λ) with the top performing algorithms from each sub-group. Two results become immediately clear. First, there is some advantage to be obtained by including even approximate solutions to the update whitening problem which can be seen by Autostep's higher performance. This is promising as the next chapter focuses on addressing this problem in a principled manner. Second, PARL2 and VES, which both entirely eliminate the step-size parameter, perform at least as well as the highly optimized Sarsa(λ) and the



(a) Performance with optimized parameters



(b) Difficulty and Impact of parameter tuning

Figure 4.5: Passive-aggressive step sizes (PARL) and Sarsa(λ), averaged over the RL Benchmark set.

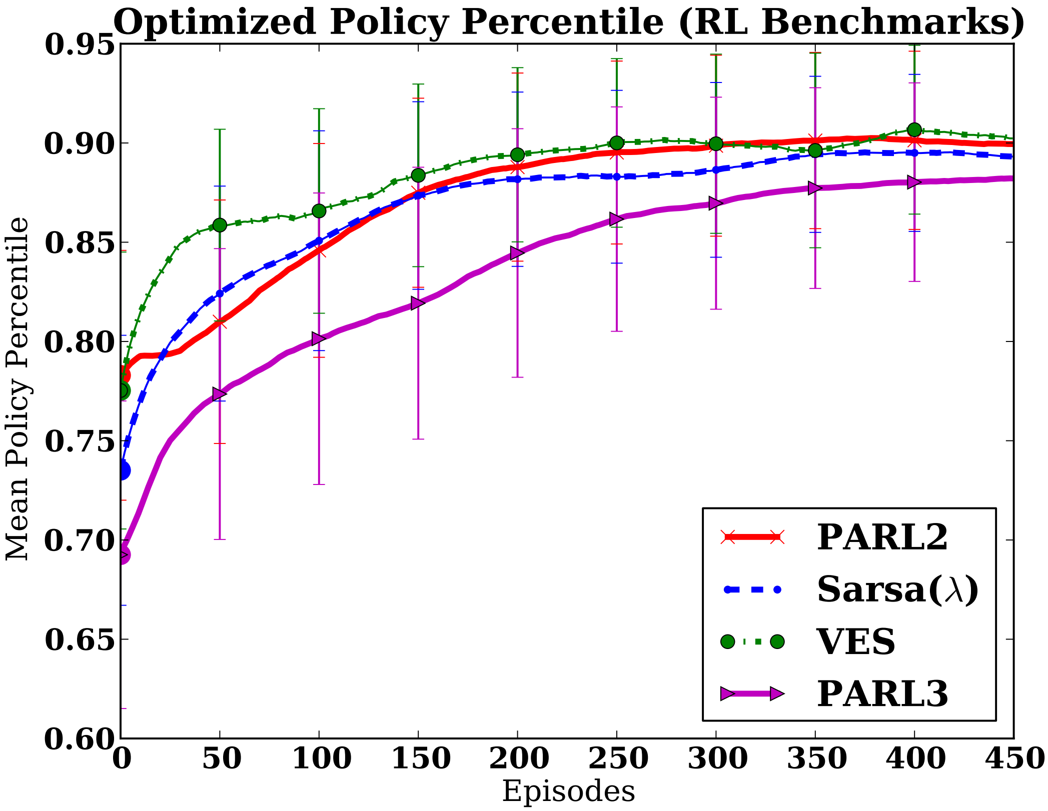
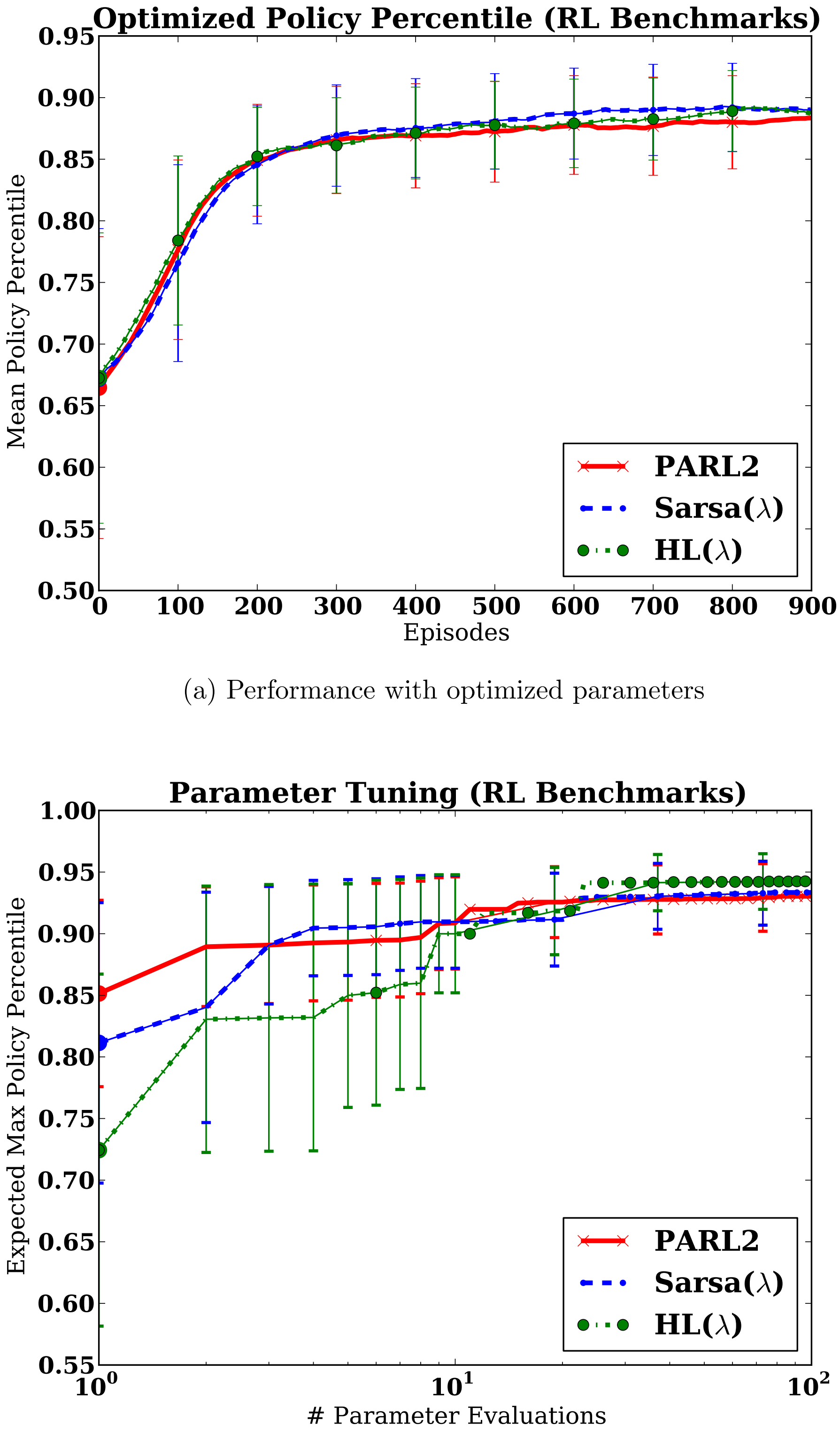


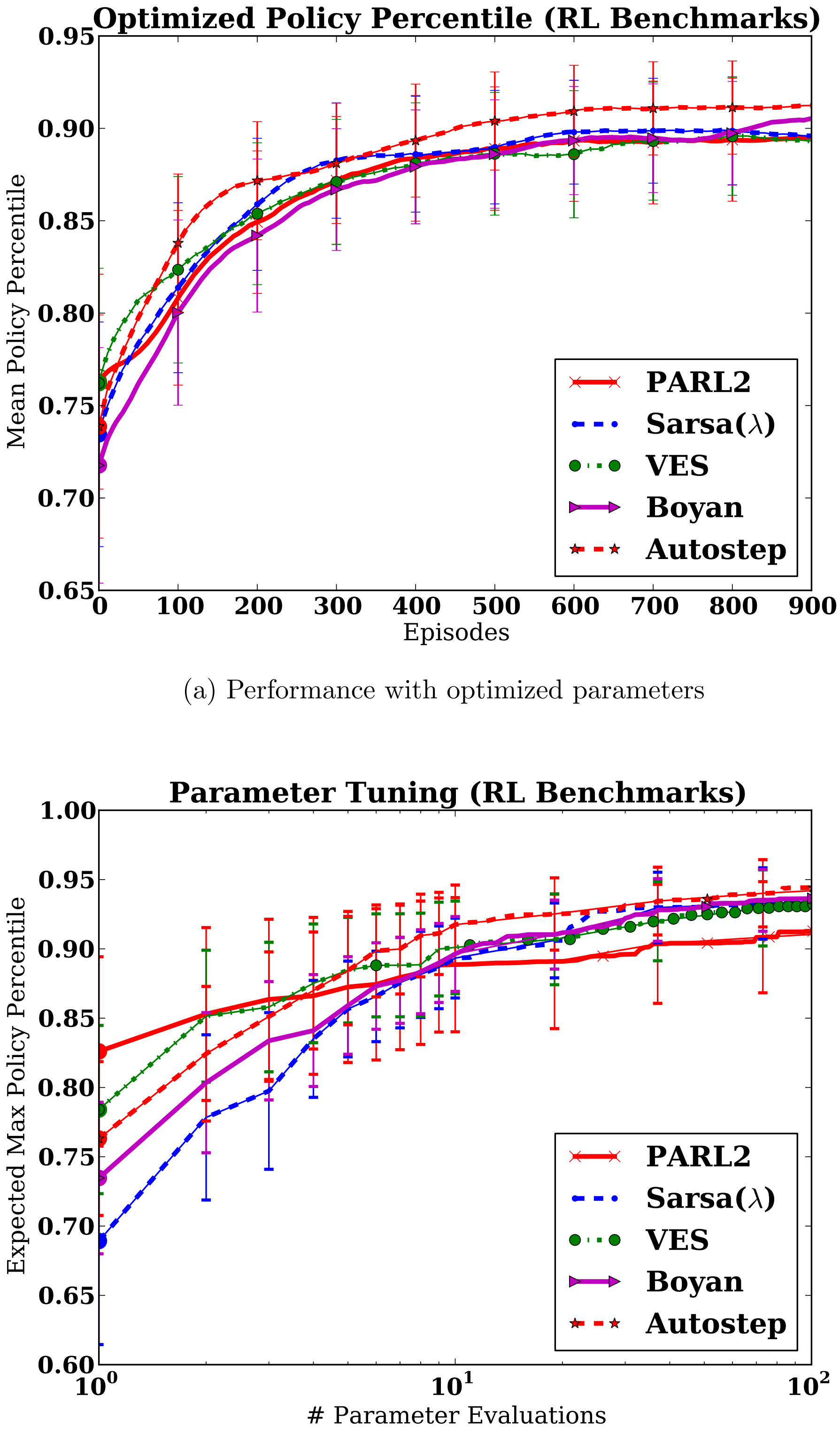
Figure 4.6: Performance with optimized parameters of VES, PARL2, PARL3 and Sarsa(λ).



(a) Performance with optimized parameters

(b) Difficulty and Impact of parameter tuning

Figure 4.7: HL(λ), PARL2 and Sarsa(λ), averaged over the finite MDPs in the RL Benchmark set.



(a) Performance with optimized parameters

(b) Difficulty and Impact of parameter tuning

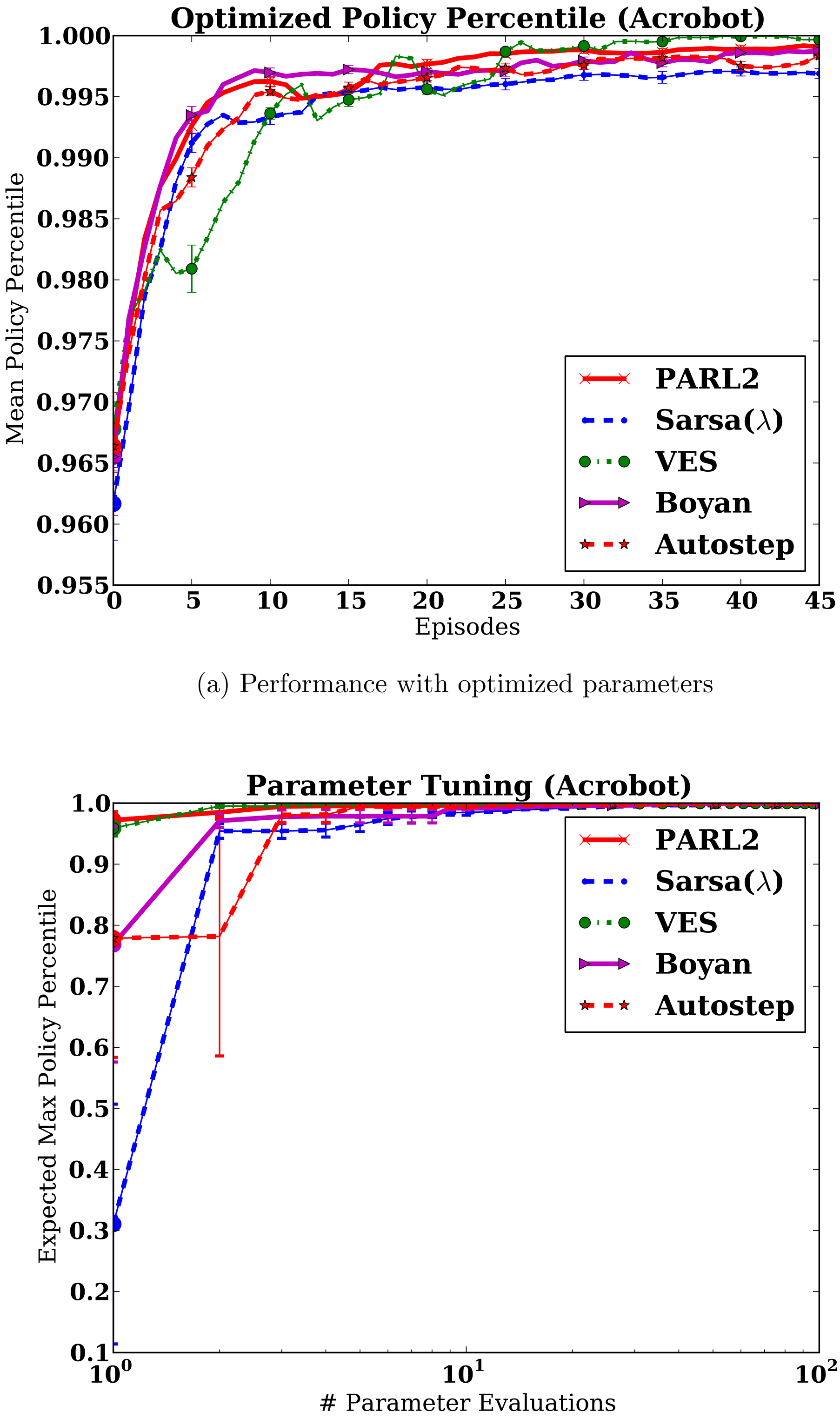
Figure 4.8: Adaptive step-size algorithms with Sarsa(λ), averaged over RL Benchmark set of domains.

parameter tuning difficulties (Figure 4.8b) make it clear that these two algorithms present a much simpler parameter tuning problem.

By considering the performance on individual domains some interesting characteristics of the adaptive algorithms become apparent. Figure 4.9 shows the same set of algorithms compared on the Acrobot domain alone. We observe that in Figure 4.9a the behavior of PARL2 and VES are fundamentally different. PARL2 is very aggressive with its updates early on, whereas VES is initially slower to improve but then rapidly overtakes the other algorithms and finds a policy that performs very near optimal. In Figure 4.9b a concrete instance of the difficulty of parameter tuning shows that, for Acrobot, VES and PARL2 learn near-optimal policies on average with the remaining parameters chosen randomly. A very similar narrative plays out on the HIV Treatment domain, shown in Figure 4.10, and the single-pole Cart Pole Balancing MDP, shown in Figure 4.11. We note in particular for single pole balancing the two algorithms that are free of any tunable step size immediately find a near-optimal policy. Something interesting is happening here. The empirical distribution over discounted returns for this domain (Chapter 3, Figure 3.9a) shows that it is much easier to find near-optimal policies than one would expect. It appears that the two parameter-free step-size algorithms are able to take advantage of this in some way.

4.6 Conclusion

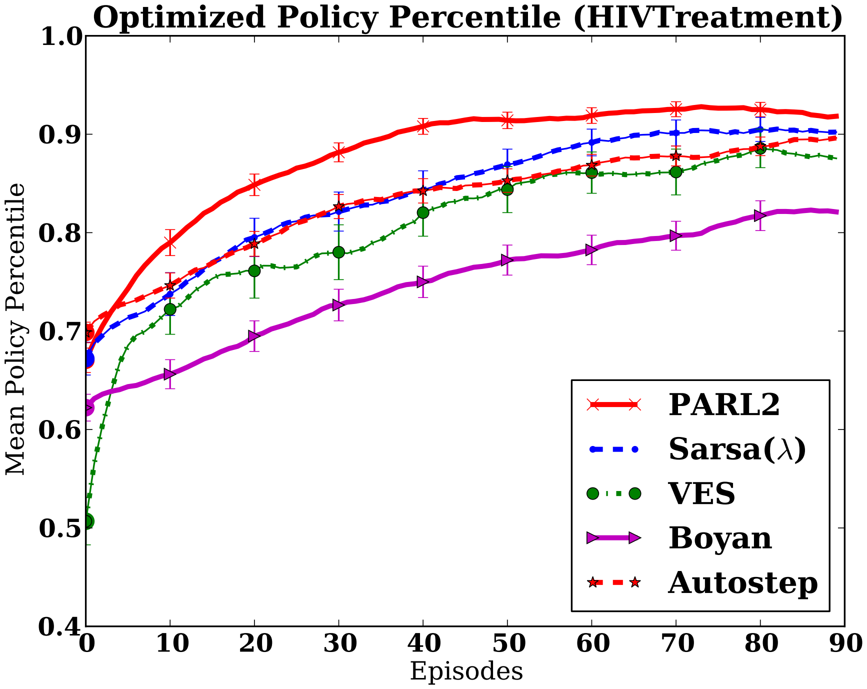
We used three different approaches to derive adaptive step-size algorithms for the Sarsa(λ) algorithm. While SID and NOSID are based upon IDBD, and VES closely related to vSGD, the PARL algorithms are entirely novel. Furthermore, VES can be seen as a novel extension of the vSGD algorithm which eliminates parameters as well as the need to assume a diagonal Hessian. Finally, we presented the results of an empirical study over these newly derived algorithms as well as many existing step-size schedules and adaptive algorithms. This very large study showed that two of our adaptive step-size algorithms (VES and PARL2) perform particularly well and both completely eliminate the need of a tunable step-size parameter. These two algorithms are also entirely different in interesting ways and we used their behavior on individual MDPs to illustrate these differences.



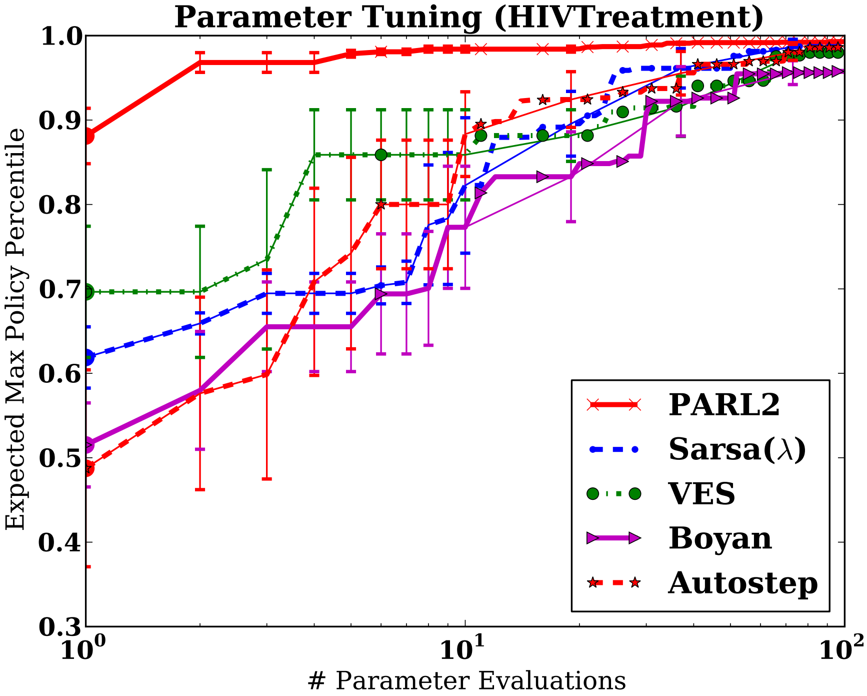
(a) Performance with optimized parameters

(b) Difficulty and Impact of parameter tuning

Figure 4.9: Adaptive step-size algorithms with Sarsa(λ), on Acrobot.

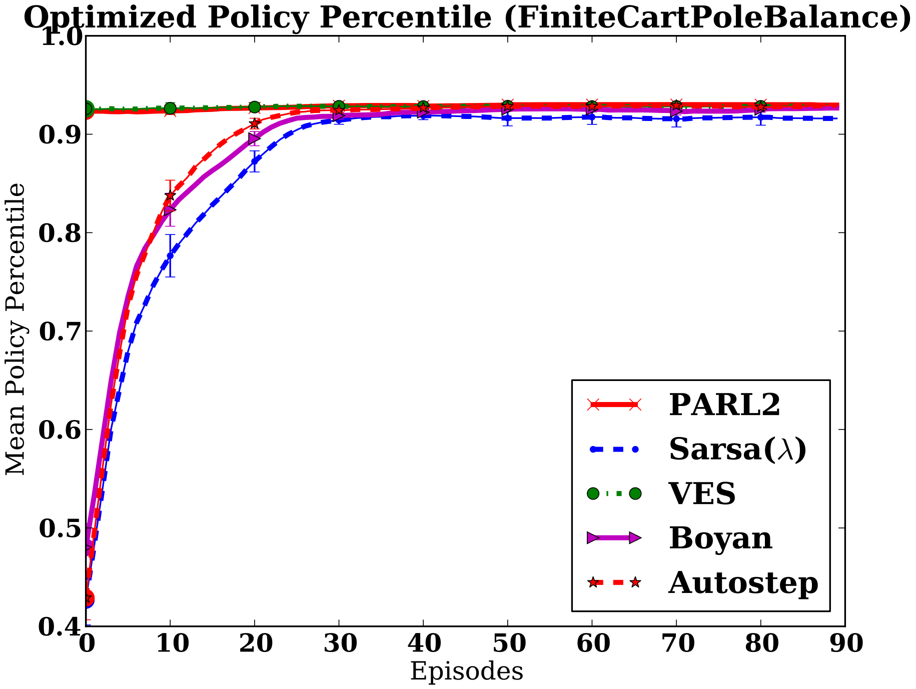


(a) Performance with optimized parameters

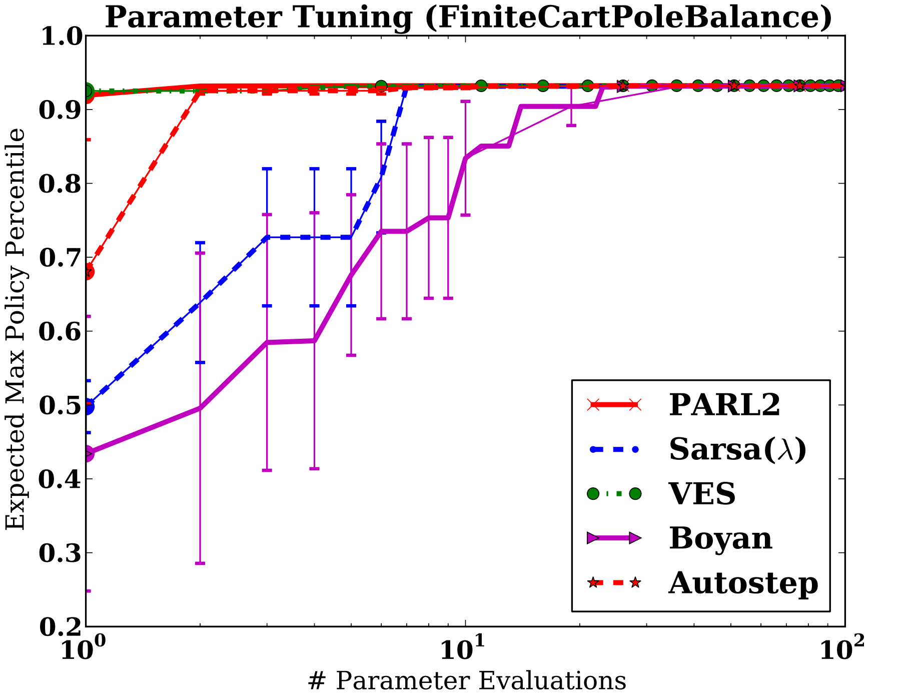


(b) Difficulty and Impact of parameter tuning

Figure 4.10: Adaptive step-size algorithms with Sarsa(λ), on HIV Treatment.



(a) Performance with optimized parameters



(b) Difficulty and Impact of parameter tuning

Figure 4.11: Adaptive step-size algorithms with Sarsa(λ), on Finite Track Cart Pole Balancing.

CHAPTER 5
NATURAL TEMPORAL DIFFERENCE LEARNING

In this chapter we investigate the application of natural gradient descent to RL algorithms based on the Bellman error. This combination is interesting because natural gradient descent is one method of approximately solving the update whitening problem and is invariant to the parameterization of the value function. This invariance property means that natural gradient descent adapts its update directions to correct for poorly conditioned representations. We present and analyze quadratic and linear time natural temporal difference learning algorithms, and prove that they are covariant. Covariance guarantees that, when smooth maps between function approximations exist, the update direction generated by our learning algorithm is invariant with respect to the function approximation representation. That is, the direction of change in function space does not depend on our choice of representation for suitably small step sizes. We conclude with experiments suggesting that the natural algorithms can match or outperform their non-natural counterparts using linear function approximation and drastically improve upon their non-natural counterparts when using non-linear function approximation. Our fundamental contribution is the construction and theoretical analysis of the class of natural temporal difference learning algorithms.

5.1 Introduction

Much recent research has focused on reinforcement learning (RL) problems with continuous actions. For these problems, a significant leap in performance occurred when Kakade [2002] suggested the application of natural gradients [Amari, 1998] to policy gradient algorithms. This suggestion has resulted in many successful policy search algorithms based on natural gradients [Morimura et al., 2005, Peters and Schaal, 2008, Bhatnagar et al., 2009, Degris et al., 2012].

Despite the successful applications of natural gradients to RL in the context of policy search, it has not been applied to Bellman-error based algorithms like residual gradient and Sarsa(λ), which are the *de facto* algorithms for problems with discrete action sets. A common complaint is that these Bellman-error based algorithms learn slowly when using function approximation. Natural gradients are a quasi-Newton approach that is known to speed up gradient descent, and thus the synthesis of natural gradients with TD has the potential to improve upon this drawback of RL. Additionally, we show that the natural TD methods are covariant, which makes them more robust to the choice of representation than ordinary TD methods.

In this chapter we provide simple quadratic-time natural temporal difference learning algorithms, show how the idea of compatible function approximation can be leveraged to achieve linear time complexity, and prove that our algorithms are covariant. We conclude with empirical comparisons between the natural and non-natural algorithms on three canonical domains (mountain car, cart-pole balancing, and acrobot) and one novel challenging domain (playing Tic-tac-toe using handwritten letters as input).

5.2 Residual Gradient

The residual gradient (RG) algorithm is the direct application of stochastic gradient descent to the problem of minimizing the *mean-squared Bellman error* (MSBE) [Baird, 1995]. It is given by the following update equations:

$$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t), \tag{5.1}$$

$$\theta_{t+1} = \theta_t - \alpha_t \delta_t \frac{\partial \delta_t}{\partial \theta}, \tag{5.2}$$

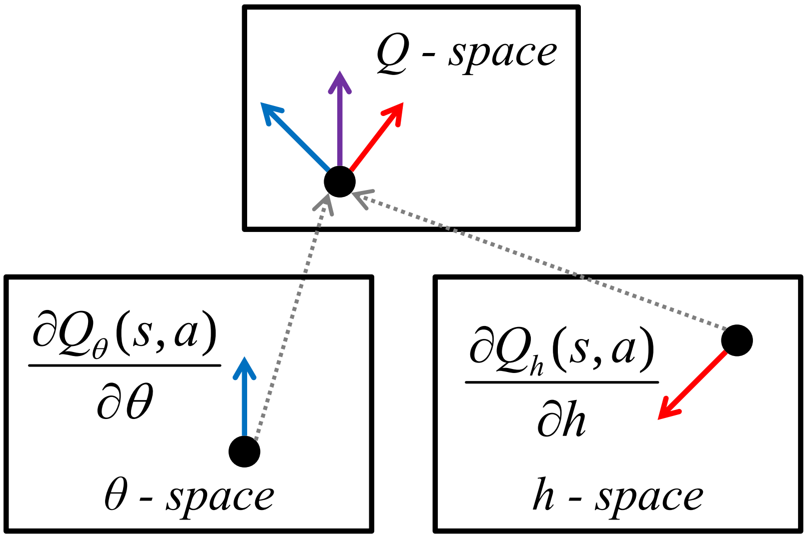
where $Q_{\theta_t} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a function approximator with weight vector θ_t. Residual gradient only follows unbiased estimates of the gradient of the MSBE if it uses double sampling or when the domain has deterministic state transitions [Sutton and Barto, 1998a]. When using residual gradient we will evaluate using standard RL domains with deterministic transitions, so the above formulation of RG is unbiased.

One significant drawback of residual gradient is that it is not covariant. Consider the algorithm at two different levels, as depicted in Figure 5.1. At one level we can consider how it moves through the space of possible Q functions. At another level, we can consider how it moves through two different weight spaces, each corresponding to a different representation of Q. Although these two representations may produce different update directions in weight space, we would expect a good algorithm to result in both representations producing the same update direction in the space of Q functions.[1]

Such an algorithm would be called *covariant*. Because residual gradient is not covariant, the choice of how to represent Q_θ influences the direction that RG moves in the space of Q functions. Other temporal difference (TD) learning algorithms like Sarsa(λ) and TDC [Sutton et al., 2009] are also not covariant. Natural gradients can be viewed as a way to correct the direction of an update to account for a particular parameterization.[2] Although natural gradients do not always result in covariant updates, they frequently do [Bagnell and Schneider, 2003].

[1] For technical correctness, we must assume that both representations can represent the same set of Q functions.

[2] Parameterization refers to the form of the loss function and action-value approximation with respect to weight vectors, and is not to be confused with the tunable parameter space.



Q - space
$\frac{\partial Q_\theta(s,a)}{\partial \theta}$
θ - space
$\frac{\partial Q_h(s,a)}{\partial h}$
h - space

Figure 5.1: Q-space denotes the space of possible Q functions, while θ and h-space denote two different weight spaces. The circles denote different locations in θ and h-space that correspond to the same Q function. The blue and red arrows denote possible directions that a non-covariant algorithm might attempt to change the weight vector, which correspond to different directions in Q-space. The purple arrow denotes the update direction that a covariant algorithm might produce, regardless of the parameterization of Q.

Formally, consider the direction of steepest ascent of a function $L(\theta)$, where $L : \mathbb{R}^n \rightarrow \mathbb{R}$. If we assume that θ resides in Euclidean space, then the gradient, $\nabla L(\theta)$, gives the direction of steepest ascent. However, if we assume that θ resides in a Riemannian space with metric tensor $G(\theta)$, then the direction of steepest ascent is given by $G(\theta)^{-1}\nabla L(\theta)$ [Amari, 1998].

5.3 Natural Residual Gradient

In this section we describe how natural gradient descent can be applied to the residual gradient algorithm. The natural RG update is

$$\theta_{t+1} = \theta_t + \alpha_t G(\theta_t)^{-1} \delta_t g_t, \tag{5.3}$$

where $G(\theta_t)$ is the metric tensor for the parameter space and

$$g_t = \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta} - \gamma \frac{\partial Q_{\theta_t}(s_{t+1}, a_{t+1})}{\partial \theta}.$$

In most RL applications of natural gradients, the metric tensor is used to correct for the parameterization of a probability distribution. In these cases the Fisher information matrix is a natural choice for the metric tensor [Amari and Douglas, 1998a]. However, we are using natural gradients to correct for the parameterization of an action-value function, which is not a distribution. For a related application, Amari [1998] suggests a transformation of a parameterized function to a parameterized probability distribution. Using this transformation, the Fisher information matrix is

$$G(\theta_t) = \mathbb{E}\left[\delta_t^2 g_t g_t^\intercal\right]. \tag{5.4}$$

We will now prove that the class of metric tensors to which Equation 5.4 belongs all result in covariant gradient algorithms. The following theorem and its proof closely follow and extend the foundations laid by Bagnell and Schneider [2003] and later clarified by Peters and Schaal [2008] when proving that the natural policy gradient is covariant. No algorithm can be covariant for all parameterizations. Thus, constraints on the parameterized functions that we consider are required.

Property 5.1. *Functions $g : \Phi \times X \to \mathbb{R}$, and $h : \Theta \times X \to \mathbb{R}$ are two instantaneous loss functions over set X parameterized by $\phi \in \Phi$ and $\theta \in \Theta$ respectively. These correspond to the loss functions $\hat{g}(\phi) = \mathbb{E}_{x \in X}[g(\phi, x)]$ and $\hat{h}(\theta) = \mathbb{E}_{x \in X}[h(\theta, x)]$. For brevity, hereafter, we suppress the x inputs to g and h. Assume that there exists a differentiable function, $\Psi : \Phi \to \Theta$, such that for some $\phi \in \Phi$, we have $g(\phi) = h(\Psi(\phi))$ and the Jacobian of Ψ is full rank.*

Definition 5.1. *Algorithm A is covariant if, for all g, h, Ψ, and ϕ satisfying Property 1,*

$$g(\phi + \Delta\phi) = h(\Psi(\phi) + \Delta\theta), \tag{5.5}$$

where $\phi + \Delta\phi$ and $\Psi(\phi) + \Delta\theta$ are the parameters after an update of algorithm A.

Lemma 5.1. *An algorithm A is covariant for sufficiently small step-sizes if*

$$\Delta\theta = \frac{\partial \Psi(\phi)}{\partial \phi} \Delta\phi. \tag{5.6}$$

Proof. Let $J_{\Psi(\phi)}$ be the Jacobian of $\Psi(\phi)$, i.e., $J_{\Psi(\phi)} = \frac{\partial \Psi(\phi)}{\partial \phi}$. As such, it maps tangent vectors of h to tangent vectors of g, such that

$$\frac{\partial g(\phi)}{\partial \phi} = J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \Psi(\phi)}, \tag{5.7}$$

when $g(\phi) = h(\Psi(\phi))$, as $J_{\Psi(\phi)}$ is a tangent map [Lee, 2003, p. 63].

Taking the first order Taylor expansion of both sides of (5.5), we obtain

$$\begin{array}{ll} h(\Psi(\phi)) & +\frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} \Delta\theta \\ & + O(\|\Delta\theta\|^2) \end{array} = \begin{array}{ll} g(\phi) & +\frac{\partial g(\phi)^\intercal}{\partial \phi} \Delta\phi \\ & +O(\|\Delta\phi\|^2). \end{array}$$

For small step-sizes, $\alpha > 0$, the squared norms become negligible, and because $g(\phi) = h(\Psi(\phi))$, this simplifies to

$$\begin{aligned} \frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} \Delta\theta &= \frac{\partial g(\phi)^\intercal}{\partial \phi} \Delta\phi, \\ &= \left(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \Psi(\phi)} \right)^\intercal \Delta\phi, \\ &= \frac{\partial h(\Psi(\phi))^\intercal}{\partial \Psi(\phi)} J_{\Psi(\phi)}^\intercal \Delta\phi. \end{aligned} \tag{5.8}$$

Notice that (5.8) is satisfied by $\Delta\theta = J_{\Psi(\phi)}^\intercal \Delta\phi$, and thus if this equality holds then A is covariant. □

Theorem 5.1. *The natural gradient update $\Delta\theta = -G_\theta^{-1} \nabla h(\theta)$ is covariant when the metric tensor G_θ is given by*

$$G_\theta = \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial h(\theta)}{\partial \theta} \frac{\partial h(\theta)^\intercal}{\partial \theta} \right]. \tag{5.9}$$

Proof. First, notice that the metric tensor G_ϕ is equivalent to G_θ with $J_{\Psi(\phi)}$ twice as a factor:

$$\begin{aligned}
G_\phi &= \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial g(\phi)}{\partial \phi} \frac{\partial g(\phi)}{\partial \phi}^\intercal \right], \\
&= \mathop{\mathbb{E}}_{x \in X} \left[(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta})(J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta})^\intercal \right], \\
&= \mathop{\mathbb{E}}_{x \in X} \left[J_{\Psi(\phi)} \frac{\partial h(\Psi(\phi))}{\partial \theta} \frac{\partial h(\Psi(\phi))}{\partial \theta}^\intercal J_{\Psi(\phi)}^\intercal \right], \\
&= J_{\Psi(\phi)} \mathop{\mathbb{E}}_{x \in X} \left[\frac{\partial h(\Psi(\phi))}{\partial \theta} \frac{\partial h(\Psi(\phi))}{\partial \theta}^\intercal \right] J_{\Psi(\phi)}^\intercal, \\
&= J_{\Psi(\phi)} G_\theta J_{\Psi(\phi)}^\intercal. \qquad (5.10)
\end{aligned}$$

We show that the right hand side of (5.6) is equal to the left, which, by Lemma 1, implies that the natural gradient update is covariant.

$$\begin{aligned}
J_{\Psi(\phi)}^\intercal \Delta\phi &= J_{\Psi(\phi)}^\intercal \alpha G_\phi^{-1} \nabla g(\phi), \\
&= J_{\Psi(\phi)}^\intercal \alpha G_\phi^{+} \nabla g(\phi), \qquad (5.11) \\
&= \alpha J_{\Psi(\phi)}^\intercal \left(J_{\Psi(\phi)} G_\theta J_{\Psi(\phi)}^\intercal \right)^+ J_{\Psi(\phi)} \nabla h(\Psi(\phi)), \\
&= \alpha J_{\Psi(\phi)}^\intercal (J_{\Psi(\phi)}^\intercal)^+ G_\theta^+ J_{\Psi(\phi)}^+ J_{\Psi(\phi)} \nabla h(\Psi(\phi)).
\end{aligned}$$

Since $J_{\Psi(\phi)}$ is full rank, $J_{\Psi(\phi)}^+$ is a left inverse, and thus

$$\begin{aligned}
J_{\Psi(\phi)}^\intercal \Delta\phi &= \alpha G_\theta^{-1} \nabla h(\Psi(\phi)), \\
&= \Delta\theta. \qquad \square
\end{aligned}$$

Notice that, unlike the proof that the natural actor-critic using LSTD is covariant [Peters and Schaal, 2008], our proof does not assume that $J_{\Psi(\phi)}$ is square. Our proof is therefore more general, since it allows $|\phi| \geq |\theta|$.

5.4 Algorithms

5.4.1 Quadratic Computational Complexity

A straightforward implementation of the natural residual gradient algorithm would maintain an estimate of $G(\theta)$ and compute $G(\theta)^{-1}$ at each time step. Due to the matrix inversion, this naïve algorithm has per-time-step computational complexity $O(|\theta|^3)$, where we ignore the complexity of differentiating Q_θ. This can be improved to $O(|\theta|^2)$ using the Sherman-Morrison formula to maintain an estimate of $G(\theta_t)^{-1}$ directly. The resulting quadratic time natural algorithm is given by Algorithm 18, where $\{\alpha_t\}$ is a step size schedule satisfying $\sum_{t=0}^\infty \alpha_t = \infty$ and $\sum_{t=0}^\infty \alpha_t^2 < \infty$.

Algorithm 18 Natural Residual Gradient

```
  Initialize $G_0^{-1} = I$, $\theta_0 = 0$
$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
$g_t = \left( \frac{\partial Q_{\theta_t}(s_t,a_t)}{\partial \theta} - \gamma \frac{\partial Q_{\theta_t}(s_{t+1},a_{t+1})}{\partial \theta} \right)$
$G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1+\delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$
$\theta_{t+1} = \theta_t + \alpha_t \delta_t G_t^{-1} g_t$
```

5.4.2 Linear Computational Complexity

To achieve linear computational complexity, we leverage the idea of compatible function approximation.[3] We begin by estimating the TD-error, δ_t, with a linear function approximator $w^\intercal(\delta_t g_t)$, where w are the weights of the linear function approximator and $\delta_t g_t$ are the compatible features. Specifically, we search for a w that is a local minimum of the loss function L:

$$L(w) = \mathbb{E}\left[(1 - \delta_t w^\intercal g_t)^2\right]. \tag{5.12}$$

[3] The compatible features that we present are compatible with Q_θ, whereas the compatible features originally defined by Sutton et al. [2000] are compatible with a parameterized policy. Although related, these two types of compatible features are not the same.

At a local minimum of L, $\partial L(w)/\partial w = 0$, so

$$\mathbb{E}\left[(1-\delta_t w^\intercal g_t)\,\delta_t g_t\right] =0, \tag{5.13}$$

$$\implies \mathbb{E}\left[\delta_t g_t\right] =\mathrm{E}\left[\delta_t^2 g_t g_t^\intercal\right] w. \tag{5.14}$$

Notice that the left hand side of Eq. 5.14 is the expected update to θ_t in the non-natural algorithms. We can therefore write the expected update to θ_t as

$$\theta_{t+1} = \theta_t + \alpha_t \mathbb{E}\left[\delta_t g_t\right] = \theta_t + \alpha_t \mathbb{E}\left[\delta_t^2 g_t g_t^\intercal\right] w. \tag{5.15}$$

Therefore the expected natural residual gradient update is

$$\theta_{t+1} =\theta_t + \alpha_t G(\theta)^{-1}\mathbb{E}\left[\delta_t g_t\right], \tag{5.16}$$

$$=\theta_t + \alpha_t w. \tag{5.17}$$

The challenge remains that locally optimal w must be attained. For this we propose a two-timescale approach identical to that of Bhatnagar et al. [2009]. That is, we perform stochastic gradient descent on $L(w)$ using a step size schedule $\{\beta_t\}$ that decays faster than the step size schedule $\{\alpha_t\}$ for updates to θ_t. The resulting linear-complexity two-timescale natural algorithm is given by Algorithm 19.

Algorithm 19 Natural Linear-Time Residual Gradient

```
Initialize $w_0 = 0$, $\theta_0 = 0$
$\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
$g_t = \frac{\partial Q_{\theta_t}(s_t,a_t)}{\partial \theta} - \gamma \frac{\partial Q_{\theta_t}(s_{t+1},a_{t+1})}{\partial \theta}$
$w_{t+1} = w_t + \beta_t \left(1 - \delta_t w_t^\intercal g_t\right) \delta_t g_t$
$\theta_{t+1} = \theta_t + \alpha_t w_{t+1}$
```

The convergence properties of these two-timescale algorithms have been well studied and have been shown to converge under appropriate assumptions [Bhatnagar et al., 2009, Kushner and Yin, 2003]. To summarize, with certain smoothness assumptions, if

$$\sum_{t=0}^{\infty} \alpha_t = \sum_{t=0}^{\infty} \beta_t = \infty;\ \sum_{t=0}^{\infty} \alpha_t^2, \sum_{t=0}^{\infty} \beta_t^2 < \infty;\ \beta_t = o(\alpha_t),$$

then, since $\beta_t \to 0$ faster than α_t, θ_t converges as though it was following the true expected natural gradient. As a result, the linear complexity algorithms maintain the convergence guarantees of their non-natural counterparts.

Unfortunately, unlike compatible function approximation for natural policy gradient algorithms [Bhatnagar et al., 2009], it is not clear how a useful baseline could be added to the stochastic gradient descent updates of w. The baseline, b, would have to satisfy $\mathbb{E}\left[b\delta_t g_t\right] = 0$, which is not even satisfied by a constant non-zero b.

5.4.3 Extensions

The metric tensor that we derived for RG can be applied to other similar algorithms. For example, Sarsa(λ) is not a gradient method, however in many ways it is similar to residual gradient. We therefore propose the use of $G(\theta)$, derived for RG, with Sarsa(λ). Although not as principled as its use with RG, in both cases it corrects for the curvature of the squared Bellman error and the parameterization of Q. This straightforward extension gives us the algorithm for Natural Sarsa(λ) (Algorithm 20), and a linear time Natural Sarsa(λ) algorithm can be defined similar to Algorithm 19.

Algorithm 20 Natural Sarsa(λ)

```
  Initialize $G_0^{-1} = I$, $e_0 = 0$, $\theta_0 = 0$
  $\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
  $g_t = \frac{\partial Q_{\theta_t}(s_t, a_t)}{\partial \theta}$
  $e_t = \gamma \lambda e_{t-1} + g_t$
  $G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1 + \delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$
  $\theta_{t+1} = \theta_t + \alpha_t \delta_t G_t^{-1} e_t$
```

Another temporal difference learning algorithm which is closely related to residual gradient is the TDC algorithm [Sutton et al., 2009]. TDC is a linear time gradient descent algorithm for TD-learning with linear function approximation, and supports off-policy learning.

The TDC algorithm is given by,

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \phi_t - \alpha_t \gamma \phi_{t+1}(\phi_t^\intercal w_t), \tag{5.18}$$

$$w_{t+1} = w_t + \beta_t(\delta_t - \phi_t^\intercal w_t)\phi_t, \tag{5.19}$$

where $\phi_t = \frac{\partial Q_{\theta_t}(s_t,a_t)}{\partial \theta}$ are basis functions of the linear function approximation. TDC minimizes the mean squared projected Bellman error (MSPBE) using a projection operator that minimizes the value function approximation error. With a different projection operator the same derivation results in the standard residual gradient algorithm. Applying the TD metric tensor we get Natural TDC (Algorithm 21).

Algorithm 21 Natural TDC

```
  Initialize $G_0^{-1} = I$, $\theta_0 = 0$, $w_0 = 0$
  $\delta_t = r_t + \gamma Q_{\theta_t}(s_{t+1}, a_{t+1}) - Q_{\theta_t}(s_t, a_t)$
  $g_t = \phi_t - \gamma \phi_{t+1}$
  $G_t^{-1} = G_{t-1}^{-1} - \frac{\delta_t^2 G_{t-1}^{-1} g_t g_t^\intercal G_{t-1}^{-1}}{1+\delta_t^2 g_t^\intercal G_{t-1}^{-1} g_t}$
  $\theta_{t+1} = \theta_t + \alpha_t G_t^{-1}(\delta_t \phi_t - \gamma \phi_{t+1}(\phi_t^\intercal w_t))$
  $w_{t+1} = w_t + \beta_t(\delta_t - \phi_t^\intercal w_t)\phi_t$
```

5.5 Experimental Results

Our goal with these experiments is to show that natural TD methods improve upon their non-natural counterparts, not to promote one TD method over another. We focus our experiments on comparing the quadratic and linear time natural variants of temporal different learning algorithms with the original TD algorithms they build upon. To evaluate the performance of natural residual gradient and natural

Sarsa(λ), we performed experiments on two canonical domains: mountain car and cart-pole balancing, as well as one new challenging domain that we call visual Tic-tac-toe. We used an ϵ-greedy policy for all TD-learning algorithms. TDC is not a control algorithm, and thus to evaluate the performance of natural TDC we generate experience from a fixed policy in the acrobot domain and measure the mean squared error (MSE) of the learned value function compared with Monte Carlo rollouts of the fixed policy.

For mountain car, cart-pole balancing, and acrobot we used linear function approximation with a third-order Fourier basis [Konidaris et al., 2012]. On visual Tic-tac-toe we used a fully-connected feed-forward artificial neural network with one hidden layer of 20 nodes. This allows us to show the benefits of natural gradients when the value function parameterization is non-linear and more complex. We optimized the algorithm parameters for all experiments using a randomized search as suggested by Bergstra and Bengio [2012]. We selected the parameters that resulted in the largest mean discounted return over 20 episodes for mountain car, 50 episodes for cart-pole balancing, and $100,000$ episodes for visual tic-tac-toe. Each parameter set was tested 10 times and the performance averaged.

For mountain car and cart pole each algorithm's performance is an average over 50 and 30 trials respectively, with standard deviations shown in the shaded regions. For visual tic-tac-toe and acrobot, algorithm performance is averaged over 10 trials, again with standard deviations shown by the shaded regions. For the Sarsa(λ) experiments we include results for Natural Actor-Critic [Peters and Schaal, 2008], to provide a comparison with another approach to applying natural gradients to reinforcement learning. However, for these experiments we do not include the standard deviations because they make the figures much harder to read. We used a soft-max policy with Natural Actor-Critic (NAC).

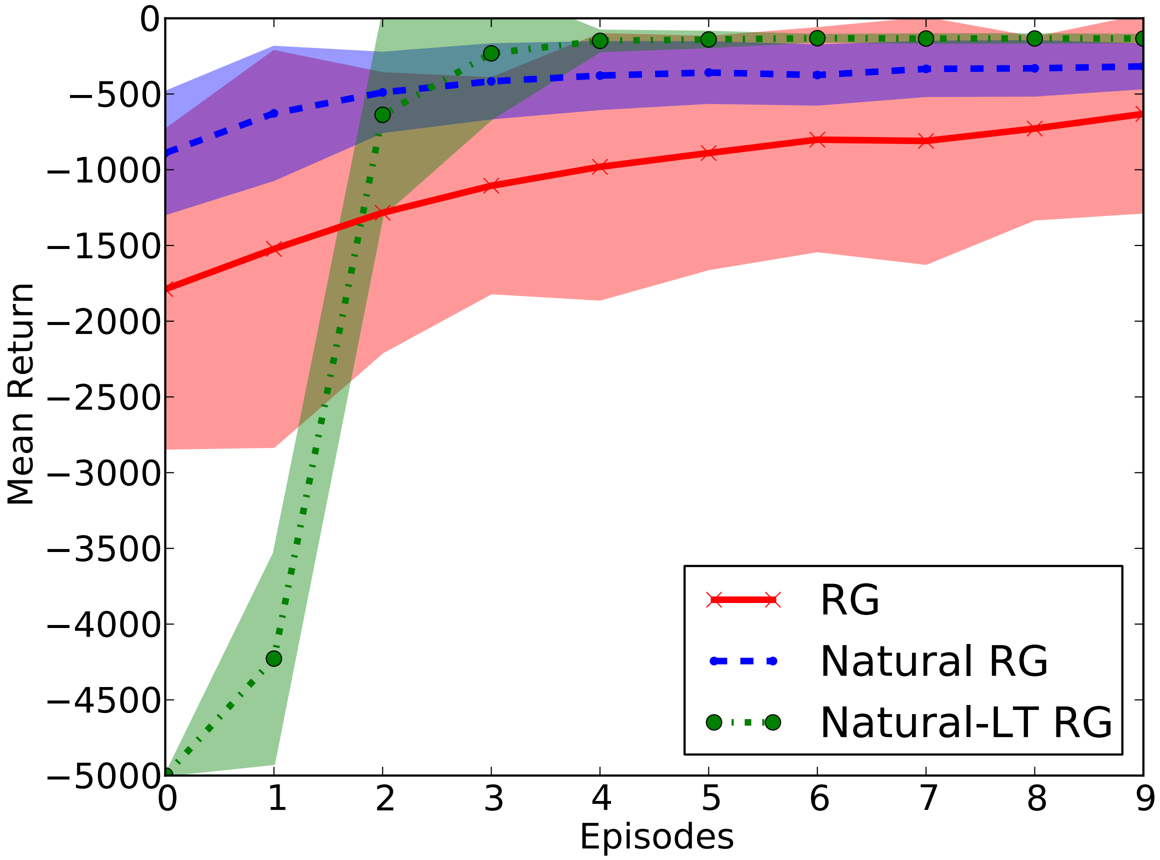


Figure 5.2: Mountain Car (Residual Gradient)

5.5.1 Mountain Car

Mountain car is a simple simulation of an underpowered car stuck in a valley; full details of the domain can be found in the work of Sutton and Barto [1998a]. Figures 5.2 and 5.3 give the results for each algorithm on mountain car. The linear time natural residual gradient and Sarsa(λ) algorithms take longer to learn good policies than the quadratic time natural algorithms. One reason for the slower initial learning of the linear algorithms is that they must first build up an estimate of the w vector before updates to the action-value function weights become meaningful. Out of all the algorithms we found that the quadratic time Natural Sarsa(λ) algorithm performed the best in mountain car, reaching the best policy after just two episodes.

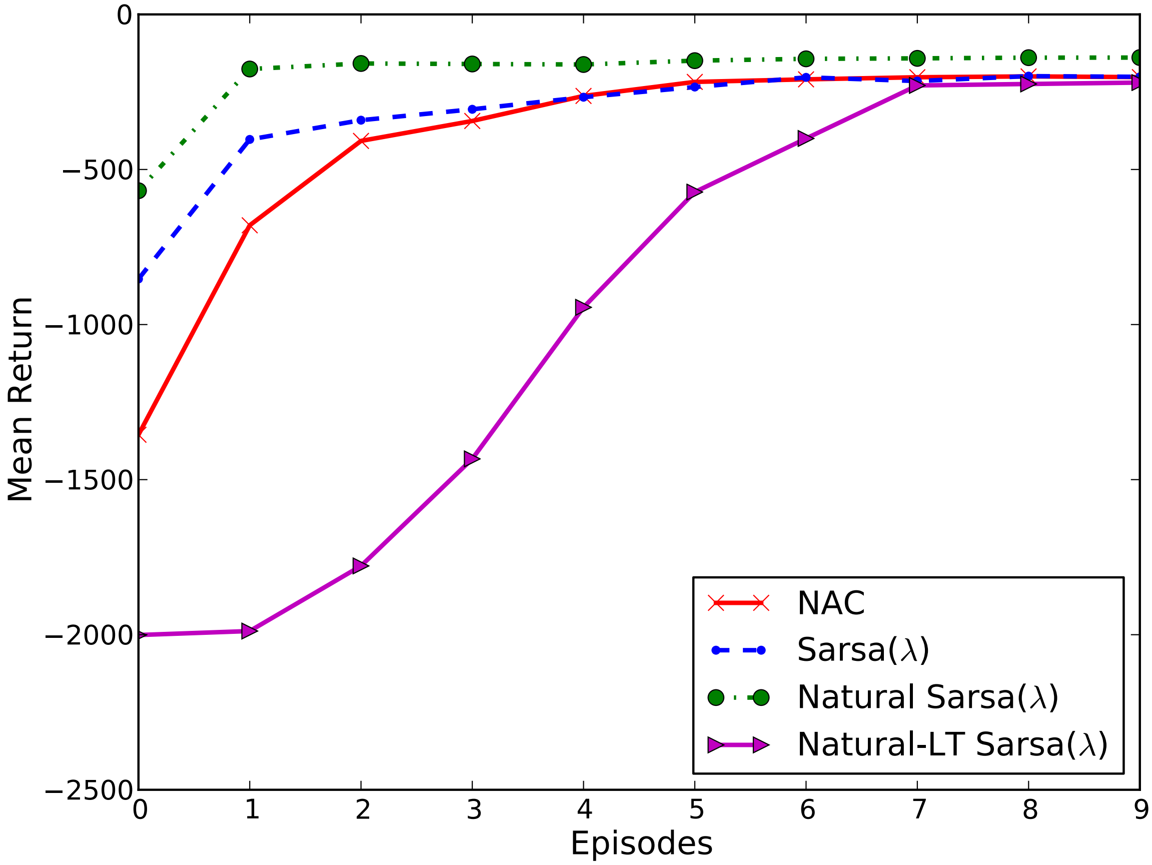


Figure 5.3: Mountain Car (Sarsa(λ))

5.5.2 Cart Pole Balancing

Cart pole balancing simulates a cart on a short one dimensional track with a pole attached with a rotational hinge, and is also referred to as the inverted pendulum problem. There are many varieties of the cart pole balancing domain, and we refer the reader to Barto et al. [1983] for complete details. Figures 5.4 and 5.5 give the results for each algorithm on cart pole balancing. In the cart pole balancing domain the two quadratic algorithms, Natural Sarsa(λ) and Natural RG perform the best. Again, the linear algorithm, takes a slower start as it builds up an estimate of w, but converges well above the non-natural algorithms and very close to the quadratic ones. Natural Sarsa(λ) reaches a near optimal policy within the first couple of episodes, and

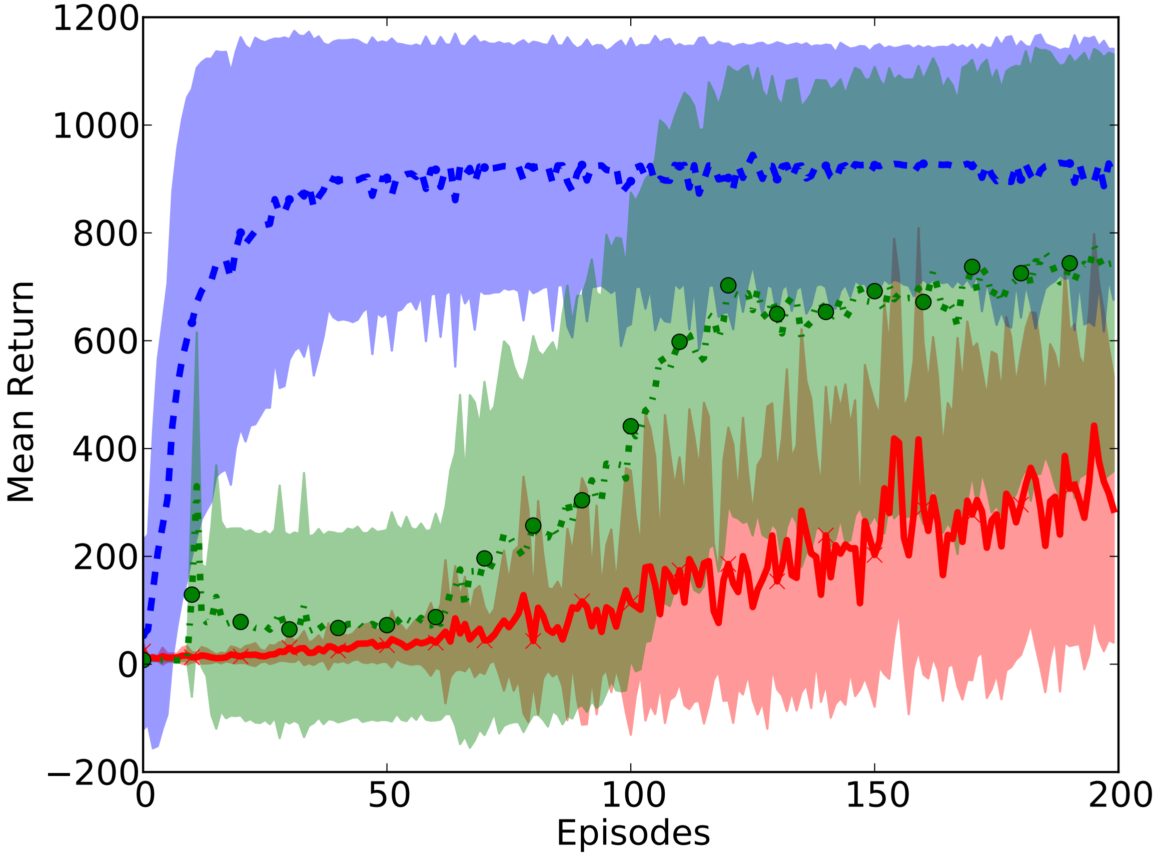


Figure 5.4: Cart Pole (Residual Gradient). Same legend as Figure 5.2

compares favorably with the heavily optimized Sarsa(λ), which does not even reach the same level of performance after 100 episodes.

5.5.3 Visual Tic-Tac-Toe

Visual Tic-Tac-Toe is a novel challenging decision problem in which the agent plays Tic-tac-toe (Noughts and crosses) against an opponent that makes random legal moves. The game board is a 3×3 grid of handwritten letters (X, O, and B for blank) from the UCI Letter Recognition Data Set [Slate, 1991], examples of which are shown in Figure 5.8. At every step of the episode, each letter of the game board is drawn randomly with replacement from the set of available handwritten letters (787 X's, 753 O's, and 766 B's). Thus, it is easily possible for the agent to never

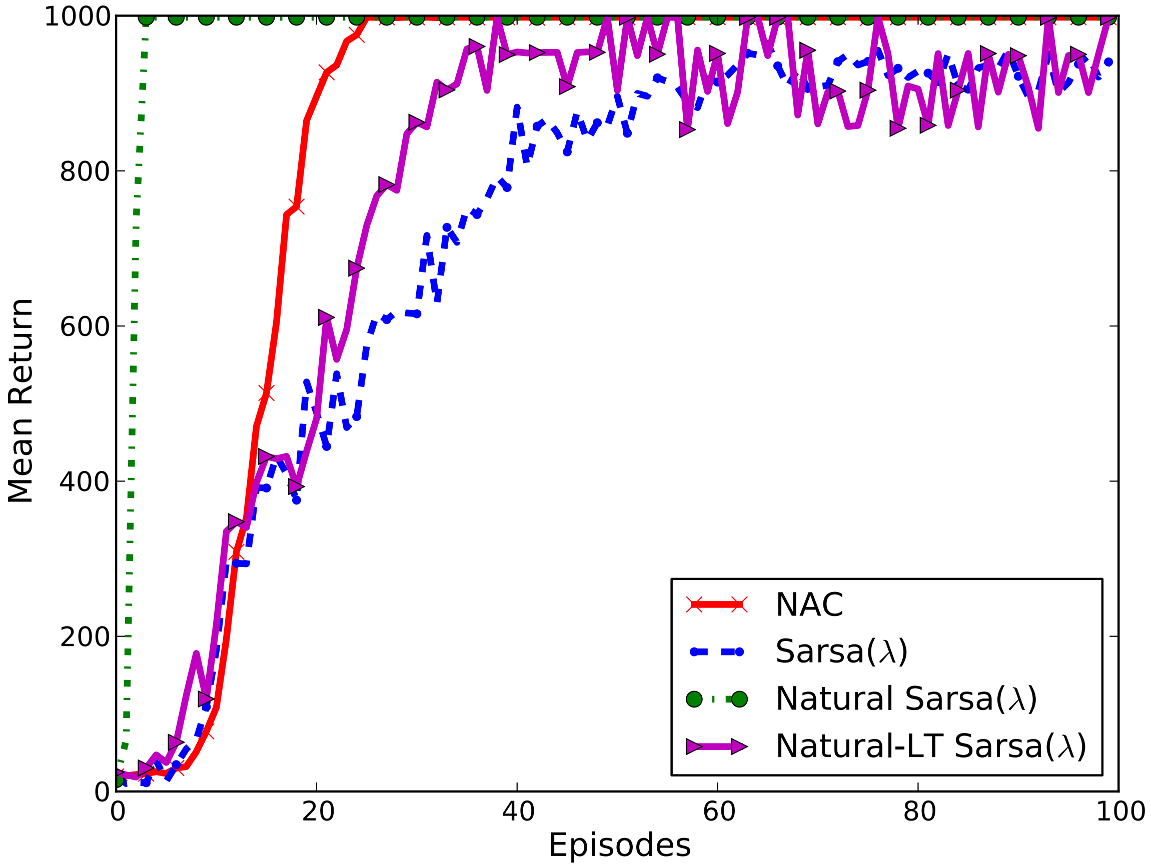


Figure 5.5: Cart Pole (Sarsa(λ))

see the same handwritten "X", "O", or "B" letter in a given episode. The agent's state features are the 16 integer valued attributes for each of the letters on the board. Details of the data set and the attributes can be found in the UCI repository.

There are nine possible actions available to the agent, but attempting to play on a non-blank square is considered an illegal move and results in the agent losing its turn. This is particularly challenging because blank squares are marked by a "B", making recognizing legal moves challenging in and of itself. The opponent only plays legal moves, but chooses randomly among them. The reward for winning is 100, -100 for losing, and 0 otherwise.

Figure 5.6 gives the results comparing Natural-LT Sarsa and Sarsa(λ) on the visual Tic-tac-toe domain using the artificial neural network described previously.

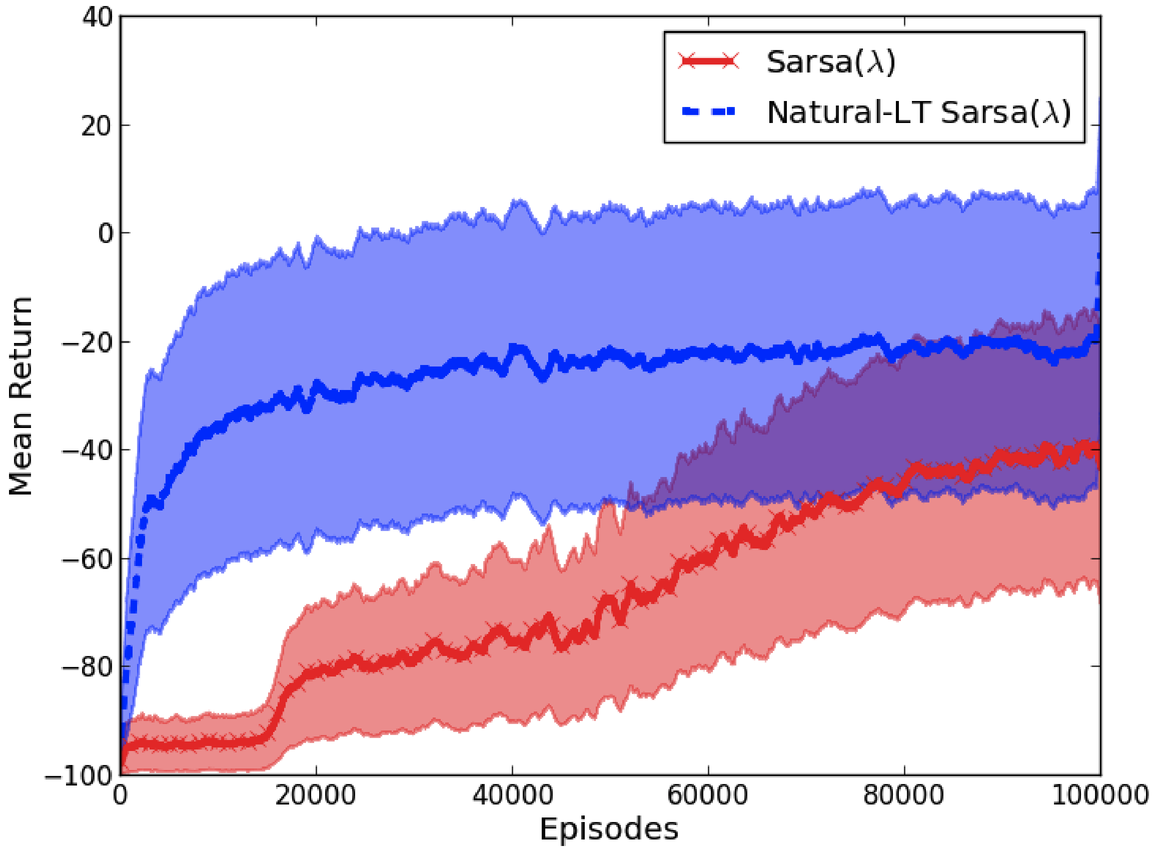


Figure 5.6: Visual Tic-Tac-Toe Experiments

These results show linear natural Sarsa(λ) in a setting where it is able to account for the shape of a more complex value function parameterization, and thus confer greater improvement in convergence speed over non-natural algorithms. We do not compare quadratic time algorithms due to computational limits.

5.5.4 Acrobot

Acrobot is another commonly studied RL task in which the agent controls a two-link under actuated robot by applying torque to the lower joint with the goal of raising the top of the lower link above a certain point. See Sutton and Barto [1998a] for a full specification of the domain and its equations of motion. To evaluate the off-policy Natural TDC algorithm we first generated a fixed policy by online training

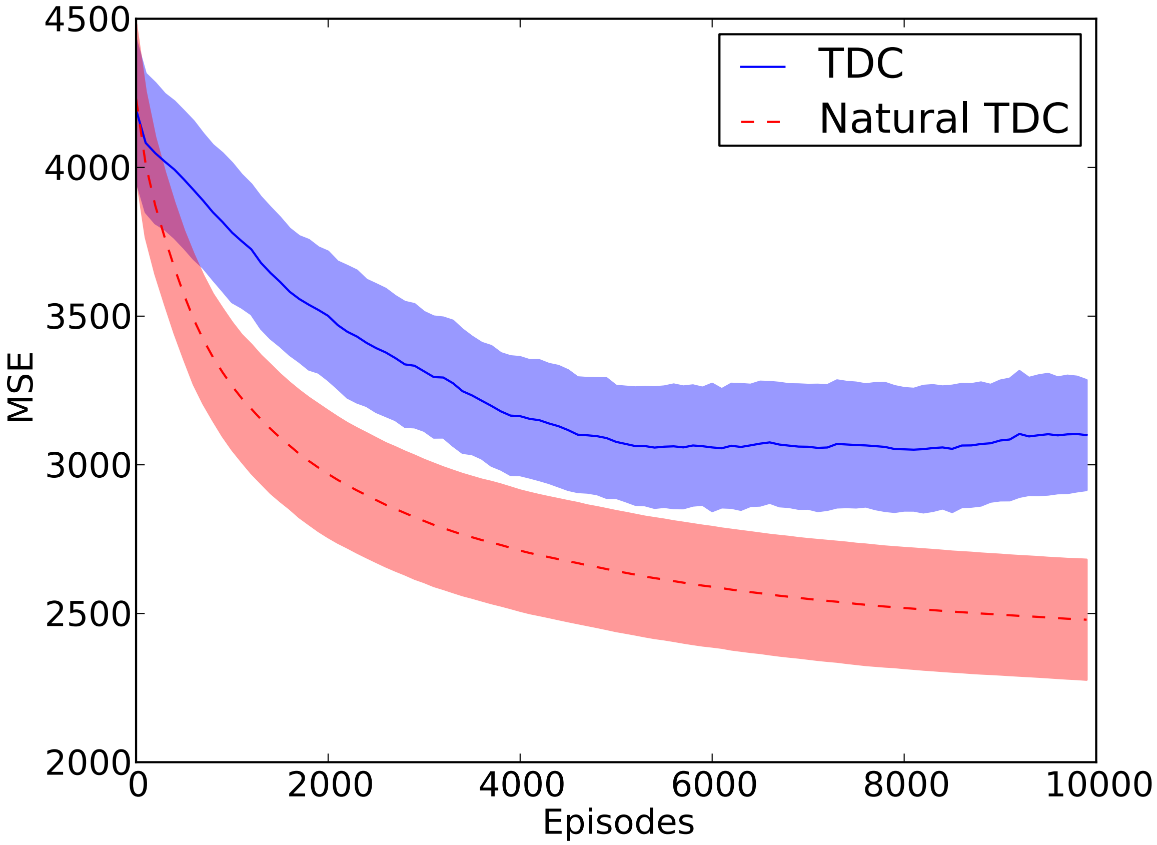


Figure 5.7: Acrobot Experiments (TDC)

of a hand tuned Sarsa(λ) agent for 200 episodes. We then trained TDC and Natural TDC for 10000 episodes in acrobot following the previously learned fixed policy. We evaluated an algorithm's learned value function every 100 episodes by sampling states and actions randomly and computing the true expected undiscounted return using Monte Carlo rollouts following the fixed policy. Figure 5.7 shows the MSE between the learned values and the true expected return.

Natural TDC clearly out performs TDC, and in this experiment converged to much lower MSE. Additionally, we found TDC to be sensitive to the step-sizes used, and saw that Natural TDC was much less sensitive to these parameters. These results show that the benefits of natural temporal difference learning, already observed in the context of control learning, extend to TD-learning for value function estimation as well.

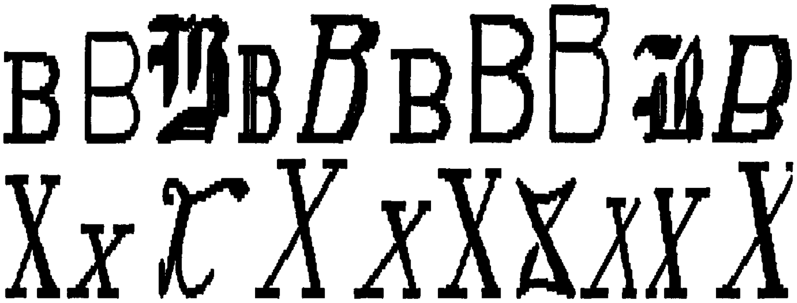


Figure 5.8: Visual Tic-Tac-Toe example letters

5.6 Discussion

Natural temporal difference learning provides one approach for solving the update whitening problem discussed in Chapter 4. The natural gradient approach to this problem is motivated both theoretically and empirically. From the theoretical perspective natural gradients have been shown to minimize the probability of increasing the generalization error, and in Theorem 5.1 we prove that natural TD updates are covariant. Empirically, natural gradients have been greatly successful at improving performance of supervised learning with neural networks and RL policy gradient algorithms. Our derivation and study of natural gradients for action-value TD algorithms such as RG and Sarsa(λ) show greatly improved performance over the non-natural versions of the same algorithms.

The work in Chapter 4 suggests that there may be deeper connections between our explicitly formulated natural TD algorithm and algorithms such as IDBD and Autostep which conflate the problems of adaptive scalar step-size and update whitening. We hinted at this connection in Chapter 2, but can now make the connection precise.

We begin with the linear time algorithm for natural TD, and abstract away which update direction is being used. Let $\triangle w_t$ be the non-natural update direction computed by either RG or Sarsa(λ). The linear time natural TD algorithm computes the *expected natural gradient* with:

$$w_{t+1} = w_t + \beta_t \left(1 - w_t^\intercal \triangle w_t\right) \triangle w_t.$$

The algorithms we presented in this chapter then update the action-values in the direction of the expected natural gradient. Compare this with the update equation used by IDBD, Autostep, and generally any of the SGD adaptive step-size methods for $h_t \approx \frac{\partial \theta_t}{\partial \alpha}$. If we take $\triangle w_t = \delta_t x_t$, then the general form of this update is:

$$h_{t+1} = (I - \alpha x_t x_t^\intercal) h_t + \alpha \delta_t x_t.$$

If a better approximate to the Hessian at time t is available then $x_t x_t^\intercal$ may be replaced to improve the estimate. However, we can see that if the expected natural gradient update is written similarly we get:

$$w_{t+1} = (I - \beta_t \triangle w_t \triangle w_t) w_t + \beta_t \triangle w_t.$$

The difference is illustrative of an interesting approximation. If we assume independence between δ_t and the vector x_t the connection is made exact:

$$\begin{aligned}
\mathbb{E}[H_t] &= \mathbb{E}[\triangle w_t], \\
&= \mathbb{E}[\delta_t^2 x_t x_t^\intercal], \\
&= \mathbb{E}[\delta_t^2] \mathbb{E}[x_t x_t^\intercal], \\
\implies h_t &\propto w_t.
\end{aligned}$$

When this assumption holds the approximation will lower the variance of our estimation of the expected natural gradient. We will complete this analysis by examining how the adaptive step-size algorithms use the expected natural gradient once it is computed. In the case of SID and NOSID, the step-size α_t is moved in the direction of $\triangle w_t^\intercal h_t$. This is reminiscent of the adaptive step-size algorithms discussed in Chapter 2 which increase the step-size when the update direction is consistent and decrease it when it changes signs. In this case, our adaptive step-size algorithms increase the step-size when the update direction is approximately in the direction of the expected natural gradient, and decrease it when the directions are opposed. Similarly, the vector value adaptive step-size algorithms IDBD and Autostep move the step-size on dimension i in the direction of $\triangle w_{t,i} h_{t,i}$.

From this analysis we conclude that the vector value adaptive step-size algorithms (IDBD and Autostep) are approximately solving the update whitening problem as well as the scalar adaptive step-size problem. Additionally, the generally strong performance of Autostep, after parameter tuning, suggests that the approximation of assuming independent Bellman errors may work well with natural TD in general.

5.7 Conclusion

We have presented the natural residual gradient algorithm and proved that it is covariant. We suggested that the temporal difference learning metric tensor, derived for natural residual gradient, can be used to create other natural temporal difference learning algorithms like natural Sarsa(λ) and natural TDC. The resulting algorithms begin with the identity matrix as their estimate of the (inverse) metric tensor. This means that before an estimate of the (inverse) metric tensor has been formed, they still provide meaningful updates—they follow estimates of the non-natural gradient.

We showed how the concept of compatible function approximation can be leveraged to create linear-time natural residual gradient and natural Sarsa(λ) algorithms. However, unlike the quadratic-time variants, these linear-time variants do not provide meaningful updates until the natural gradient has been estimated. As a result, learning is initially slower using the linear-time algorithms.

In our empirical studies, the natural variants of all three algorithms outperformed their non-natural counterparts on all three domains. Additionally, the quadratic-time variants learn faster initially, as expected. Lastly, we showed empirically that the benefits of natural gradients are amplified when using non-linear function approximation.

CHAPTER 6
CONCLUSION

In seeking to develop adaptive step-size algorithms for RL this dissertation has brought up challenging topics not frequently discussed in the field. Inherent in the goal of creating an adaptive step-size algorithm is a bias against designing algorithms which require problem specific customization. Despite this common preference the standard practices surrounding empirical methods in RL have been precisely those which hamper attempts to fulfill it. With the concept of ecological optimality in mind we proposed an improved set of empirical methods for conducting and presenting RL research and showed how they may be used to evaluate and compare RL algorithms.

We introduced a transformation of the performance measure from discounted return to *policy percentile*. While the discounted return is difficult to interpret and has a scale that is domain dependent, the policy percentile is easily interpreted, has a scale independent of the domain, and measures the benefits of performing RL as opposed to randomly guessing policies. We proposed a procedure for running parameter tuning and reporting the difficulty and impact of the the parameter tuning process for a given RL algorithm. We completed the set of empirical methods with a discussion of methods for hypothesis testing in RL and proposing a broad set of RL domains (RL Benchmark) to be used for experiments.

Now equipped with the tools for inquiry we turned toward the adaptive step-size problem within the context of RL. Based upon a separability assumption we derived new adaptive scalar step-sizes for RL and used our new empirical methods to evaluate their performance. In particular we developed three new parameter-free adaptive step-size algorithms: VES, PARL2, and PARL3. Our large experimental study revealed that these methods work as well as Sarsa(λ) with a tunable step-size parameter, but that they tend to out perform the original algorithm on continuous MDPs and slightly under perform on finite MDPs. This is a result of the difference between approximate and exact updates and suggests that in the case of finite MDPs an algorithm, such as HL(λ), designed under this assumption may be preferable. From the perspective of ecological optimality, VES and PARL2 out perform all other algorithms considered as they both perform well with minimal parameter tuning on the entire range of domains contained in RL Benchmark.

Finally, we turned to the update whitening problem and used the method of natural gradients to derive an algorithm for solving this problem approximately. The natural algorithms generally out performed their non-natural counter parts on the smaller set of domains used for evaluation. We concluded the study of natural temporal difference learning with an analysis of the vector valued adaptive step-size algorithms IDBD and Autostep in terms of our newly derived linear-time natural algorithms and found that they are closely related.

6.1 Future Work

Two important contributions of this thesis in terms of future work they encourage are the improved empirical methods for RL and the scalar step-size derivations under the separability assumption. The first provides a higher standard for experimental results in RL and strongly argues against the common practice of reporting results without evaluating the parameter tuning process used to generate them. Following these proposed methods has the potential to lead to better RL algorithms and more reproducible research results. The second contribution shows that future research may not need to solve both the adaptive scalar step-size and update whitening problem simultaneously. Instead, if we can prove this separability assumption holds, then it frees us to explore each problem separately and to later combine them to form new and more robust RL algorithms.

This also suggests that the future work naturally following from this dissertation is to combine the natural TD algorithms with one of the parameter-free adaptive scalar step-sizes. This is fairly straight forward in the case of the quadratic time algorithms as they have a single tunable step-size parameter which may be readily replaced by an adaptive algorithm such as VES. However, the linear-time natural TD algorithms are two-timescale algorithms and require two step-sizes. Some preliminary exploration of the problem shows that in this case we can replace one of these step-sizes with an adaptive algorithm without much difficulty, but that to replace both requires a fundamentally different approach than any existing methods. The problem is that the two step-sizes are inherently co-dependent, and they must be solved with this in mind.

APPENDIX

FUNCTION APPROXIMATION DETAILS

Domain	Fourier Basis Order
Acrobot	3
BicycleRiding	3
Cart Pole (Balance, 1-pole)	3
Cart Pole (Balance, 2-poles)	3
Cart Pole (Swing Up)	5
HIV Treatment	3
MountainCar	3
PuddleWorld	3
Planar Swimmer	9 (decoupled)

Table A.1: Fourier basis order used for continuous MDPs

BIBLIOGRAPHY

Hirotugu Akaike. On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method. *Annals of the Institute of Statistical Mathematics*, 11(1):1–16, 1959. URL `http://www.springerlink.com/index/L210170141677V14.pdf`.

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10: 251–276, 1998.

S. Amari and S. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, volume 2, pages 1213–1216, 1998a.

S. Amari and S.C. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, 1998*, volume 2, pages 1213–1216 vol.2, 1998b. doi: 10.1109/ICASSP.1998.675489.

Shun-ichi Amari. Natural gradient works efficiently in learning. 2000.

John Asmuth, Lihong Li, Michael L Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 19–26. AUAI Press, 2009.

J. A. Bagnell and J. Schneider. Covariant policy search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1019–1024, 2003.

L. Baird. Residual algorithms: reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

Jonathan Barzilai and Jonathan M. Borwein. Two-point step size gradient methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988. URL `http://imajna.oxfordjournals.org/content/8/1/141.short`.

Richard Bellman. A markovian decision process. *Journal of Mathematical Mechanics*, 6:679–684, 1957.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. In *Journal of Machine Learning Research*, 2012.

Dimitri P. Bertsekas and John Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

Dimitri P Bertsekas and John N Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee. Natural actor-critic algorithms. *Automatica*, 45(11):2471–2482, 2009.

David Blackwell. Discounted dynamic programming. *The Annals of Mathematical Statistics*, pages 226–235, 1965.

J. Boyan. Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 49–56, 1999.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

Paul R Cohen. *Empirical methods for artificial intelligence*, volume 139. MIT press Cambridge, 1995.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *The Journal of Machine Learning Research*, 7:551–585, 2006.

Christian Darken and John Moody. Note on learning rate schedules for stochastic optimization. In *Proceedings of the 1990 conference on Advances in neural information processing systems 3*, NIPS-3, pages 832–838, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. ISBN 1-55860-184-8. URL `http://dl.acm.org/citation.cfm?id=118850.119956`.

Christian Darken and John Moody. Towards faster stochastic gradient search. *Advances in Neural Information Processing Systems 4*, 1992.

Richard Dearden, Nir Friedman, and David Andre. Model based bayesian exploration. In *Proceedings of the fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.

T. Degris, P. M. Pilarski, and R. S. Sutton. Model-free reinforcement learning with continuous action in practice. In *Proceedings of the 2012 American Control Conference*, 2012.

Thomas G. Dietterich. The MAXQ method for hierarchical reinforcement learning. In *Proceedings of the fifteenth international conference on machine learning*, volume 8, 1998.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011a.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011b.

Michael Duff. Design for an optimal probe. In *ICML*, pages 131–138, 2003.

Damien Ernst, G-B Stan, Jorge Gonçalves, and Louis Wehenkel. Clinical data based optimal sti strategies for hiv: a reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, pages 667–672. IEEE, 2006.

Abraham P. George and Warren B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65: 167–198, 2006a.

Abraham P. George and Warren B. Powell. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Mach. Learn.*, 65(1): 167–198, October 2006b. ISSN 0885-6125. doi: 10.1007/s10994-006-8365-9. URL http://dx.doi.org/10.1007/s10994-006-8365-9.

Alborz Geramifard, Michael Bowling, Martin Zinkevich, and Richard S Sutton. ilstd: Eligibility traces and convergence analysis. *Advances in Neural Information Processing Systems*, 19:441, 2007.

Alborz Geramifard, Finale Doshi, Joshua Redding, Nicholas Roy, and Jonathan How. Online discovery of feature dependencies. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 881–888, 2011.

Alborz Geramifard, Robert H Klein, Christoph Dann, William Dabney, and Jonathan P How. RLPy: The Reinforcement Learning Library for Education and Research. http://acl.mit.edu/RLPy, 2013.

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *J. Artif. Intell. Res.(JAIR)*, 19:399–468, 2003.

William W. Hager. Updating the inverse of a matrix. *SIAM REVIEW*, 31:221—239, 1989. URL http://citeseer.uark.edu:8080/citeseerx/viewdoc/summary?doi=10.1.1.115.3334.

Arie Hordijk and Alexander A Yushkevich. Blackwell optimality. In *Handbook of Markov decision processes*, pages 231–267. Springer, 2002.

Ronald Howard. *Dynamic programming and Markov processes.* MIT Press, Cambridge, MA, 1960.

Marcus Hutter and Shane Legg. Temporal difference updating without a learning rate. *Advances in Neural Information Processing Systems*, 2007.

Marcus Hutter and Shane Legg. Temporal difference updating without a learning rate. *arXiv:0810.5631*, October 2008. URL `http://arxiv.org/abs/0810.5631`. Advances in Neural Information Processing Systems 20 (NIPS 2008) pages 705-712.

Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307, 1988.

S. Kakade. A natural policy gradient. In *Advances in Neural Information Processing Systems*, volume 14, pages 1531–1538, 2002.

Sham Kakade. *A Natural Policy Gradient.* 2001.

Harry Kesten. Accelerated stochastic approximation. *The Annals of Mathematical Statistics*, 29(1):41–59, March 1958. ISSN 0003-4851. doi: 10.1214/aoms/1177706705. URL `http://projecteuclid.org/euclid.aoms/1177706705`. Mathematical Reviews number (MathSciNet): MR93851; Zentralblatt MATH identifier: 0087.13404.

G. D. Konidaris, S. R. Kuindersma, R. A. Grupen, and A. G. Barto. Robot learning from demonstration by constructing skill trees. volume 31, pages 360–375, 2012.

Harold J. Kushner and George Yin. *Stochastic Approximation and Recursive Algorithms and Applications.* Springer, July 2003. ISBN 9780387008943.

M. Lagoudakis and R. Parr. Model-free least-squares policy iteration. In *Neural Information Processing Systems: Natural and Synthetic*, pages 1547–1554, 2001.

Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

Nicolas Le Roux, Pierre-Antoine Manzagol, and Yoshua Bengio. Topmoumoute online natural gradient algorithm. In *NIPS*, 2007.

Yann Lecun, Patrice Y. Simard, and Barak Pearlmutter. Automatic learning rate maximization by on-line estimation of the hessian's eigenvectors. In *Advances in Neural Information Processing Systems*, pages 156–163. Morgan Kaufmann, 1993.

John M. Lee. *Introduction to Smooth Manifolds.* Springer, 2003.

William S Lovejoy. A survey of algorithmic methods for partially observed markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.

G. D. Magoulas, V. P. Plagianakos, and M. N. Vrahatis. Adaptive stepsize algorithms for on-line training of neural networks. 2001. URL `http://www.math.upatras.gr/ vpp/pdf/a-4.pdf`.

Ashique Rupam Mahmood, Richard S. Sutton, Thomas Degris, and Patrick M. Pilarski. Tuning-free step-size adaptation. *International Conference on Acoustics, Speech, and Signal Processing*, 2012a.

Ashique Rupam Mahmood, Richard S. Sutton, Thomas Degris, and Patrick M. Pilarski. Tuning-free step-size adaptation. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2121–2124, 2012b.

Takamitsu Matsubara, Tetsuro Morimura, and Jun Morimoto. Adaptive step-size policy gradients with average reward metric. *Journal of Machine Learning Research-Proceedings Track*, 13:285–298, 2010.

F Mirozahmedov and S.P. Uryasev. Adaptive stepsize regulation for stochastic optimization algorithm. In *Zurnal vicisl. mat. i. mat. fiz.*, volume 23(6), pages 1314–1325, 1983. URL `http://130.203.133.150/showciting;jsessionid=C42DBE672C7A869A5EF97307369C1801?ci`

Joseph Modayil, Adam White, and Richard S Sutton. Multi-timescale nexting in a reinforcement learning robot. In *From Animals to Animats 12*, pages 299–309. Springer, 2012.

T. Morimura, E. Uchibe, and K. Doya. Utilizing the natural gradient in temporal difference reinforcement learning with eligibility traces. In *International Symposium on Information Geometry and its Application*, pages 256–263, 2005.

Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2nd edition, July 2006. ISBN 0387303030.

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71:1180–1190, 2008.

Matteo Pirotta, Marcello Restelli, and Luca Bascetta. Adaptive step-size for policy gradient methods. In *Advances in Neural Information Processing Systems*, pages 1394–1402, 2013.

Rajesh Ranganath, Chong Wang, Blei David, and Eric Xing. An adaptive learning rate for stochastic variational inference. In *Proceedings of The 30th International Conference on Machine Learning*, pages 298–306, 2013.

Martin Riedmiller. *Rprop - Description and Implementation Details*. 1994.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.

Stéphane Ross, Paul Mineiro, and John Langford. Normalized online learning. *arXiv preprint arXiv:1305.6646*, 2013.

Ilya O. Ryzhov, Peter I. Frazier, and Warren B. Powell. A new optimal step size for approximate dynamic programming. 2012.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *arXiv:1206.1106*, June 2012. URL http://arxiv.org/abs/1206.1106.

Nicol N. Schraudolph. Online learning with adaptive local step sizes. In *Neural Nets—WIRN Vietri-99: Proceedings of the 11th Italian Workshop on Neural Nets*, pages 151–156, 1999. URL http://cnl.salk.edu/ schraudo/pubs/Schraudolph99c.pdf.

Hugo Simao and Warren Powell. Approximate dynamic programming for management of high-value spare parts. *Journal of Manufacturing Technology Management*, 20 (2):147–160, February 2009. ISSN 1741-038X. doi: 10.1108/17410380910929592. URL http://www.emeraldinsight.com/journals.htm?articleid=1770975.

Özgür Şimşek and Andrew G Barto. An intrinsic reward mechanism for efficient exploration. In *Proceedings of the 23rd international conference on Machine learning*, pages 833–840. ACM, 2006.

D. Slate. UCI machine learning repository, 1991. URL http://archive.ics.uci.edu/ml.

Malcolm Strens. A bayesian framework for reinforcement learning. In *ICML*, pages 943–950, 2000.

Wenyu Sun and Ya-Xiang Yuan. *Optimization theory and methods: nonlinear programming*, volume 1. springer, 2006.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998a.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1998b.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12*, pages 1057–1063, 2000.

Richard Sutton. Gain adaptation beats least squares? In *In Proceedings of the 7th Yale Workshop on Adaptive and Learning Systems*, pages 161–166, 1992a.

Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the National Conference on Artificial Intelligence*, pages 171–171, 1992b. URL http://rlai.cs.ualberta.ca/ sutton/papers/sutton-92a-remastered.pdf.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Neural Information Processing Systems 8*, pages 1038–1044, 1996.

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

Yuval Tassa, Tom Erez, and William D Smart. Receding horizon differential dynamic programming. In *NIPS*, 2007.

P. S. Thomas. Bias in natural actor-critic algorithms. Technical Report UM-CS-2012-018, Department of Computer Science, University of Massachusetts at Amherst, October 2012.

John N Tsitsiklis. On the convergence of optimistic policy iteration. *The Journal of Machine Learning Research*, 3:59–72, 2003.

Goh Khang Wen, Mustafa Mamat, Ismail bin Mohd, and Yosza Dasril. A novel of step size selection procedures for steepest descent method. *Applied Mathematical Sciences*, 6(51):2507–2518, 2012. URL `http://www.m-hikari.com/ams/ams-2012/ams-49-52-2012/mamatAMS49-52-2012-2.pdf`.

Alexis P Wieland. Evolving neural network controllers for unstable systems. In *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, volume 2, pages 667–673. IEEE, 1991.

Laurenz Wiskott and Terrence J Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural computation*, 14(4):715–770, 2002.

Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969. URL `http://epubs.siam.org/doi/abs/10.1137/1011036`.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

Max Woodbury. Inverting modified matrices. Princeton University, 1950.

Zhirong Yang and Jorma Laaksonen. Principal whitened gradient for information geometry. *Neural Networks*, 21(2):232–240, 2008.

Ya-xiang Yuan. Step-sizes for the gradient method. *AMS IP STUDIES IN ADVANCED MATHEMATICS*, 42(2):785, 2008. URL `ftp://159.226.92.9/pub/yyx/papers/p0504.pdf`.