A Reinforcement Learning Controller for Functional Electrical Stimulation of a Human Arm

BY

Philip Sebastian Thomas, B.S.

Submitted in partial fulfillment of the requirements

for the degree of Master of Science

Thesis Advisor: Dr. Michael S. Branicky

Department of Electrical Engineering and Computer Science

Case Western Reserve University

August, 2009

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Philip S. Thomas

candidate for the Master of Science degree *.

(signed) Michael S. Branicky
(chair of the committee)

Antonie van den Bogert

Soumya Ray

(date) 19 June 2009

*We also certify that written approval has been obtained for any proprietary material contained therein.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

A Reinforcement Learning Controller for Functional Electrical Stimulation of a Human Arm

Abstract

by

PHILIP SEBASTIAN THOMAS

This thesis demonstrates the feasibility of using reinforcement learning (RL) for functional electrical stimulation (FES) control of a human arm as an improvement over (i) previous closed-loop controllers for upper extremities that are unable to adapt to changing system dynamics and (ii) previous RL controllers that required thousands of arm movements to learn. We describe the relevance of the control task and how it can be applied to help people with spinal cord injuries. We also provide simulations that show previous closed-loop controllers are insufficient. We provide background on possible RL techniques for control, focusing on a continuous actor-critic architecture that uses function approximators for its mappings. We test various function approximators, including Artificial Neural Networks (ANNs) and Locally Weighted Regression (LWR) for this purpose. Next, we introduce a novel function approximator, Incremental Locally Weighted Regression (ILWR), which is particularly suited for use in our RL architecture. We then design, implement, and perform clinically relevant tests using ANNs for the two mappings in the continuous actor-critic. During these trials, unexpected behavior is observed and eventually used to create a hybrid controller (that switches among different learning parameter sets) that can both adapt to changes in arm dynamics in 200 to 300 arm movements and remain stable in the long-term. A non-switching controller with similar performance is achieved using ILWR in place of an ANN for the controller's critic mapping.

ACKNOWLEDGEMENTS

There are many people without whom this thesis would not have come to fruition. To my family, thank you for the emotional support and guidance you have given me throughout my life. I cannot sufficiently express my gratitude for the continual support that you have provided.

I have also been lucky to have had, since high school, the continuous guidance of my advisers. In high school, David Kosbie introduced me to the wonders of computer science. As an undergraduate student at CWRU, Professor Michael Branicky introduced me to the field of artificial intelligence and reinforcement learning, and eventually recruited me for the research project on which this thesis is founded. As my thesis adviser, he has not only taught me how to perform research, but also assisted me throughout the entire process.

The initial idea of applying reinforcement learning to functional electrical stimulation was developed by Dr. Antonie van den Bogert, who also advised me throughout this research effort, and whose understanding of the underlying biomechanical systems has been indispensible. I have also consulted with many others who have provided valuable insight, notably Kathleen Jagonik and Professor Soumya Ray.

I would also like to thank the National Institute of Health for funding this research via NIH Grant R21HD049662. Finally, I would like to thank my friends who have supported me throughout the writing of this thesis by keeping me on task by asking how it was progressing, and who also provided emotional support as well as a reprieve from work.

CHAPTER 1:

INTRODUCTION

1.1 Functional Electrical Stimulation (FES)

An estimated 255,702 people in the United States suffer from spinal cord injuries (SCI), with approximately 12,000 new cases each year, 42% of which result from motor vehicle crashes (NSCISC, 2008). People with SCI frequently suffer from paralysis, rendering them unable to move their limbs, though most of their nerves and muscles may be intact. Functional Electrical Stimulation (FES) can activate these muscles to restore movement by activating motor neurons with electrical currents, which are applied via subcutaneous probes. Figure 1.1 depicts a typical FES setup. By intelligently selecting the current applied to the motor neurons associated with each muscle, individual muscles can be stimulated by various amounts, allowing researchers to control a subject's muscles. For background information on FES, refer to (Sujith, 2008; Ragnarsson, 2008; Sheffler and Chae, 2007; Peckham and Knutson, 2005).

For the purpose of this thesis, it is sufficient to view FES from a control perspective in which the muscles to be stimulated are a system or plant whose state is determined by joint angles, joint angle velocities, and several hidden states, as depicted in Figure 1.2. The plant inputs are electrical stimulation levels for each muscle, ranging from 0% to 100%, which result in muscle forces that change the state. The controller is given the current and target states, and generates the stimulations required to reach the target state.

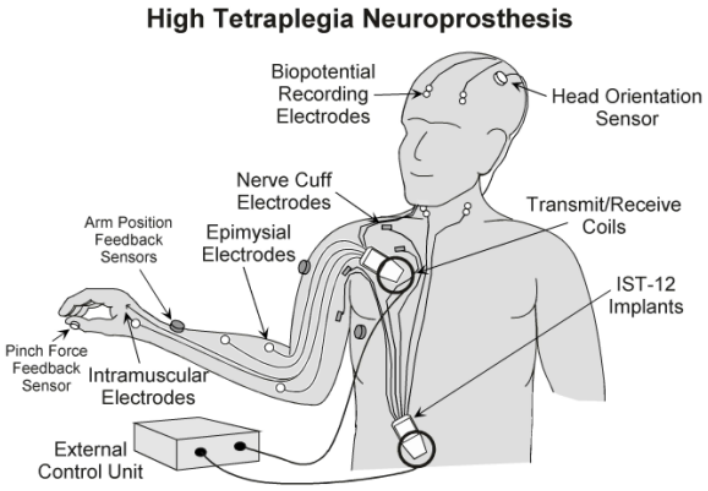


Figure 1.1: Typical FES setup in which the control unit communicates with implanted electrodes and sensors via coils that transmit, receive, and provide power to the internal system. Reproduced from Peckham and Knutson (2005).

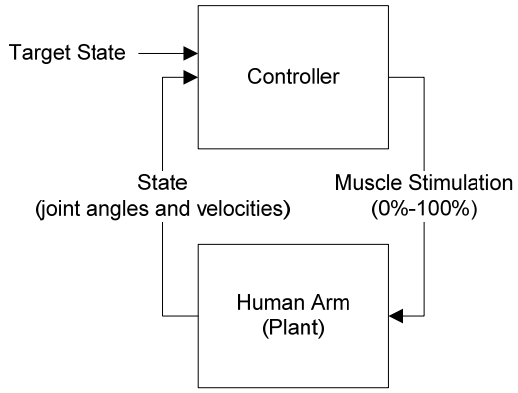


Figure 1.2: Block diagram of FES as a control task in which the controller receives the current state and target state, and generates muscle stimulations, which in turn affect the plant, which updates the state given to the controller.

Open-loop FES control has been successfully applied to basic systems such as hand grasp (Peckham et al., 2001), rowing (Wheeler et al., 2002) and gait (Kobetic and Marsolais, 1994; Braz et al., 2007). In order to produce accurate movements, open-loop control requires detailed

information about the system properties, which is often not available for more complex tasks that have not been accurately modeled, or whose dynamics change over time (Crago et al., 1996).

Closed-loop FES control does not have these drawbacks, and has therefore been used successfully on more complicated tasks such as hand grasp (Crago et al., 1991), knee joint position control (Chang et al., 1997), and standing up (Ferrarin et al., 2002). However, challenges related to using the required sensors have prevented feedback control from being applied in a clinical setting (Jaeger, 1992). Complex controllers that require detailed state information have been tested only in simulation (Stroeve, 1996; Abbas and Triolo, 1997).

Jagodnik and van den Bogert (2007) designed a Proportional Derivative controller (PD controller; see Section 2.1) for planar control of a paralyzed subject's arm. The gains for the PD controller were tuned to minimize joint angle error and muscle forces for a two-dimensional arm simulation using a *Hill-based muscle model* (Schultz et al., 1991). During human trials, Jagodnik and van den Bogert (2007) found that the PD controller's gain matrix often required retuning to account for variations in the dynamics of the subject's arm both from day to day, as well as within one FES session. The subject's arm also differed from the ideal arm used in simulation because it had unpredictable biceps stimulation due to spasticity. Results from simulation, which are provided in Section 2.1, support the claim that PD controllers do not perform well with such unmodelled or changing dynamics.

In practice, basic closed-loop controllers, such as the Proportional Integral Derivative controller (PID controller; see Section 2.1), must be manually tuned to each subject to account for differing dynamics between simulation, real-world application, and each individual arm. Additionally, these controllers require retuning during consecutive trials on the same subject.

The dynamics between two subjects may vary as a result of different physical dimensions, muscle strengths, muscle atrophy, and muscle spasticity, while the dynamics of each subject's arm changes primarily due to muscle fatigue, which is exacerbated by FES's high stimulation frequency compared to a healthy central nervous system (Lynch and Popovic, 2008).

Reinforcement learning (RL) techniques (Sutton and Barto, 1998) can be used to create controllers that adapt to these changes in system dynamics, finding non-obvious and efficient strategies. Within FES, RL has been tested in simulation to control a standing up movement (Davoodi and Andrews, 1998), but this did not require generalization and did not include varying target states. RL has also been shown to control arm movements (Izawa et al., 2004), however, learning required too many training movements for clinical applications (between 2,000 and 5,000).

1.2 Problem Statement (Adaptive RL FES Controller Task)

In this thesis, which is an extension of prior research by Thomas et al. (2008a; 2009a), we show the feasibility of using RL for FES control of a human arm as an improvement over previous closed-loop controllers for upper extremities that are unable to adapt to changing system dynamics. Specifically, we control horizontal planar movement of the right arm of a subject with complete paralysis in a simulated environment without friction.

The arm model we use, called the *Dynamic Arm Simulator 1* (DAS1), described and utilized in (Blana, Kirsch, and Chadwick, 2009), has two joints (shoulder and elbow) and is driven by six muscles; see Figure 1.3. Two of the six muscles act across both joints. Each muscle

is modeled by a three-element Hill model (cf. Section 1.1) and simulated using two differential equations, one for activation and one for contraction (McLean et al., 2003). Consequently, muscle force is not directly controlled but indirectly via muscle dynamics. The internal muscle states (active state and contractile element length) are hidden and not available to the controller. The biomechanics and dynamics are identical to those used by Jagodnik and van den Bogert (2007).

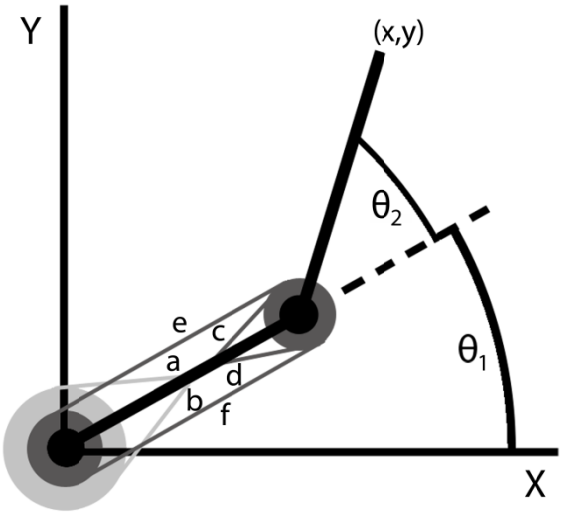


Figure 1.3: Two-joint, six-muscle biomechanical arm model used. Antagonistic muscle pairs are as follows, listed as (flexor, extensor): monoarticular shoulder muscles (a: anterior deltoid, b: posterior deltoid); monoarticular elbow muscles (c: brachialis, d: triceps brachii (short head)); biarticular muscles (e: biceps brachii, f: triceps brachii (long head)).

The state at time t, $s(t) \in \mathbb{R}^6$, is defined as

$$s(t) = \left[\theta(t), \dot{\theta}(t), \theta_{\text{Goal}}(t)\right]^T, \tag{1.1}$$

where $\theta(t)$ is a vector of the shoulder and elbow joint angles,

$$\theta(t) = \left[\theta_1(t), \theta_2(t)\right]^T, \tag{1.2}$$

$\dot{\theta}(t)$ is a vector of joint angular velocities, $\theta_{\text{Goal}}(t)$ is a vector of target joint angles, and T denotes matrix transpose.

The goal of this thesis is to create a RL controller that begins with a policy similar to that of the PD controller of Jagodnik and van den Bogert (2007), but that is also stable, robust to sensor noise, and capable of adapting to realistic changes in arm dynamics within 200 to 300 movements. The task of creating such a controller will be referred to as the *Adaptive RL FES Controller Task*. Because the RL controller must be able to adapt to continuously changing dynamics, learning rates and exploration should not be decayed.

1.3 Thesis Contribution

As far as we know, this research is the first successful effort to tackle the Adaptive RL FES Controller Task, defined previously in Section 1.2. This thesis both demonstrates that RL control is feasible and deserves future research, and also serves as a guide for the implementation of such a system in human trials. In creating the necessary RL controller, we utilize function approximators to represent the mapping of states to both actions and other values. To improve performance of the controller, we create a novel function approximator, Incremental Locally Weighted Regression (ILWR), described in Chapter 3. The benefits of using ILWR over the more common Artificial Neural Networks (ANNs), described in Section 2.3.1, are discussed in Chapters 3 and 7.

Further contribution to the RL literature is summarized in Section 8.1. The work leading to this thesis has also been published in two conference papers (Thomas et al., 2008a; Thomas et

al., 2009a) and several research posters (Thomas et al., 2008b; Thomas et al., 2008c; Jagodnik et al., 2008; Thomas et al., 2009b).

1.4 Thesis Outline

This thesis is separated into eight chapters and six appendices as follows.

Chapter 1 defines the problem of FES control, provides an overview of prior research, and presents information about the purpose and layout of this document.

Chapter 2 covers background information used in subsequent chapters. The first section, 2.1, covers PD and PID controllers and demonstrates why they are insufficient for the Adaptive RL FES Controller Task, defined in Section 1.2. Section 2.2 covers the basics of reinforcement learning and introduces the continuous actor-critic architecture, which is the foundation for all reinforcement learning trials performed in the subsequent chapters. Section 2.3 introduces the function approximators used throughout this thesis in the continuous actor-critic architecture. Section 2.4 concludes the chapter with a case study of the continuous actor-critic.

Chapter 3 introduces the Incremental Locally Weighted Regression (ILWR) algorithm that we devised for use in the actor-critic as a local function approximator. It is then tested on simple problems.

Chapter 4 contains a summary of the tests devised to evaluate the actor-critic's ability to adapt to realistic and clinically relevant changes in arm dynamics, as required in the Adaptive RL FES Controller Task.

Chapter 5 presents the results from the tests described in Chapter 4, using ANNs to represent the actor and critic. Different parameter settings achieve either rapid initial learning or long-term stability, but not both. Chapter 5 concludes with discussion of an unexplained phenomenon.

Chapter 6 provides several attempts to achieve rapid initial learning as well as long-term stability, and concludes with the creation of a hybrid controller that combines the best attributes of the different results from Chapter 5.

Chapter 7 provides the results from the tests described in Chapter 4, using ILWR to represent the critic. The resulting controller achieves both rapid initial learning as well as long-term stability—without the need to switch between various parameter settings.

Chapter 8 concludes with a summary of the results and recommendations for future work.

Appendix A presents a derivation of the derivative of the ILWR function approximator's output with respect to the outputs of its knowledge points.

Appendix B presents a derivation of the derivative of the ILWR function approximator's output with respect to the inputs of its knowledge points.

Appendix C provides an algorithm for efficiently computing the derivative derived in Appendix B.

Appendix D provides a derivation of equations relating to the gradient descent algorithm.

Appendix E provides the setup files for DAS1, the arm simulation model used.

Appendix F provides a summary of all parameter sets used in the continuous actor-critic.

1.5 Implementation

All implementation in this thesis was done in Microsoft Visual C++ 2008 Professional using the Microsoft Windows Vista Ultimate 64-bit operating system, running on a computer with 6GB of RAM, and an Intel Q6600 Quad Core 2.4GHz CPU, unless otherwise specified.

CHAPTER 2:

BACKGROUND

2.1 Proportional Derivative (PD) and Proportional Integral Derivative (PID) Controllers

Two of the simplest linear closed-loop controllers, which are commonly used in real-world control problems, are *Proportional Derivative* (PD) and *Proportional Integral Derivative* (PID) controllers (Franklin, Powell, and Workman, 1997). In this section, we first discuss the prior application of a PD controller to FES control of a human arm (Jagodnik and van den Bogert, 2007) and its limitations. Next, we introduce the PID controller and present experiments that illustrate why it is insufficient for our adaptive control task.

Jagodnik and van den Bogert (2007) trained a PD controller to move a subject's arm from an initial configuration, s_0, to a goal configuration, s_{Goal}. Configurations consist of the two joint angles, $\theta = \left[\theta_1, \theta_2\right]^T$, two target joint angles, $\theta_{\text{Goal}} = \left[\theta_{\text{Goal}_1}, \theta_{\text{Goal}_2}\right]^T$, and the time derivative of the joint angle, $\dot{\theta}$. To achieve realistic results, shoulder and elbow joint angles were both restricted to 20 to 80 degrees. The four-dimensional state was represented as

$$s(t) = \left[\theta(t) - \theta_{\text{Goal}}(t),\ \dot{\theta}(t)\right]^T. \tag{2.1}$$

The first two terms correspond to the error in the joint angles, while the latter two terms are the error in the joint angle time derivative (velocity) when the desired velocity at the target state is zero. Equation 2.1 is often referred to as the *error vector*.

Jagodnik and van den Bogert (2007) define the PD controller as Equation 2.2, where u is the vector of six muscle stimulation values, G is the six (muscles) by four (state) gain matrix, and s is a column vector of the current state, as defined in Equation 2.1.

$$u = Gs. \tag{2.2}$$

Jagodnik and van den Bogert (2007) applied a simulated annealing minimization algorithm to the DAS1 model to derive the gain matrix provided in Table 2.1. During human trials using these gains, an actual subject's arm moved toward its goal configuration smoothly. Upon reaching the goal configuration, it oscillated around the target, but usually stabilized within one second (Jagodnik and van den Bogert, 2007). At this point, the muscle stimulations are all small (less than .02). The weights were optimized for initial configurations and goal configurations between 20 and 80 degrees at each joint.

Label in Figure 1.3		$\theta_1 - \theta_{\text{Goal}_1}$	$\theta_2 - \theta_{\text{Goal}_2}$	$\dot{\theta}_1$	$\dot{\theta}_2$
a	u_1	-1.01787396	0.33212884	-0.15703080	-0.01493472
b	u_2	1.13413540	0.15551148	0.17941533	0.05653848
c	u_3	-0.00686023	-1.18587656	-0.03227234	-0.10250393
d	u_4	-0.17434505	1.03188031	0.01106836	0.07647948
e	u_5	0.49111238	0.97548825	0.11836239	0.08897244
f	u_6	-0.43426468	-0.72186559	-0.09872529	-0.07017570

Table 2.1: PD Controller gains derived by Jagodnik and van den Bogert (2007).

The PID controller has the same output equation as the PD, Equation 2.2, though the state is augmented to include the integral of the error:

$$s(t) = \left[\theta(t) - \theta_{\text{Goal}}(t), \quad \dot{\theta}(t), \quad \int_0^t \theta(\tau) - \theta_{\text{Goal}}(\tau)\, d\tau \right]^T . \tag{2.3}$$

When implemented, the integral error term was approximated using a backward rectangular approximation.

We implemented a PID controller to determine whether a more sophisticated closed-loop architecture could better cope with the changing dynamics of the arm. The gains were tuned using a variant of the First-Choice Random-Restart Hill Climbing minimization algorithm (Russell and Norvig, 1995) in which the gradient is sampled in fifty random directions, the best of which is followed. Each random direction involves random changes in up to 10 of the 36 dimensions. These changes were steps of 5% of each current gain value, with sign changes allowed as each weight approached 0. For the random restarts, the proportional and derivative gains were taken from the PD controller (Table 2.1), and the integral gains were chosen randomly between -1 and 1. We used the same evaluation criteria as Jagodnik and van den Bogert (2007).

To test the PID's ability to adapt to changing dynamics in simulation, the arm model was modified to include a baseline biceps stimulation. The biceps muscle (e in Figure 1.3) was given the PID's instructed stimulation plus an additional 20% (not to exceed 100%). This simulated the spasticity that was observed during human trials of the PD controller. When using the PID controller during a two-second episode with an initial state of shoulder joint angle $\theta_1 = .349$ (20°), elbow joint angle $\theta_2 = 1.571$ (90°) and a goal of $\theta_{\text{Goal}_1} = 1.571$ (90°), $\theta_{\text{Goal}_2} = .349$ (20°), the arm overshot the goal state by .216 radians (12.4°) for the shoulder

angle, and .231 radians (13.2°) on the elbow angle, which equates to an overshoot of 23cm, assuming the upper and lower arms are both .3m long. Over time, the integral term built up, and the arm settled to the correct steady state. Unlike the PD and PID controllers, the RL controller described in the next section learns to avoid steady state error and overshoot given this unexpected muscle spasticity.

Retuning of static linear controllers could restore performance but would require extensive trial-and-error experimentation to find the optimal controller for each subject. Such a design process would not scale well to systems with more muscles and more joints, especially considering that this trial-and-error must be performed on a human. Unlike the PD and PID controllers, an RL controller, as described in the next section, does not require manual retuning; it learns on its own to avoid overshooting the goal position when presented with unexpected muscle spasticity.

2.2 Reinforcement Learning (RL)

The purpose of this section is to provide necessary background knowledge in *reinforcement learning* (RL), and to explain the reasons for choosing the continuous actor-critic architecture. This section is divided further into ten subsections. Subsection 2.2.1 contains an overview of the history of RL techniques. Subsection 2.2.2 outlines the problem that defines RL. Subsection 2.2.3 defines the value function, which is used in Subsection 2.2.4 to define the optimal policy. Subsection 2.2.5 introduces the Q-function, which does not explicitly store the value function, and which can be updated using temporal-difference (TD) methods, discussed in Subsection 2.2.6. Subsection 2.2.7 introduces the discrete actor-critic architecture, which is another TD method, and presents results on a simple problem. Subsection 2.2.8 introduces a

version of the actor-critic architecture created for problems where both state and time are continuous. Subsection 2.2.9 provides the Stochastic Real-Valued Unit Algorithm, which is used when analyzing the continuous actor-critic architecture in Subsection 2.2.10.

2.2.1 History

The setup for RL originated as far back as 1950, when Shannon proposed using an action-value function similar to that of modern Q-learning (Shannon, 1950). Markov decision processes (MDPs) were presented in the late 1950s (Bellman, 1957) as a mathematical framework that would later be applied to RL. The RL problem was studied in its current form in the early 1970s as researchers such as Witten and Corbin (1973) began experimenting with MDPs. This research, later analyzed by Witten (1977), was actually a form of actor-critic learning, as was the paper published soon thereafter (Barto, Sutton, and Anderson, 1983). Actor-critic architectures stepped out of the limelight with the introduction of the Q-learning algorithm (Watkins, 1989). For background information and a more detailed history of RL, see (Kaelbling, 1996) and (Sutton and Barto, 1998).

Gullapalli (1990) introduced the Stochastic Real-Valued (SRV) Unit algorithm, reviewed in Subsection 2.2.9, as well as proofs of its convergence under certain strong assumptions. This algorithm was a type of actor-critic which was adapted by Doya (2000) for problems with continuous time and space.

2.2.2 Problem Statement

RL involves a program or robot, called an *agent*, learning which actions will maximize the rewards provided by the environment. The state of the environment at time t, $x(t) \in X \subseteq \mathbb{R}^n$ changes as a function of the agent's action, $u(t) \in U \subseteq \mathbb{R}^m$. The exact way the environment changes as a function of the agent's actions is described by either the stochastic *transition function*,

$$T(x,u,x') : X \times U \times X \to [0,1], \tag{2.4}$$

which represents the probability of progressing to state x' if the agent takes action u in state x, or the deterministic transition function,

$$T(x,u) : X \times U \to X, \tag{2.5}$$

which returns the state x' that results from taking action u in state x. Though our arm model, DAS1, is deterministic, the stochastic transition function is typically used in literature.

These definitions of the transition function satisfy the Markov property because the distribution of the next state can be determined using only the current state and action. The practical example of FES control of a human arm (both human trials and DAS1), does not satisfy this constraint due to hidden state variables of the plant, such as the lengths of the parallel elastic, series elastic, and contractile elements (McLean, Su, and van den Bogert, 2003), which cannot presently be directly measured using sensors nor practically imputed from observable variables. However, these internal states have fast decay times, resulting in little history dependence. The agent is therefore able to glean enough information about the environment from the observable

states to learn a reasonable policy, as observed in the results of this thesis. We therefore treat the environment as though the hidden states were not present, knowing that this may hinder results and remove convergence guarantees. We will thus assume that the Markov property is satisfied for the remainder of this thesis.

The agent has a *policy* (which may change over time) that specifies what action should be taken for any state, $\pi(x): X \to U$. After the agent takes action u in state x, the environment returns a *reward signal*,

$$R(x,u): X \times U \to \mathbb{R}. \tag{2.6}$$

In other sources, the reward function is independent of the agent's actions, mapping each state to a reward,

$$R(x): X \to \mathbb{R}. \tag{2.7}$$

We will use the latter formulation. This reward signal defines the problem, in which the agent seeks the policy that maximizes the reward signal over time. A block diagram of the agent's interaction with the environment is provided in Figure 2.1.

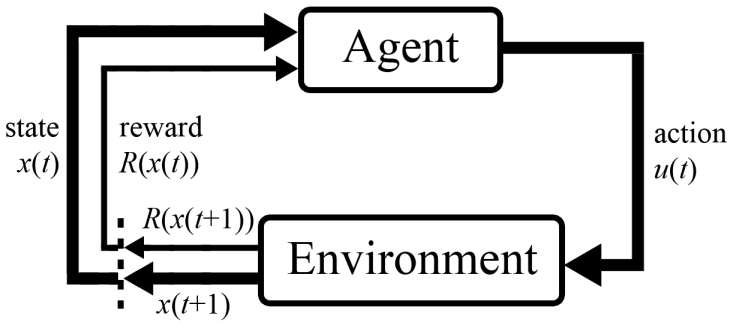


Figure 2.1: RL problem, figure adapted from (Sutton and Barto, 1998, Figure 3.1).

At time *t*, the agent should not necessarily take the action that maximizes only the reward received from the current action, because such an action may result in low rewards in the future. Rather, the agent should maximize the time-decayed sum of expected future rewards,

$$E\left\{\sum_{k=t}^{\infty}\gamma^{k-t}R\left(x\left(k\right)\right)\middle|x\left(t\right),\pi\right\}, \tag{2.8}$$

where $0<\gamma<1$ is a time decay constant (also known as the *discount rate*), the current time is *t*, and the state and actions follow the transition function and the current policy, π, from the current state, $x\left(t\right)$. The decay of the reward signal over time is necessary because the infinite sum of the reward signal without decay may diverge.

2.2.3 Value Function

The agent will often keep an approximation of the *value function*, sometimes called the *utility function*, which represents the time-decayed sum of expected future reward, when carrying out the policy π starting in state *x*.

$$V^{\pi}\left(x\right)=E\left\{\sum_{t=0}^{\infty}\gamma^{t}R\left(x\left(t\right)\right)\middle|x\left(0\right)=x,\pi\right\}. \tag{2.9}$$

The value function for the optimal policy is known as the *optimal value function*, which can be represented by

$$V^{*}\left(x\right)=\max_{\pi}V^{\pi}\left(x\right). \tag{2.10}$$

The *Bellman Equation* (Russell and Norvig, 1995) states that

$$V^{\pi}(x) = R(x) + \gamma \sum_{x'} T(x, \pi(x), x') V^{\pi}(x'), \tag{2.11}$$

in the stochastic case, and

$$V^{\pi}(x) = R(x) + \gamma V^{\pi}(x'), \tag{2.12}$$

in the deterministic case, where

$$x' = T(x, \pi(x)). \tag{2.13}$$

Similarly, for the optimal value function,

$$V^{*}(x) = R(x) + \gamma \max_{u} \sum_{x'} T(x, u, x') V^{*}(x'), \tag{2.14}$$

for the stochastic case, and

$$V^{*}(x) = R(x) + \gamma V^{*}(x'), \tag{2.15}$$

for the deterministic case, where

$$x' = T\left(x, \arg\max_{u} V^{*}(T(x, u))\right). \tag{2.16}$$

The value function can be iteratively approximated using the *Bellman update*, derived from Equations 2.11 and 2.12 as

$$V_{i+1}(x) \leftarrow R(x) + \max_{u} \gamma V_{i}(T(x, u)), \tag{2.17}$$

in the deterministic case, or

$$V_{i+1}(x) \leftarrow R(x) + \gamma \max_u \sum_{x'} T(x,u,x') V_i(x'), \tag{2.18}$$

in the stochastic case. The continual application of these equations has been shown to converge to a unique optimal solution, $V^*(x)$, if the agent chooses its actions greedily with respect to the value function as in Equation 2.19 for the deterministic case or Equation 2.20 for the stochastic case (Russell and Norvig, 1995):

$$u(t) = \arg\max_u V\left(T\left(x(t),u\right)\right) \tag{2.19}$$

$$u(t) = \arg\max_u \sum_{x'} T\left(x(t),u,x'\right) V\left(x'\right). \tag{2.20}$$

2.2.4 Optimal Policy

The agent's goal is to learn a policy as similar as possible to the optimal policy,

$$\pi^*(x) = \arg\max_u V^*(x), \tag{2.21}$$

which maximizes the expected future reward. If the transition function is known, then the optimal policy can be determined by solving the Bellman equation to find the optimal value function, and then selecting actions by

$$\pi^*(x) = \arg\max_u \left(\sum_{x'} T(x,u,x') V^*(x') \right), \tag{2.22}$$

in the stochastic case, and

$$\pi^*(x) = \arg\max_u V^*(T(x,u)), \tag{2.23}$$

in the deterministic case.

2.2.5 Q-Functions

Rather than explicitly storing the value function, some agents may keep an approximation of the *Q-function* for policy π, $Q^\pi(x,u)$, which represents the expected sum of future rewards if the agent takes action *u* starting in state *x*. This function, also known as the *action-value function*, can be written as

$$Q^\pi(x,u) = E\left\{\sum_{t=0}^{\infty} \gamma^t R(x(t)) \middle| x(0) = x, u(0) = u, \pi\right\}. \tag{2.24}$$

If an agent can find the optimal Q-function,

$$Q^*(x,u) = \max_\pi Q^\pi(s,a), \tag{2.25}$$

then it can derive the optimal policy as

$$\pi^*(x) = \arg\max_{u \in U} Q^*(x,u). \tag{2.26}$$

Q-learning is more appropriate to our problem because, unlike using only a value function via the Bellman update, a model of the environment, *T*, is not required if the agent has access to the Q-function. Though use of the Q-function does not require the agent to explicitly represent the transition function, the dimensionality of the problem has increased from learning

$V:\mathbb{R}^n \to \mathbb{R},$ to learning $Q:\mathbb{R}^{n+m} \to \mathbb{R},$ where *n* is the dimension of the state space, and *m* is the dimension of the action space.

2.2.6 Temporal Difference (TD) Methods

In order to learn the optimal value and Q-functions, agents often utilize the *temporal-difference* (TD) *error* or simply *TD-error*,

$$\delta = R\left(x(t)\right) + \gamma V^{\pi}\left(x(t+1)\right) - V^{\pi}\left(x(t)\right), \tag{2.27}$$

which represents the difference between the observed reward plus the resulting next state's value and the expected future reward that was predicted. Notice that Equation 2.12, the definition of *V*, is satisfied when the TD-error is zero. If the observed reward and resulting state value is larger than the prediction, the TD-error is positive, whereas if the observed reward and resulting state value are smaller than the prediction, the TD-error is negative. The TD-error update for the value function is

$$V^{\pi}\left(x(t)\right) \leftarrow V^{\pi}\left(x(t)\right) + \alpha\delta, \tag{2.28}$$

where $\alpha > 0$ is a learning rate. Similarly, the TD update for learning the Q-function is,

$$Q^{\pi}\left(x(t),u(t)\right) \leftarrow Q^{\pi}\left(x(t),u(t)\right) + \alpha\left[R\left(x(t)\right) + \gamma \max_{u'} Q^{\pi}\left(x(t+1),u'\right) - Q^{\pi}\left(x(t),u(t)\right)\right]. \tag{2.29}$$

In summary, if the transition function, *T*, is available, the value function, *V*, can be learned using the direct application of the Bellman equation. If *T* is not available, then the Q-

function can be learned using its TD update. Though the Q-function has the benefit of being model-free, its dimension is $n + m$, compared to the value function's dimension of n.

The actor-critic architecture, described in Subsections 2.2.7 and 2.2.8, is a model-free TD-learning method, which, in an attempt to sidestep the curse of dimensionality, splits the learning problem into two lower-dimensional problems relative to Q-learning. Also, Equations 2.18 through 2.26, in general, are impractical in a continuous environment because the maximum over u cannot necessarily be computed efficiently. Some work has been done to generate cases where this maximum can be efficiently computed (Hagen, 2000), though these methods have not gained popularity. The actor-critic architecture is better suited for adaptation to continuous time and space, as shown in Subsection 2.2.8.

2.2.7 Discrete Actor-Critic

Actor-critic methods are typically TD methods, which learn the policy in parallel with the value function, without the need for a model of the environment and without computing the full action-value function. The actor-critic consists of two components, shown in Figure 2.2. The *actor* represents the current policy, mapping states to actions, $A : X \to U$. The *critic* represents the value function for the current actor, $C : X \to \mathbb{R}$.

The actor-critic is designed to reduce dimensionality compared to a Q-learning style algorithm, as described further in Subsection 2.2.8. Overall, the agents will perform a type of gradient descent in policy space. From its current policy, the actor-critic will make minor changes to its policy. If the changes are beneficial (positive TD-error), they will be kept; if not (negative TD-error), they will be discouraged. This is akin to sampling the gradient in policy

space in one direction and either stepping in that direction or the opposite direction. However, the utility of each policy is approximated by the critic, which is itself only an approximation. Thus, only approximations of the points in policy space are available for computing the gradient, which can result in divergence.

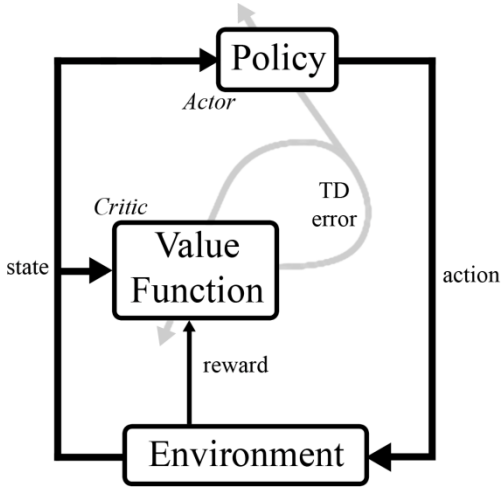


Figure 2.2: Actor-critic architecture diagram, recreated from (Sutton and Barto, 1998, Figure 6.15).

This approach is a minimization routine like any other, and is susceptible to the same problems, primarily local minima. Changes to the policy are implemented by adding exploration, in the form of noise, to the actions. The larger the noise, the greater the deviation from the current policy will be. It is common for exploration to be decayed as the policy improves, much like the temperature in a simulated annealing minimization routine.

The actor-critic architecture for discrete space and time is described in Chapters 6.6 (Actor-Critic Methods) and 7.7 (Eligibility Traces for Actor-Critic Methods) of (Sutton and Barto, 1998). Chapter 7 of (Sutton and Barto, 1998) also provides an introduction to *eligibility*

traces, which represent how much the value function at each visited state should be changed if a TD-error is observed at the current time, *t*. Equations 2.34 and 2.35 provide equations for exponentially decaying eligibility traces for the discrete actor-critic.

They can be added to most temporal-difference methods to increase learning efficiency. Without eligibility traces, a distant future reward will slowly propagate back to the states resulting in it. Using eligibility traces, the first time a reward is seen, the previously visited states resulting in the reward can be immediately updated.

> *[…] an eligibility trace is a temporary record of the occurrence of an event, such as the visiting of a state or the taking of an action. The trace marks the memory parameters associated with the event as eligible for undergoing learning changes. When a TD error occurs, only the eligible states or actions are assigned credit or blame for the error. Thus, eligibility traces help bridge the gap between events and training information. Like TD methods themselves, eligibility traces are a basic mechanism for temporal credit assignment.* – (Sutton and Barto, 1998)

The actor, defined in Equation 2.30, is stochastic to allow for exploration (which can be tapered over time or as the policy approaches the optimal policy). For the discrete implementation, both the actor and the critic are typically implemented using tables instead of

function approximators. The actor is represented by a table of probability values, $p(x,u)$, where the stochastic policy is computed using the *Gibbs Softmax* method provided in Equation 2.30 (Sutton and Barto, 1998),

$$\pi_t(x,u) = \Pr\{u_t = u \mid x_t = x\} = \frac{e^{p(x,u)}}{\sum_b e^{p(x,b)}}. \tag{2.30}$$

The critic is represented by a real-valued table representing the value of each state, $V(x)$.

The actor-critic works by generating an action using Equation 2.30, applying the action to the environment, generating a TD-error using the critic, and finally using the TD-error to update both the actor and the critic. The critic remains *on-policy*, meaning it approximates the value function for the current policy of the actor (Sutton and Barto, 1998). Equation 2.31 defines the TD-error, where r_t is the reward from progressing from state x_{t-1} to x_t, and γ is the discount rate of the utility,

$$\delta_t = r_{t+1} + \gamma V(x_{t+1}) - V(x_t). \tag{2.31}$$

Equation 2.32 provides the TD update equation for the critic, where α is the learning rate, and $V^\pi(x)$ is the approximated utility of state *x* given the current policy, π:

$$V^\pi(x_t) \leftarrow V^\pi(x_t) + \alpha\delta_t. \tag{2.32}$$

Equation 2.33 is the TD update equation for the actor, where β is the learning rate:

$$p(x_t,u_t) \leftarrow p(x_t,u_t) + \beta\delta_t. \tag{2.33}$$

Once the eligibility traces are defined, they can be readily added to these equations as a linear scaling factor on the TD-error for each state. The more difficult part is deciding how best to define the eligibilities. Variations of the definitions of Sutton and Barto (1998) were used and are provided in Equation 2.34 for the actor, and Equation 2.35 for the critic, where λ is a decay constant:

$$e_t(x,u) = \begin{cases} \gamma\lambda\left(e_{t-1}(x,u)+1\right), & \text{if } x = x_t, \text{ and } u = u_t, \\ \gamma\lambda e_{t-1}(x,u), & \text{otherwise,} \end{cases} \tag{2.34}$$

$$e_t(x) = \begin{cases} \gamma\lambda\left(e_{t-1}(x)+1\right), & \text{if } x = x_t, \\ \gamma\lambda e_{t-1}(x), & \text{otherwise.} \end{cases} \tag{2.35}$$

The eligibility traces can be added to the update equations as previously described, resulting in Equation 2.36 for the critic and Equation 2.37 for the actor:

$$V^{\pi}(x_t) \leftarrow V^{\pi}(x_t) + \alpha\delta_t e_t(x_t), \tag{2.36}$$

$$p(x_t,u_t) \leftarrow p(x_t,u_t) + \beta\delta_t e_t(x_t,u_t). \tag{2.37}$$

These Equations (2.30, 2.31, 2.34, 2.35, 2.36, and 2.37) were used to implement a simple discrete problem. An agent was placed randomly on a 10×10 grid and allowed to move up, down, left, or right. If it tried to move off the grid, it stayed in the same position for that iteration. Episodes were terminated when the agent reached the terminal state. Rewards were all zero except when the agent moved to the terminal state, where the reward was one. The utilities learned by the actor-critic are displayed in Table 2.2, with $\gamma = .9$. Eligibilities were cleared at the start of each episode.

0.14358	0.163716	0.190235	0.213739	0.240321	0.268046	0.297888	0.331051	0.370866	0.412367
0.164114	0.190959	0.217898	0.244666	0.272905	0.305	0.340462	0.377249	0.420366	0.46828
0.187638	0.217985	0.245377	0.275355	0.306591	0.342318	0.380683	0.423436	0.46984	0.523183
0.211337	0.244625	0.27606	0.309058	0.344233	0.382565	0.425359	0.473855	0.526008	0.583735
0.237658	0.272447	0.307345	0.343995	0.384081	0.427922	0.475969	0.527296	0.585228	0.65193
0.267387	0.304041	0.342326	0.382065	0.426934	0.475675	0.530004	0.588924	0.651768	0.726157
0.297307	0.340978	0.381293	0.426311	0.475233	0.52991	0.589838	0.655925	0.726743	0.807879
0.333444	0.37896	0.425197	0.47491	0.529278	0.5894	0.655785	0.728876	0.807826	0.898406
0.370964	0.421787	0.470426	0.528127	0.587141	0.655189	0.728628	0.80993	0.899958	0.999766
0.412948	0.467835	0.522984	0.58428	0.651165	0.724479	0.806889	0.898681	0.999989	N/A

Table 2.2: Critic (value function) generated using discrete actor-critic for the 10×10 gridworld described in the text, with $\gamma = 0.9$. The terminal state (10, 10) is the lower right.

Sutton and Barto (2007) assert that this is only one example of an actor-critic method, and that other variations may select actions in ways other than that provided in Equation 2.30. This variant does not reduce the dimensionality of the problem since the dimension of the actor is the same as that of the Q-function. The dimension reducing properties of the actor-critic are shown in the following section, which introduces the continuous actor-critic. The discrete and continuous cases must be differentiated because function approximators must be used to represent the continuous policy and value function, whereas the discrete system in this section can be implemented using tables. Similarly, minor changes to the TD-error equation must be made to account for the switch to continuous time.

2.2.8 Continuous Actor-Critic

Doya's *continuous actor-critic* (Doya, 2000) extends the discrete actor-critic to continuous time and space using function approximators and modified update equations. While most of the setup remains the same as with the discrete actor-critic, the function the actor

computes must be modified because the Gibbs Softmax method (Equation 2.30) does not work for continuous actions. It is therefore changed to $A: X \to U$, which maps states directly to actions. While Doya's update equation for the actor is derived from the SRV algorithm (Gullapalli, 1990), this new actor is not stochastic. Because exploration is no longer inherent to the actor, explorational noise must be artificially injected into the actions.

Doya provides the continuous counterpart of the TD error as

$$\delta(t) = r(t) - \frac{1}{\tau} V(x(t)) + \dot{V}(t), \tag{2.38}$$

where τ is a constant that determines the discount rate of rewards. Using a backward Euler approximation for the derivative of the value function, he finds

$$\delta(t) = r(t) + \frac{1}{\Delta t}\left[\left(1 - \frac{\Delta t}{\tau}\right) V(x(t)) - V(x(t - \Delta t))\right]. \tag{2.39}$$

The critic, a function approximator $V(x(t); w): X \to \mathbb{R}$, with parameter vector w, can be updated using exponential eligibility traces as

$$\kappa \dot{e}_i(t) = -e_i(t) + \frac{\partial V(x(t); w)}{\partial w_i}, \tag{2.40}$$

and

$$\dot{w}_i = \eta_C \delta(t) e_i(t), \tag{2.41}$$

where η_C is the critic's learning rate, κ scales the decay of the eligibility traces over time, and $0 < \kappa \leq \tau$ (Doya, 2000).

He then proposes that the actor be updated using the SRV algorithm, citing (Gullapalli, 1990). The update for the actor, $A(x(t); w^A)$, which has its own parameter vector, w^A, is

$$\dot{w}_i^A = \eta_A \delta(t) n(t) \cdot \frac{\partial A\left(x(t); w^A\right)}{\partial w_i^A}, \tag{2.42}$$

where η_A is the actor's learning rate. The actions of the actor are selected as

$$u(t) = S\left(A\left(x(t); w^A\right) + \sigma n(t)\right), \tag{2.43}$$

where σ is a constant, and S is a monotonically increasing function. For the remainder of this thesis, the sigmoid function, provided in Equation 2.62, is used for *S*. The term $n(t)$, which is the same dimension as the action space, defines the explorational noise that is injected into the policy. In his examples, Doya used

$$\tau_n \dot{n}(t) = -n(t) + N(t), \tag{2.44}$$

where $N(t)$ is normal Gaussian noise of the same dimension as the action space.

This thesis focuses on the application of the continuous actor-critic to the Adaptive RL FES Controller Task.

2.2.9 Stochastic Real-Valued Unit Algorithm (SRV Algorithm)

When the continuous actor-critic's update equation for the actor (Equation 2.42) is presented in (Doya, 2000), the Stochastic Real-Valued Unit algorithm (SRV algorithm; Gullapalli, 1990) is cited as the source and justification. Subsection 2.2.10 discusses convergence of the continuous actor-critic, for which a background on the SRV algorithm will be useful. It is therefore presented in this subsection.

In this system, the states are real-valued vectors in the state space $\mathbf{X} \subseteq \mathbb{R}^n$, and the actions are real numbers $\mathbf{U} \subseteq \mathbb{R}$. The rewards have fixed range $r(t) \in [0,1]$. The reward function

is defined by the distribution $G: \mathbf{R} \times \mathbf{X} \times \mathbf{U} \to [0,1]$, where $G(r,x,u) = \Pr\{r(t) \le r | x(t) = x, u(t) = u\}$. The optimal action can then be defined as

$$u^* = \arg\max_{u \in \mathbf{U}} \{E(r|x,u)\}. \tag{2.45}$$

The actor is represented by a parameter vector, $\boldsymbol{\theta}$, while the critic is represented by a parameter vector, ϕ. For a given time step, the state $x(t)$ is selected randomly, though the system obeys the transition function. The agent's action is generated stochastically via the Gaussian distribution

$$u(t) \sim N(\mu = \boldsymbol{\theta}^T x(t), \sigma = S(\hat{r}(t))), \tag{2.46}$$

where $\hat{r}(t) = \phi^T x(t)$ is the estimation of the reward to be received and S is a monotonically decreasing non-negative function, such that $S(1) = 0$. The variance of the action distribution decreases as the reward increases, which equates to decaying exploration as performance improves.

After generating the current action, the agent receives a reward for its action, $r(u(t), x(t))$. Recall that this signal is a random variable. The actor parameter array is then updated as follows.

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \sigma(t)(r(u(t), x(t)) - \hat{r}(t))(u(t) - \mu(t))x(t), \tag{2.47}$$

where

$$\mu(t) = \boldsymbol{\theta}^T x(t), \tag{2.48}$$

$$\sigma(t) = S(\hat{r}(t)), \tag{2.49}$$

$$\hat{r}(t) = \phi^T x(t), \tag{2.50}$$

and

$$u(t) \sim N\left(\mu(t), \sigma(t)\right). \tag{2.51}$$

The critic parameter array is updated such that

$$\phi(t+1) = \phi(t) + \rho\left(r\left(u(t), x(t)\right) - \hat{r}(t)\right)x(t), \tag{2.52}$$

where ρ is a learning rate.

Gullapalli (1990) shows that, under certain strong assumptions, this system will converge to the optimal actor.

2.2.10 Continuous Actor-Critic Analysis

The relation between Doya's update equation (Equation 2.42) for the actor and Gullapalli's SRV update (Equation 2.47) is not obvious. If one substitutes $\delta(t) = r\left(u(t), x(t)\right) - \hat{r}(t)$, which is not strictly true—though the two terms are related—the SRV update for the actor becomes

$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}(t) + \sigma(t)\delta(t)\left(u(t) - \mu\right)x(t). \tag{2.53}$$

Further substituting $w^A = \boldsymbol{\theta}$, and $n(t) = \sigma(t)\left(u(t) - \mu\right)$, which again is not true, though the two terms are related, yields

$$w^A(t+1) = w^A(t) + n(t)\delta(t)x(t). \tag{2.54}$$

This equation is identical to Doya's update because in the SRV algorithm,

$$x(t) = \frac{\partial A\left(x(t); w^A\right)}{\partial w_i^A}. \tag{2.55}$$

This adaptation of the SRV algorithm is not perfect, so all convergence guarantees no longer apply, though Gullapalli's intuitive justification carries over. This justification is reproduced below.

> *If this noise has caused the unit to receive a [reward] that is **more** than the expected evaluation, then it is desirable [that] the [actor] should [...] be changed in the direction of the noise. That is, if the noise is positive, the unit should update its parameters so that the mean value increases. Conversely, if the noise is negative, the parameters should be updated so that the mean value decreases. On the other hand, if the evaluation received is **less** than the expected evaluation, then the [actor should be updated] in the direction opposite to that of the noise.*
>
> – (Gullapalli, 1990)

Though this justification is still sound, the adaptation to the continuous actor-critic is still only an approximation.

To the best of our knowledge, there are no convergence proofs for the continuous actor-critic. This is likely due to mathematical complications introduced by the addition of function approximators. We must therefore rely on Gullapalli's intuition to understand why the continuous actor-critic learns. This intuition relies on the critic being accurate, which can become problematic, as shown in Section 2.4.

However, we can use Gullapalli's intuition to make informed decisions about parameter settings. With a positive TD-error, the actor ought to reinforce the explorational noise used. The actor's surface should be warped so the output at the current state approaches the output plus the current noise. Thus, the locality of the update to the actor should be such that the states close enough to benefit from similar exploration will be updated, but states further away (where the same exploration may not be beneficial) will not be updated.

In order to achieve an accurate critic, so that Gullapalli's intuition applies, the updates to the critic should eventually be local. However, global updates may initially be beneficial to allow the critic to take on the general shape of the value function. We therefore recommend increasing the locality of critic updates over time. This will be left to future work because the Adaptive RL FES Controller Task does not allow for decaying learning rates (which is akin to increasing the locality of updates).

2.3 Function Approximators

In order to implement the continuous actor-critic, *function approximators* are used to represent V and π (also known as A and C). These function approximators are then updated via a form of gradient descent during training. Function approximators can be split into two classes, local and global. When changes are made to a *local approximator*, the value of the function will only change within some neighborhood, while an update to a *global approximator* changes the value of the function over the entire domain.

Initial tests, described below, were performed with two global approximators, Artificial Neural Networks (ANNs) and Functional Link Networks (FLNs), and three local approximators,

k-Nearest Neighbor (*k*-NN), Locally Weighted Regression (LWR), and Radial Basis Functions (RBFs), all of which are defined in the following subsections. Though an approximator consisting of a linear combination of the inputs could exactly represent the PD controller, it was not included in tests because it is equivalent to a one-layer ANN with a linear activation function. In these trials, the approximators used a supervised learning technique, gradient descent, to approximate the value of states when using the PD controller as the policy for FES control of the simulated arm. In this chapter, these preliminary results are reviewed. Although these results are for supervised learning, they give insight into the learning speed of each method, which was then used to decide which approximators to implement in the actor-critic architecture.

For this task, called the *Utility Approximation Task* (UAT), 720,000 state-utility points were generated. These points consist of states of the form

$$x(t) = \left[\theta(t), \quad \dot{\theta}(t), \quad \theta_{\text{Goal}}(t) \right]^T, \tag{2.56}$$

and their empirically calculated utility. The utility was computed as the integral of the reward signal over a two-second episode:

$$\zeta \int_{t=0}^{2} \left[W \sum_{i=1}^{6} \left(\text{MuscleForce}_i^2 \right) + \left(\theta_{\text{Goal}} - \theta \right)^T \left(\theta_{\text{Goal}} - \theta \right) + \dot{\theta}^T \dot{\theta} \right] dt, \tag{2.57}$$

where $W = 164.14$ and $\zeta = -\left(\pi / 180 \right)^2$. This reward signal is similar to that used by Jagodnik and van den Bogert (2007) to create the PD controller discussed in Section 2.1, and also resembles the reward signal used in the actor-critic tests in Chapters 5 through 8. The start and

goal states for each episode were both sampled randomly from $\theta \in [.349, 1.571] \times [.349, 1.571] = [20°, 90°] \times [20°, 90°]$ and $\dot{\theta} \in [0,1] \times [0,1]$.

The 720,000 points were split into a training set of 550,000 points and a testing set of the remaining 170,000 points. Performance was judged based on the number of training iterations required for convergence, as well as the *sum of the squared error* (SSE) on the testing set, once converged. The SSE is defined as

$$SSE = \sum_{i=1}^{n} (x_i - h_i)^2, \tag{2.58}$$

where x is a scalar representing the desired result for the i^{th} testing point, $n = 170,000$ is the number of testing points, and h is a scalar representing the output of the approximator. The average SSE if all outputs were zero was 187,929.

The remainder of this chapter describes ANNs, FLNs, *k*-NN, LWR, and RBFs, and describes their performance on the task of learning the utility function. A summary of the performance of each function approximator is provided in Subsection 2.3.6.

2.3.1 Artificial Neural Networks (ANNs)

Because ANNs have been successfully applied to many classification, function approximation, and RL tasks (Baxt, 1990; Hutchinson, 1994; Le Cun et al, 1990; Leung et al., 1990; Pomerleau, 1993; Sejnowski et al., 1990; Shea and Liu, 1990), they serve as an experimental control for comparison to other methods. In this work, we use fully connected feed-forward ANNs with one or two hidden layers. We use notation similar to that of Russell and

Norvig (1995). In this notation, the output of the l^{th} node in the input layer is denoted a_l, the output of the k^{th} node in the hidden layer following the inputs (called the second hidden layer in this thesis) is denoted a_k, the output of the j^{th} node in the first hidden layer is denoted a_j, and the output of the i^{th} output node is denoted a_i.

The output for each node is defined as

$$a_i = S\left(in_i\right), \tag{2.59}$$

where S is the *activation function* (also called a *threshold function*) for the i^{th} node, and

$$in_i = \sum_j w_{j,i} a_j, \tag{2.60}$$

where w is an array of weights, and $w_{j,i}$ denotes the weight between the j^{th} node of the first hidden layer and the i^{th} node of the output layer. An additional node with a fixed output of -1 is added to each layer, other than the output layer, to serve as a threshold for neurons in the subsequent layer (Russell and Norvig, 1995). Equations 2.59 and 2.60 apply to all layers, with the sum in Equation 2.60 being over the neurons in the previous layer. For further description of the notation, see (Russell and Norvig, 1995).

Neurons in the output layer use a *linear activation function*,

$$L\left(x\right) = x, \tag{2.61}$$

while all other neurons used the *sigmoid activation function*,

$$S\left(x\right) = \frac{1}{1+e^{-x}}. \tag{2.62}$$

The derivatives of Equations 2.61 and 2.62 are required for gradient descent updates such as error backpropagation, and are provided in Equations 2.63 and 2.64, respectively:

$$\frac{\partial}{\partial x} L(x) = L'(x) = 1, \tag{2.63}$$

$$\frac{\partial}{\partial x} S(x) = S'(x) = (1 - S(x)) S(x). \tag{2.64}$$

For future reference, the derivatives of the ANN output with respect to the weights in each layer are provided below. These are required for actor-critic updates. Equations 2.65, 2.66, and 2.67 provide the derivatives of an ANNs output with respect to a weight in the output layer, second hidden layer, and first hidden layer respectively.

$$\frac{\partial a_y}{\partial w_{j,i}} = \begin{cases} L'(in_i) a_j, & \text{if } y = i, \\ 0, & \text{otherwise,} \end{cases} \tag{2.65}$$

$$\frac{\partial a_i}{\partial w_{k,j}} = L'(in_i) w_{j,i} S'(in_j) a_k, \tag{2.66}$$

$$\frac{\partial a_i}{\partial w_{l,k}} = a_l \sum_j \left(L'(in_i) w_{j,i} S'(in_j) w_{k,j} S'(in_k) \right). \tag{2.67}$$

For training on the UAT, the backpropagation algorithm was used (Russell and Norvig, 1995). The computational cost of this algorithm resembles that of the actor-critic updates, and was therefore computed for various network sizes. Training and testing for one point using ANNs was efficient, requiring 23.4 microseconds to train a network with 20 neurons in its only hidden layer on one point, and 2.68 microseconds to run the same network on a query point. These values were determined using a 1.6 GHz AMD Turion 64 processor running 32-bit

Windows XP. Table 2.3 shows the training and testing times for various network sizes as well as their performance on the UAT. Each training epoch consisted of training on 550,000 points, and each testing epoch consisted of running the network on 170,000 points, as discussed at the top of Section 2.3.

First Hidden Layer Size	Second Hidden Layer Size	Training Time Per Epoch (Seconds)	Testing Time Per Epoch (Seconds)	Learning Rate	Epochs to Convergence	SSE
5	0	3.1148	.3476	.000001	121	20,000
10	0	6.912	.8	.00001	31	15,000
20	0	12.88	1.479	.000001	500	2,913
5	5	6.1188	.6844	.0001	1	11,088
10	5	9.4092	1.0564	.000001	100	5,000
10	10	12.875	1.422	.0000001	500	3,080

Table 2.3: Training and testing times for ANNs of various sizes on the UAT. Each size was tested with learning rates varying from .001 to .0000001, training for up to 500 epochs. The setups resulting in the smallest SSE for each network size are reported above.

The smallest SSE achieved by an ANN was 2,913 when using 20 neurons in its single hidden layer, with a learning rate of $\alpha = 1 \times 10^{-6}$ after training for 500 epochs over the training set of 550,000 points. Overall, learning with backpropagation was remarkably slow with respect to the number of updates required for convergence. With larger learning rates, the function failed to converge to an accurate value, while smaller learning rates require hundreds of epochs (on the order of 50 million updates) to converge.

2.3.2 Functional Link Networks (FLNs)

Functional Link Networks (FLNs) attempt to increase the learning speed and stability of ANNs during training by predetermining interesting features of the current state to use as additional inputs to the ANN (Klassen, Pao, and Chen, 1988). The features chosen were varied throughout these tests. The best found are provided in Table 2.4.

θ_1	θ_2	$\dot{\theta}_1$	$\dot{\theta}_2$
$\theta_{\text{Goal}_1} - \theta_1$	$\theta_{\text{Goal}_2} - \theta_2$	$\theta_1 \sin(\theta_1)$	$\theta_2 \sin(\theta_2)$
$\theta_1 \theta_2 \sin(\theta_2)$	$\sin(\theta_1)$	$\sin(\theta_2)$	$(\theta_{\text{Goal}_1} - \theta_1)^2$
$(\theta_{\text{Goal}_2} - \theta_2)^2$	$\dot{\theta}_1 \dot{\theta}_2 \sin(\theta_2)$	$(\dot{\theta}_2)^2 \sin(\theta_2)$	$(\dot{\theta}_1)^2 \sin(\theta_2)$

Table 2.4: 16 features found to work well for FLNs.

These features were chosen after manually inspecting the equations for the dynamics of the simulated arm. An SSE on the UAT of 4,893 was achieved using all 16 parameters in Table 2.4, after training for only 60 epochs with 13 neurons in the first hidden layer and no second hidden layer, $\alpha = 1 \times 10^{-6}$. Though this error is worse than that of the best ANN, learning occurred faster.

In conclusion, the improvement over the standard ANN was not significant enough to warrant further investigation.

2.3.3 *k*-Nearest Neighbors (*k*-NN)

k-Nearest Neighbors (*k*-NN) served mainly as a stepping stone to the implementation of Locally Weighted Regression (LWR), discussed below. A KD-Tree was implemented to store all the training instances for both *k*-NN and LWR. It was found that finding the *k*-nearest neighbors using the KD-Tree was not the computational choke point, so further consideration of nearest neighbor runtimes was not necessary.

For the remainder of this subsection, k is the number of nearest neighbor points used in the approximation of query point q. These are the k points with the smallest (usually Euclidean) distance to the query point. For a given definition of w_i, the approximation function is defined by Equation 2.68,

$$h(x) = \sum_{i=1}^{k} \frac{w_i \cdot U(x_i)}{\sum_j w_j}, \tag{2.68}$$

where $U(x_i)$ is the known output for the input x_i, which in the UAT is the numerically determined utility of the state x_i. In order to improve performance, the state was changed from Equation 2.56 to

$$x(t) = \left[\theta(t), \quad \dot{\theta}(t), \quad \theta_{\text{Goal}}(t) - \theta(t)\right]^T. \tag{2.69}$$

k-NN can be implemented using different weighting schemes. Tests were performed using no weighting, inverse weighting, and exponential weighting. The approximation function $h(x)$ is the same for each of these weighting schemes. Equation 2.70 defines the weights if

using no weighting, while Equation 2.71 defines the weights when using inverse-distance weighting, and Equation 2.72 defines the weights when using exponential distance:

$$w_i = 1, \tag{2.70}$$

$$w_i = \frac{1}{d^2(x_i, q)}, \tag{2.71}$$

$$w_i = 2^{-d(x_i, q)}. \tag{2.72}$$

In Equations 2.71 and 2.72, d is the Euclidean distance and q is the query point.

Unlike ANN-based algorithms, there is no training phase for k-NN, so it is more difficult to judge training time. However, using all 550,000 training points on the UAT, k-NN achieved much smaller errors than ANNs, as shown in Figure 2.3. The lowest error achieved was 1,286 using the inverse-distance weighting and k between 8 and 11.

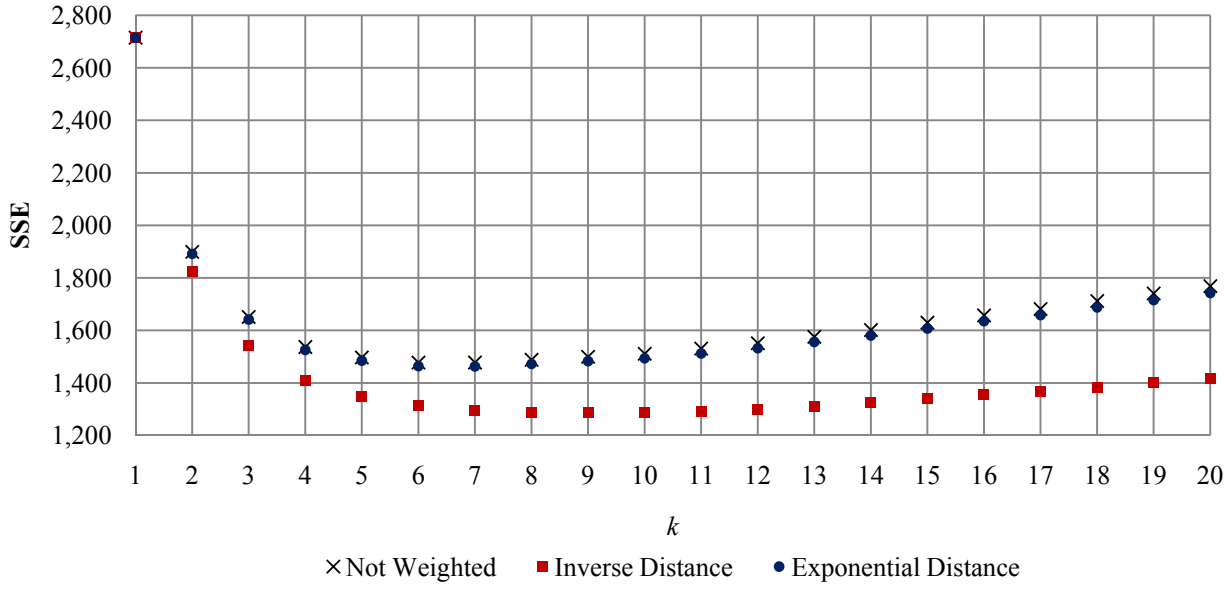


Figure 2.3: Sum of the squared error on the UAT testing set when using k-NN with varying k and weighting schemes.

The ANNs and FLNs have the innate property that they can be used to approximate a function that changes over time. Instance based methods like *k*-NN and LWR do not have this property because they are not based on the same system of training. Thus, in order for *k*-NN to be used in the actor-critic, it would have to be modified as with LWR in Chapter 3.

2.3.4 Locally Weighted Regression (LWR)

Like *k*-NN, *Locally Weighted Regression* (LWR) is a memory-based regression method, however, rather than merely taking a weighted average of the outputs for the *k*-nearest neighbors, LWR fits a linear model to the nearest neighbors, which it then uses to extrapolate the value at the query point. The fit for the linear model is computed to minimize the squared error of each of the *k*-nearest neighbors, weighted exponentially by the inverse of their distances from the query point. The *LWR Algorithm* is provided as Algorithm 2.1.

The parameter **D** is a diagonal matrix representing the weight of each dimension as well as the size of the neighborhood to be considered. Defining

$$\mathbf{D} = h \cdot diag\left(\left[n_1, n_2, ..., n_n\right]\right), \tag{2.73}$$

we split **D** into two components: *h*, which scales the neighborhood, and the array of n_i, which normalize the range of the input dimensions. Commonly, the values of *n* are fixed to

$$n_i = \frac{1}{\sigma_i^2}, \tag{2.74}$$

where σ_i is the standard deviation of the i^{th} dimension of the training points. The value for *h* on the UAT can then be determined using *leave-one-out cross validation*, defined in Algorithm 2.2.

The LWR Algorithm:

Given:

A query point $\mathbf{x}_q$ and p training points $\{\mathbf{x}_i, y_i\}$ in memory

Compute Prediction:

a) compute diagonal weight matrix $\mathbf{W}$ where

$$w_{ii} = \exp\left(-\frac{1}{2}\left(\mathbf{x}_i - \mathbf{x}_q\right)^T \mathbf{D}\left(\mathbf{x}_i - \mathbf{x}_q\right)\right)$$

b) build matrix $\mathbf{X}$ and vector $\mathbf{y}$ such that

$$\mathbf{X} = \left(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, ..., \tilde{\mathbf{x}}_p\right)^T \text{ where } \tilde{\mathbf{x}}_i = \left[\left(\mathbf{x}_i - \mathbf{x}_q\right)^T 1\right]^T$$

$$\mathbf{y} = \left(y_1, y_2, ..., y_p\right)^T$$

c) compute locally linear model

$$\beta = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{W}\mathbf{y}$$

d) the prediction for $\mathbf{x}_q$ is thus

$$\hat{y}_q = \beta_{n+1}$$

Algorithm 2.1: The LWR Algorithm, reproduced from (Schaal, Atkeson, and Vijayakumar, 2002).

A variation of leave-one-out cross validation was implemented to search for *h*, but it was computationally intensive when the number of candidate values for *h* was large ($|H|$ in Algorithm 2.2). It was therefore only used with a small set of candidate values to obtain an initial range for *h*, after which a manual search was done to find a near optimal *h*. This manual search was similar to Algorithm 2.2, except that the consecutive values of h_r were determined by the author after studying the results of previous values.

Leave-One-Out Cross Validation:

Given: A set H of reasonable values h_r

Algorithm:

For all $h_r \in H$:

$sse_r = 0$

For i=1:p

a) $x_q = x_i$
b) Temporarily exclude $\{x_i, y_i\}$ from training data
c) Compute LWR prediction $\hat{y}_q$ with reduced data
d) $sse_r = sse_r + \left(y_i - \hat{y}_q\right)^2$

Algorithm 2.2: The Leave-One-Out Cross Validation Algorithm, reproduced from Schaal et al., (2002), which can be used to determine the proper setting for h in Algorithm 2.1.

According to Schaal et al. (2002), the computational complexity of Algorithm 2.1 is $O\left(pn^2\right)$, where p is the number of training points, and n is the dimension of the query point. In order for the matrix inverse in LWR to be calculated (step c in Algorithm 2.1), the total number of training points used must be greater than the dimension of the output. To store the training points, the same KD-Tree used for k-NN was used. Because the Gaussian weighting kernel falls off steeply, only the k (fixed) nearest training points were run through the LWR Algorithm to generate an approximation. Figure 2.4 depicts the Gaussian weighting kernel for a sample point, exemplifying how most points need not be included in each regression.

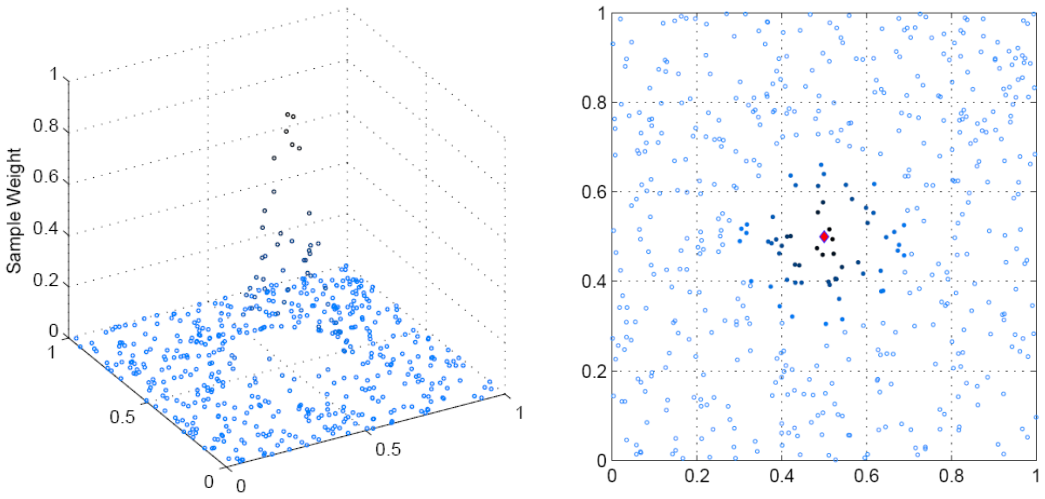


Figure 2.4: Locally Weighted Regression Point Weighting (query point (0.5, 0.5), marked by the diamond). Close by points that have significant weights (greater than 0.05) are marked by filled circles. Sample points on any given circle centered at the query point will have equal weights. Because only a small number of the closest points have non-negligible weights, the LWR algorithm can be run using only the *k*-nearest neighbors without significant loss of precision. Reproduced from (Wedge, 2004, Figure 3.5).

When testing on the UAT, the state was once again modified to that in Equation 2.61, as with *k*-NN. Table 2.5 shows the SSE on the testing set for various values of *h* and *k*. LWR's SSE of 294 is the lowest of all the approximators tested. It achieved this SSE without requiring the long training of ANNs and FLNs. As described for *k*-NN, this is also its drawback—it is not natively a learning algorithm, so it will have to be modified.

h	*k*	SSE
0.0001	10	553.504
0.0001	15	303.006
0.0001	**20**	**294.74**
0.0001	25	304.421
0.0001	30	320.461
1e-005	10	461.128
1e-005	15	302.998
1e-005	**20**	**294.758**
1e-005	25	304.444
1e-005	30	320.487
1e-006	10	463.428
1e-006	15	303.015
1e-006	**20**	**294.759**
1e-006	25	304.446
1e-006	30	320.489
1e-007	10	594.221
1e-007	15	303.031
1e-007	**20**	**294.758**
1e-007	25	304.443
1e-007	30	320.486

Table 2.5: Performance of LWR on the UAT with various *h* and *k*. The lowest SSEs appear in bold.

2.3.5 Radial Basis Functions (RBFs)

In order to mimic Doya's work (Doya, 2000) in Section 2.4, *Radial Basis Functions* (RBFs) were implemented in the continuous actor-critic. The benefit of RBFs is that they are local approximators, which are made for iterative learning, unlike the memory-based *k*-NN and LWR algorithms.

RBFs are the weighted sum of multiple *kernel functions* as shown in Equation 2.75:

$$F(q) = w_0 + \sum_{i=1}^{N} w_i K\left(q, x_i\right), \tag{2.75}$$

where q is the query point, w_i are tunable parameters, and N is the total number of kernel functions used. Gaussian functions were used for the kernels in our implementation:

$$K(x, x_i) = e^{\frac{-d^2(x, x_i)}{2\sigma^2}}, \tag{2.76}$$

where σ is the standard deviation of the Gaussian, x is the query point, x_i is the center of the i^{th} kernel, and $d(x, x_i)$ is a distance metric. Different distance metrics may be used to account for different units.

The distance metric, d, used was Euclidean distance. For dimensions of the state that are in radians, the distance in radians was found to be the smallest distance, allowing for wraparound from 0 to 2π. Though each weight, w_i, can be changed, the centers of the kernel functions, x_i, are fixed.

In this implementation, the kernel functions were evenly distributed in a grid covering the volume of interest. The derivatives with respect to each weight are needed later for performing gradient descent and are therefore provided in Equations 2.77 and 2.78:

$$\frac{\partial F}{\partial w_0} = 1, \tag{2.77}$$

$$\frac{\partial F}{\partial w_{i \geq 1}} = K(q, x_i). \tag{2.78}$$

RBFs appear to be more robust to large learning rates than ANNs, though the standard deviation of the kernel functions must be well set. For a description of what happens with improper standard deviations, see Figure 20.13 in (Russell and Norvig, 1995). RBFs are known to scale poorly to higher dimensions because the number of kernels required grows exponentially

with the dimension. Thus, RBFs were not selected for implementation on the FES control problem nor the UAT, but rather were only implemented in Section 2.4 when duplicating the work of (Doya, 2000).

2.3.6 Function Approximator Performance Summary

This subsection consolidates the performance descriptions of all the function approximators and compares the results. Table 2.6 provides an overview of the performance of ANNs, FLNs, *k*-NN, and LWR on the UAT. The memory-based methods, *k*-NN and LWR, achieved the smallest SSE, and thus deserve further consideration for use in the continuous actor-critic on the Adaptive RL FES Controller Task. However, these methods are not iterative, and are therefore not suitable for use in their current form. Chapter 3 modifies LWR to create an incremental algorithm for use in the continuous actor-critic.

	Best SSE	**Pros**	**Cons**
ANNs	2,913	Well researched and understood	Slow training
FLNs	4,893	Learns much faster than ANN	Higher error than ANN, possibly due to more local minima
***k*-NN**	1,286	Simple, low error	Not incremental algorithm
LWR	295	Lowest error	Many tunable parameters, not incremental algorithm

Table 2.6: Summary of the performance of ANNs, FLNs, *k*-NN, and LWR on the UAT. RBFs are not included because they were not implemented for the UAT (see Subsection 2.3.5).

Also, notice that the analysis of the results from *k*-NN and LWR may not be indicative of the results for incremental versions thereof because a training phase must be introduced. We selected the number of known points for *k*-NN and LWR to be all of the training points, though the relation of this selection to the training time of incremental variants of *k*-NN and LWR is unknown.

2.4 Pendulum Swing-Up Case Study

In order to better understand the continuous actor-critic (Subsection 2.2.8), the pendulum swing-up task (Doya, 2000) was implemented. The pendulum swing-up task was used by Doya to showcase the continuous actor-critic's learning speed. The pendulum environment is depicted in Figure 2.5.

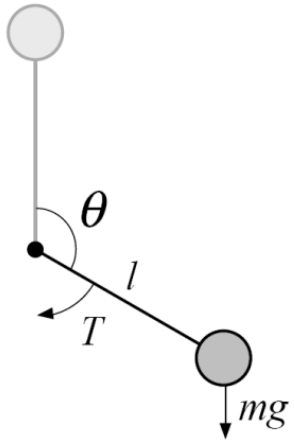


Figure 2.5: Reproduced from (Doya, 2000). The pendulum model consists of an arm with a weight on the end. A limited amount of torque, $-u^{Max} \le T \le u^{Max}$, can be applied to the pendulum. The dynamics follow the equations $\dot{\theta} = \omega$ and $ml^2\dot{w} = -\mu w + mgl\sin\theta + u$, where $m = 1,\ g = 9.8,\ l = 1,\ \mu = .01$ (amount of damping of the pendulum's motion), and $u^{Max} = 5$. Though a time step of .02 seconds was used for updating the actor-critic, motion approximation was done using a forward Euler approximation with a time step of .001 seconds.

The agent must learn to swing the pendulum up to $\theta = 0$. To do this, the agent receives an instantaneous reward of $R(x) = \cos\theta$. Doya used the initial configuration $x(0) = [\theta_0, \omega_0]^T$, where $\omega_0 = 0$, and the initial angle, θ_0, is selected randomly in $[-\pi, \pi]$. For our tests, we used $\omega_0 = 0$ and $\theta_0 = \pi$ to simplify empirical evaluation of each policy.

The actor and critic were both represented using RBFs (Subsection 2.3.5), which are reviewed here. The Gaussian kernels were distributed in a 15×15 grid across the domain $\theta \in [0, 2\pi]$, $\dot{\theta} \in \left[-\frac{5}{4}\pi, \frac{5}{4}\pi\right]$. The kernels used

$$K_i(\theta, \dot{\theta}) = e^{-\frac{(\theta-\theta_i)^2 + (\dot{\theta}-\dot{\theta}_i)^2}{2\sigma^2}}, \tag{2.79}$$

where θ_i is the center of the i^{th} kernel, and $\sigma = .45$. The output for each function approximator is

$$f(\theta, \dot{\theta}) = w_0 + \sum_{i=1}^{n} w_i K_i(\theta, \dot{\theta}), \tag{2.80}$$

where n is the total number of kernels, and the vector w, of length $n+1$ is the set of tunable weights for the function approximator.

The agent's performance is evaluated as the integral of the rewards over a 20-second episode starting with the pendulum hanging straight down. Though Doya terminated episodes when the pendulum completed an entire revolution, our trials were terminated when the agent reached $\theta = 0$ or after the full 20 seconds. For evaluations, explorational noise was not included. The parameters $\tau = 1$, $\tau_N = 1$, $\eta^A = 5$, $\eta^C = 1$, and $\kappa = 1$ were unchanged from (Doya, 2000).

Exploration was decayed differently from Doya's implementation. We set

$$\sigma = \frac{1}{1 + \frac{\text{Episode Number}}{10}}. \tag{2.81}$$

Each setup was run 100 times, and the evaluations were averaged to create Figure 2.6.

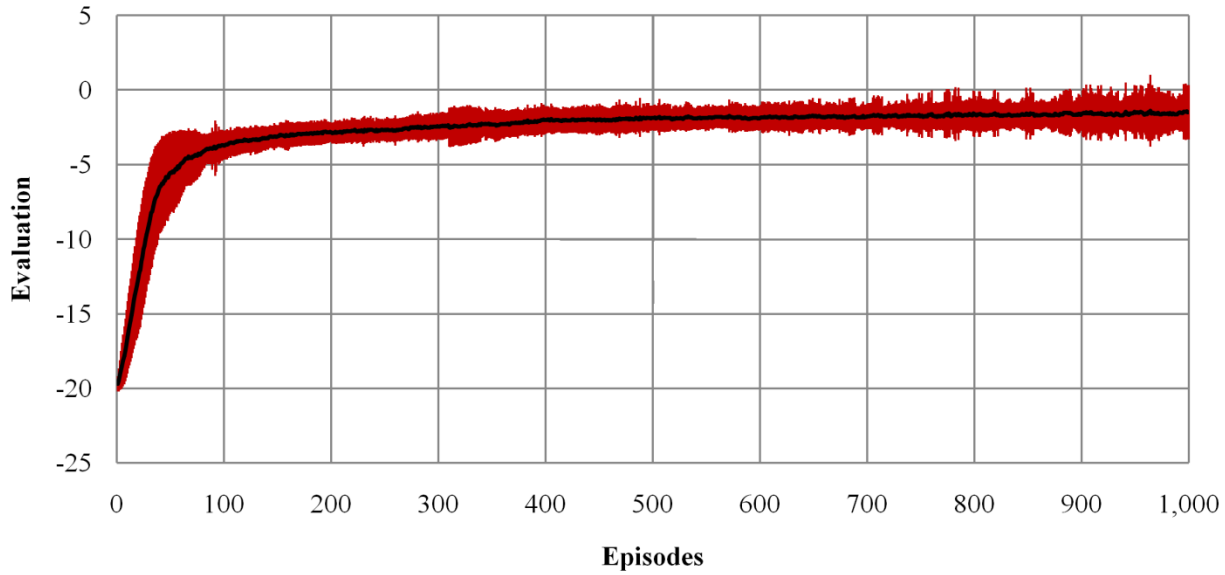


Figure 2.6: Mean performance (*N*=100) of the continuous actor-critic on the pendulum swing up task. Standard deviation error bars are included.

In a typical run after 1,000 episodes of training, the agent swings the pendulum up clockwise to $\theta = 4.3,$ then back counterclockwise to $\theta = 2.2,$ after which it swings clockwise up to $\theta = 0.$ This is a local minimum in policy space because initially swinging to $\theta = 2.2$ would require less time. Additionally, on the final swing up to $\theta = 0,$ the pendulum moves slowly over the final .2 radians. This is likely an artifact of our choice to terminate episodes when the pendulum reaches $\theta = 0$ rather than after a full over-rotation.

To better understand learning, it is useful to view how accurate the critic is during training. To quantify this, after each training episode a separate critic was trained on the actor-

critic's policy for 1,000 episodes using the same parameters as the actor-critic, except with no exploration. The squared difference between the two critics' values was summed over the domain, evaluated every .1 radians from 0 to π for θ and every .2 radians per second from $-5\pi/4$ to $5\pi/4$ for *w*. Figure 2.7 shows the relation between this quantification of critic error and the evaluation. These errors are over the entire state space, though the training episodes only cover the subset of the domain encountered by the current policy.

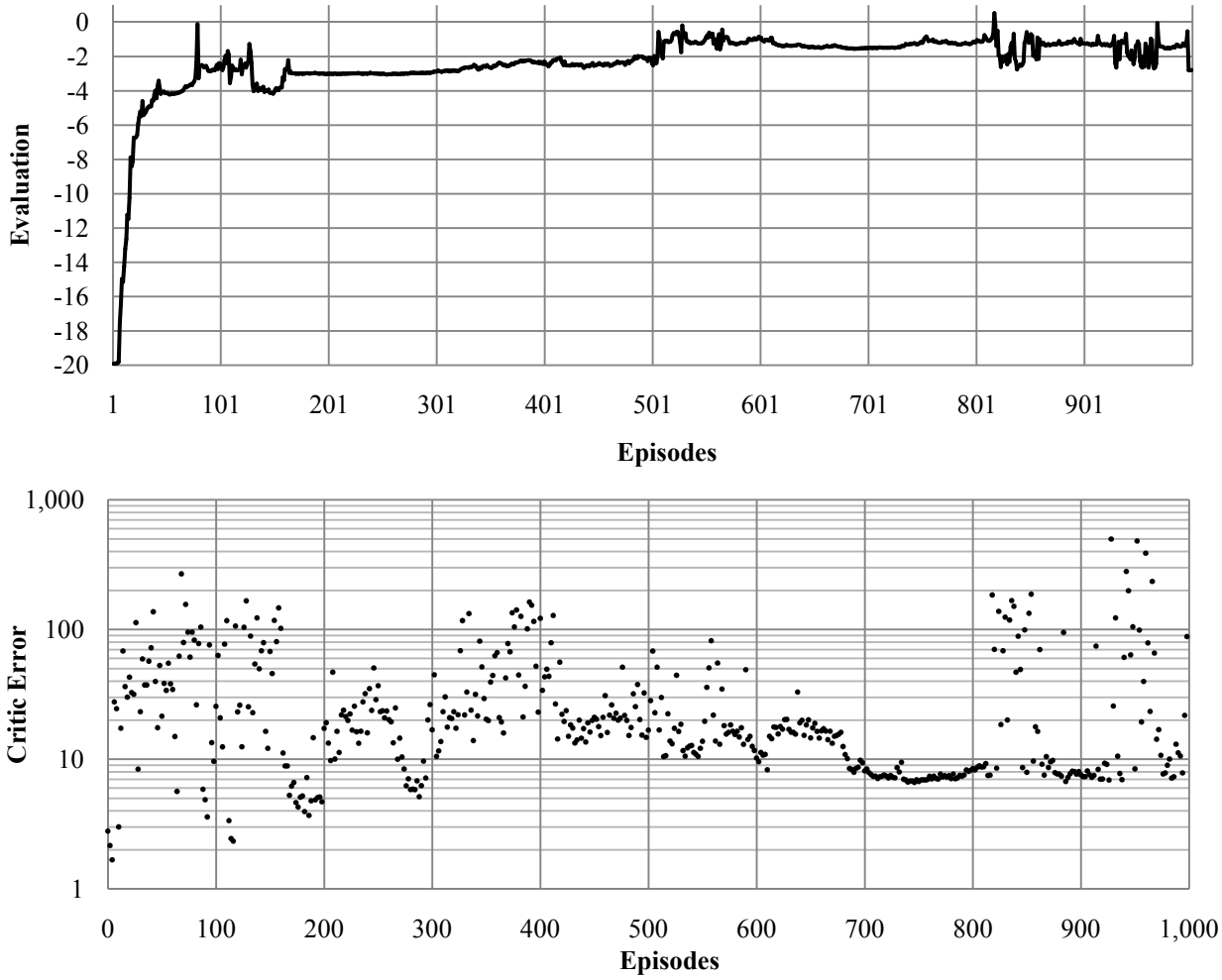


Figure 2.7: Comparison of how accurate the critic is (bottom), to learning (top). Notice the logarithmic scale of the vertical axis in the bottom plot. During the rapid initial learning phase, the critic is less accurate than during the later stages. Some instability is observed both in the critic and the policy after 800 episodes and after 900 episodes.

On the pendulum swing-up task, which utilizes shaping rewards in a manner similar to that of the application of the continuous actor-critic to the DAS1 model in Chapters 5 and 7, the actor-critic achieves rapid initial learning, which occurs even though the critic is not yet accurate. When the critic becomes accurate, learning is slower but more stable. Gullapalli's intuition for why the system learns (Subsection 2.2.10) only applies well when the critic is accurate. Future work should be done to better explain why the system learns when the critic is inaccurate.

CHAPTER 3:

INCREMENTAL LOCALLY WEIGHTED REGRESSION

This chapter has been placed prior to Chapters 4, 5, and 6 in order to preserve the flow of background, methods, and then results. Because the methods provided herein are not utilized until Chapter 7, the reader may opt to skip this chapter and return to it after Chapter 6.

The Locally Weighted Regression (LWR) algorithm (Algorithm 2.1, presented in Subsection 2.3.4), performed the best on the preliminary Utility Approximation Test (UAT) from Chapter 2, approximating the desired function with the least error. It is a promising candidate for use with the continuous actor-critic because it has the potential for local updates, unlike Artificial Neural Networks (ANNs). However, unlike ANNs, which are updated incrementally, the LWR algorithm is *memory-based*. This means that LWR takes a set of known input and output pairs, which it uses to generate future approximations. For RL applications, the input and output pairs are often not known, but rather it is known that the output should be larger or smaller.

In this chapter, we convert the LWR algorithm to an incremental version, dubbed *Incremental Locally Weighted Regression* (ILWR). We then test its performance on simple tasks and real-world tasks, which give a better understanding of how the different ILWR variants, presented in the following sections, relate to each other.

Previous efforts have been made to create an incremental LWR algorithm (Schaal, Atkeson, and Vijayakumar, 2002). These methods attempt to reproduce results if points were all

stored and then slowly forgotten over time. Unlike these prior incremental LWR algorithms, which have required *forgetting factors* (Vijayakumar, D'Souza, and Schaal, 2005), ILWR does not explicitly forget points. ILWR is also fundamentally different due its placement of regression points off the surface to be approximated (cf. Figure 3.3).

Previous incremental LWR algorithms, sometimes called *locally weighted learning* (LWL) methods, also tend to be kernel centric. Such methods, as well as Radial Basis Functions (RBFs; Subsection 2.3.5) are known to scale poorly to high-dimensional problems. If the Adaptive RL FES Controller Task is to be scaled up to include more muscles and degrees of freedom, we desire a function approximator that scales well for our problem. LWR is known to have several advantages over kernel based models (Atkeson, Moore, and Schaal, 1996), including its ability to perfectly represent a planar function, such as the PD control law of Equation 2.2. In our incremental adaptation of LWR presented in this chapter, we will preserve this planar local model rather than switching to kernels, with the hope that it will scale better to future problems.

The memory-based LWR algorithm can be converted to an incremental method using the gradient descent rule provided in Equation 3.1. As with other function approximators, such as ANNs, the equations for LWR are continuously differentiable with respect to its parameters, making it suitable for use with the gradient descent rule. LWR will be given a fixed number of points, initialized randomly or using prior knowledge (such as placing higher initial point densities in areas with large $\partial^2 y / \partial x^2$), which will be incrementally modified to better approximate the desired function. At each step, a query point, x_q, y_q is observed. LWR generates an approximation, $\hat{y}_q$, then modifies the weights to decrease the error term.

The points stored initially and updated during training will be referred to as *knowledge points*, while the points used to train the knowledge points will be called *training points*.

According to (Mitchell, 1997), the gradient descent rule, in terms of the parameters or weights, w, for a function approximator, is

$$w \leftarrow w + \Delta w, \tag{3.1}$$

$$\Delta w = -\eta \nabla E(w), \tag{3.2}$$

where η is a learning rate and the error term, $E(w)$ is defined as

$$E(w) \equiv \frac{1}{2} \sum_{i=1}^{d_o} \left(y_{q,i} - \hat{y}_{q,i} \right)^2, \tag{3.3}$$

where d_o is the dimension of the output, $\hat{y}$ is ILWR's approximation for the current query, and $y_{i,j}$ is the j^{th} output of the i^{th} training point, y_i. We will apply this to the LWR algorithm, with the knowledge points treated as the weights, w. This choice of error term simplifies the expansion of the gradient descent rule, defined for each weight as

$$w_i \leftarrow w_i - \eta \cdot \sum_{\alpha=1}^{d_o} \left[\left(\hat{y}_{q,\alpha} - y_{q,\alpha} \right) \frac{\partial \hat{y}_{q,\alpha}}{\partial w_i} \right]. \tag{3.4}$$

A derivation of Equation 3.4 is provided in Appendix D. Because we wish to train after each point, the error term in Equation 3.3 does not include a sum over all training points observed. This form of gradient descent is often referred to as incremental gradient descent or stochastic gradient descent. For further information on the differences between true gradient descent and its stochastic approximation, see (Mitchell, 1997).

Next, we must consider what the weights, w, represent. One approach is to consider the outputs of the knowledge points, y_i, the weights. For approximating a function $y = f(x)$ where $x, y \in \mathbb{R}$, such as that depicted in Figure 3.1, this means fixing the x-coordinates of knowledge points, but allowing the y-coordinates to vary. This will be referred to as *Static Input Incremental Locally Weighted Regression* (SI-ILWR).

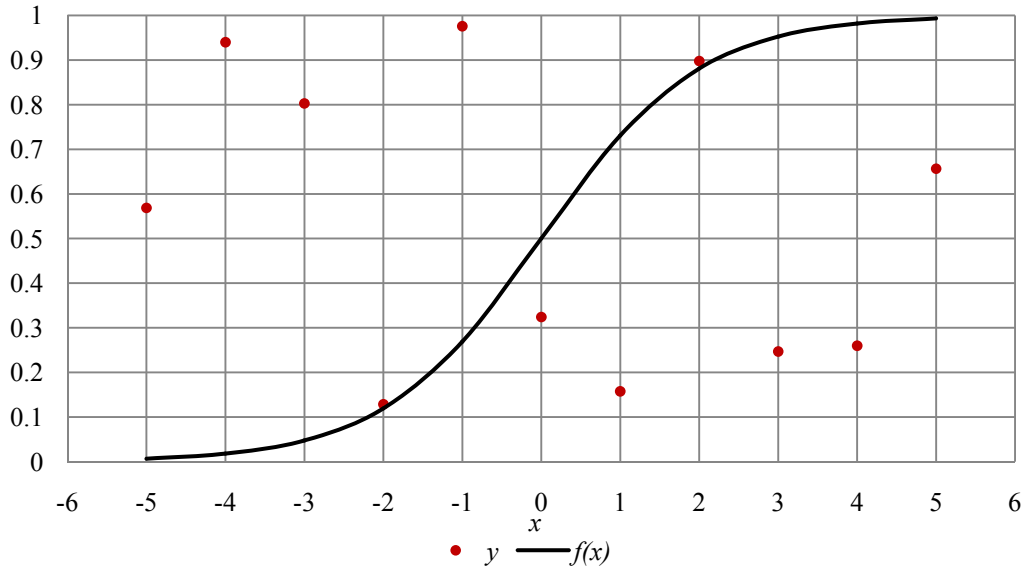


Figure 3.1: Example ILWR problem, where y denotes the initial knowledge points, and $f(x)$ denotes the desired function.

Another option is to vary both the inputs and the outputs. This allows gradient descent to choose the density of knowledge points over the input space. In Figure 3.1 this means allowing points to move along both the x- and y-axes. This method will be referred to as *Dynamic Input Incremental Locally Weighted Regression* (DI-ILWR). One expects DI-ILWR to perform better than SI-ILWR because it can move points to more interesting regions of the domain. In this method, different learning rates are used in Equation 3.4 for updates of the inputs and outputs of the knowledge points. SI-ILWR and SO-ILWR, described in the following paragraph, are

specific cases of DI-ILWR, with one learning rate set to zero. In other chapters, references to ILWR therefore refer to DI-ILWR.

The third option, fixing the outputs and allowing the inputs to vary, is not practical, but serves as an interesting example. In the example provided in Figure 3.1, this means fixing the *y*-values of each knowledge point, but moving them along the *x*-axis. This method will be referred to as *Static Output Incremental Locally Weighted Regression* (SO-ILWR).

When implementing Equation 3.4, SI-ILWR requires only $\partial\hat{y}_{q,k} / \partial y_{i,j}$, SO-ILWR requires only $\partial\hat{y}_{q,k} / \partial x_{i,j}$, and DI-ILWR requires both. Notice that $\partial\hat{y}_{q,k} / \partial x_{i,j}$ cannot be extracted from $\partial\hat{y}_{q,k} / \partial \mathbf{X}$ because it does not account for the change in the approximation due to the change in $\mathbf{W}$ when $x_{i,j}$ changes.

The derivative of each output with respect to each training point's output is

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial y_{i,j}} = \begin{cases} 0, & \text{when } j \neq k, \\ \left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\right]_{d_i+1,i}, & \text{otherwise,} \end{cases} \tag{3.5}$$

where d_i is the dimension of the inputs. A derivation of this equation is provided in Appendix A. The derivative of each output with respect to each training point's input is

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial x_{i,j}} = \left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\left[\mathbf{X}^T\frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{y} + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{y}\right] - \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\left[\mathbf{X}^T\left(\mathbf{W}\frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X}\right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{X}\right]\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\mathbf{y}\right]_{d_i+1,k} \tag{3.6}$$

where $\partial\mathbf{X} / \partial x_{i,j}$ is a $p \times d_i + 1$ matrix with all entries zero except $\mathbf{X}_{i,j} = 1$, and where $\partial\mathbf{W} / \partial x_{i,j}$ is a $p \times p$ matrix with all entries zero except the entry at *i,i*, which is

$$\left[\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right]_{i,i} = \mathbf{W}_{i,i}\mathbf{D}_{j,j}\left(x_{q,j} - x_{i,j}\right). \tag{3.7}$$

Equation 3.7 assumes **D** is fixed, or not a function of $x_{i,j}$. This assumption is discussed in the following paragraph. Derivations of Equations 3.6 and 3.7 are provided in Appendix B. Appendix C shows how to efficiently implement Equation 3.6 by taking advantage of the structures of $\partial\mathbf{X} / \partial x_{i,j}$, $\partial\mathbf{W} / \partial x_{i,j}$, and **W**.

One key feature of these learning algorithms is that all changes are effectively local. As the distance from the change increases, the influence of the change decreases exponentially. If **D** were not fixed, then changes to a knowledge point's inputs would change the weighting of each dimension, which is a global change. We desire local updates, so **D** is fixed. It is our untested belief that the gradient descent algorithm will learn to work with the dimension weightings it is provided as long as the values are reasonable. If one intends to implement these algorithms without fixing **D**, the derivation of Equation 3.7 deviates at Equation B23.

3.1 Experiments

The following four subsections show results for ILWR learning on simple problems. The results are useful for understanding the inner workings of ILWR, evaluating performance relative to ANNs, as well as justifying further investigation of ILWR for FES control of a human arm.

3.1.1 Sigmoid Environment

In the first environment, dubbed the *Sigmoid Environment*, the agent must learn to approximate the sigmoid function

$$S(x) = \frac{1}{1+e^{-x}}. \tag{3.8}$$

This test is designed not to judge each method's learning speed, but instead to judge how accurate of an approximation can eventually be achieved with small learning rates and as many training points as are necessary for convergence. Only five knowledge points were used, each initialized randomly throughout the domain, $x \in [-10,10]$, and the range, $y \in (0,1)$. Evaluations were computed by taking the normalized (divided by 21) sum of the squared error over the domain, sampled at every unit, as in Equation 3.9:

$$\text{Evaluation} = \frac{1}{21}\sum_{x=-10}^{10}\left(\frac{1}{1+e^{-x}} - \hat{y}(x)\right)^2, \tag{3.9}$$

where $\hat{y}(x)$ is ILWR's approximation for the point x. The knowledge points are randomly placed over the domain and range. A typical initial evaluation with this setup is .243.

All three ILWR variants were trained using 2,000,000 randomly generated training points, far more than were necessary to achieve numerical convergence. In all tests, **D** was fixed as $\mathbf{D} = diag(.05)$. The best evaluation achieved by SI-ILWR was .004, with a learning rate of .005. The best evaluation achieved by DI-ILWR was .0000875, with learning rates of .01 for outputs, and .015 for inputs. The best evaluation achieved by SO-ILWR was .0075, with learning rate .001. Figures 3.2, 3.3, and 3.4 present the results when using SI-ILWR, DI-ILWR, and SO-ILWR, respectively.

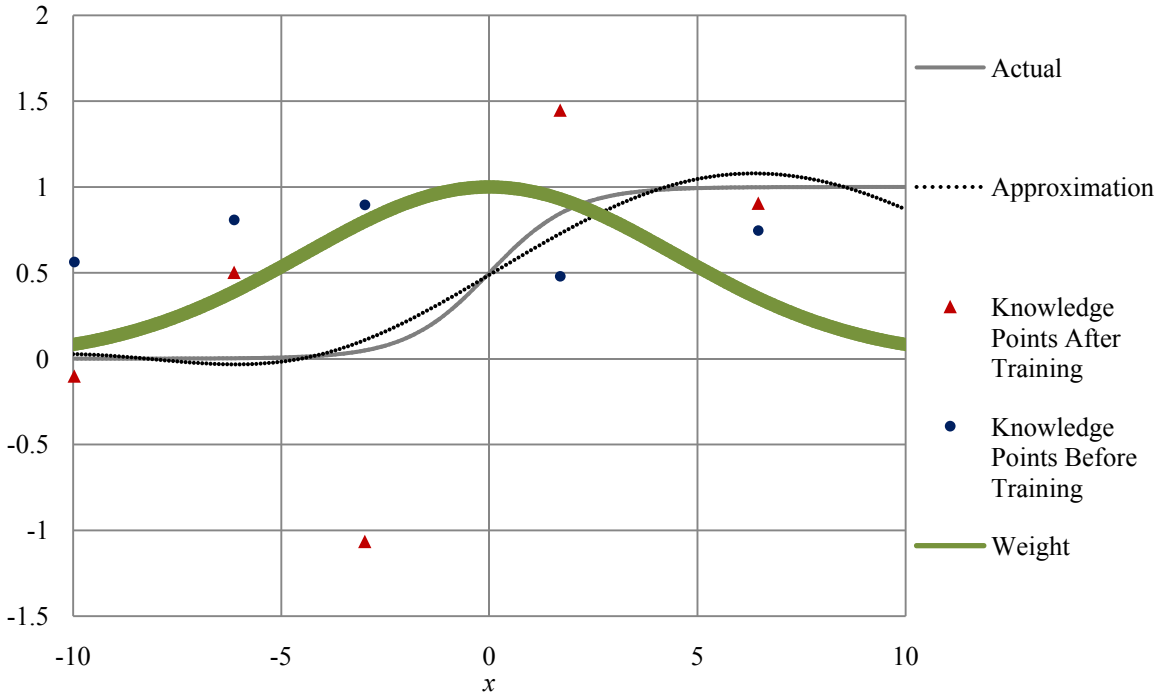


Figure 3.2: Final function approximated by SI-ILWR after 2,000,000 random training points, with a learning rate of .005. The final evaluation (Equation 3.9) is .004. The thick green line represents what the weights would be for knowledge points with various x, if the query point was at zero. This weighting, determined by **D**, is identical for the remainder of this subsection.

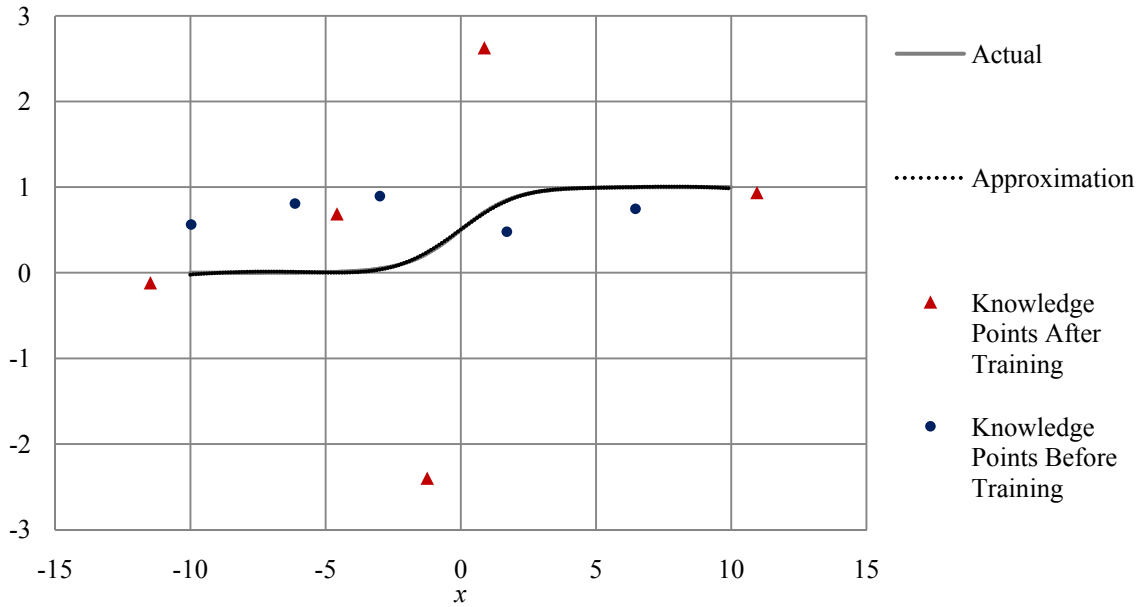


Figure 3.3: Final function approximated by DI-ILWR after 2,000,000 random training points, with a learning rate of .01 for outputs and .015 for inputs. The plot for DI-ILWR's approximation obscures the plot of the actual sigmoid over most of the domain. The final evaluation (Equation 3.9) is .0000875.

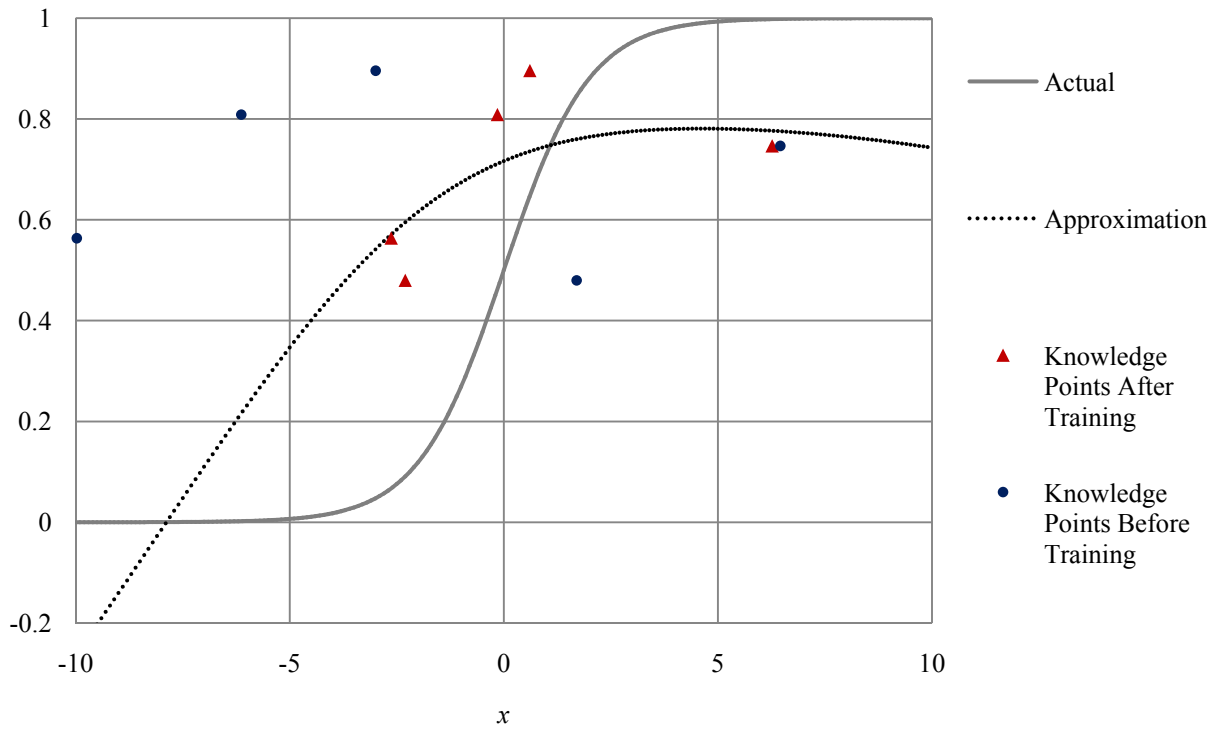


Figure 3.4: Final function approximated by SO-ILWR after 2,000,000 random training points, with a learning rate of .001. The final evaluation (Equation 3.9) is .0075.

Notice that, unlike the LWR algorithm, the knowledge points are not necessarily positioned on the target surface. Instead, they are positioned at the locations that result in the most accurate approximation of the surface. As the density of knowledge points increases and the locality of **D** increases, one would expect the points to converge to the actual surface.

As expected, DI-ILWR performs best, accurately approximating the sigmoid. SO-ILWR performs the worst, though its performance is remarkably good considering that the *y*-values of the points could not be changed, and none of the random initial outputs for the knowledge points in Figure 3.4 are less than .4. Learning curves were not provided for this learning task because we were interested in final error and gaining a basic understanding of the three variants of ILWR, not learning speed. Learning curves will be provided for the problem in Subsection 3.1.2.

3.1.2 Double Environment

The second environment, dubbed the *Double Environment*, tests the speed of learning for approximating a function with two inputs and two outputs. This is equivalent to approximating two separate functions given only one set of knowledge points. Tests were run using ten, and then 100 knowledge points. The desired function is defined as

$$f(x) = \left[(x_1 - 5)^2 + (x_2 - 5)^2, \quad (x_1 + 5)^2 + (x_2 + 5)^2 \right]. \tag{3.10}$$

This function was chosen because the two outputs have different points of maximal $\partial^2 y / \partial x^2$, which are expected to be areas where higher knowledge point densities are most beneficial. The domain and range used were $x_1, x_2 \in [-10, 10]$ and $y_1, y_2 \in [0, 450]$. Evaluations were done by taking the average squared error over the domain, sampled every square unit. A typical initial evaluation is approximately 65,000. A smaller evaluation is better.

For the first tests, the knowledge base of ten points was initialized randomly over the domain and range. $\mathbf{D}$ was fixed as $\mathbf{D} = diag(.05, .05)$. Learning rates were manually optimized to achieve evaluations less than 1,500 as rapidly as possible. Of all the learning rates tested, those that achieved an evaluation less than 1,500 after the fewest training points are reported in Table 3.1. Results are compared in Table 3.1 to those of an ANN with ten neurons in its only hidden layer, trained using the error backpropagation algorithm (Russell and Norvig, 1995). The choice of ten neurons results in 52 tunable weights in the ANN, which is comparable to the 40 tunable weights in DI-ILWR with ten knowledge points. SO-ILWR is not included in Table 3.1 because the best evaluation it achieved was 50,000, and it has a tendency to diverge even with small learning rates.

	Input Learning Rate	Output Learning Rate	Episodes to Evaluation <1,500	Minimum Learning Rate of Divergence	Tunable Parameters
SI-ILWR	NA	.2	2,500	.6	20
DI-ILWR	.00001	1	128	NA	40
ANN	NA	.0002	80,000	.15	52

Table 3.1: Relative learning speeds of each algorithm, optimal learning rates for rapidly achieving an evaluation less than 1,500, and maximum learning rates before divergence occurs. For the ANN, learning rates between .0002 and .15 converge to worse evaluations than 1,500. DI-ILWR does not have a learning rate for divergence listed because its learning rate is two-dimensional.

Figure 3.5 shows the learning curves for the SI-ILWR and DI-ILWR setups in Table 3.1. DI-ILWR performs an order of magnitude better than SI-ILWR with this setup, suggesting that, when using few knowledge points, DI-ILWR is capable of achieving significantly lower errors than SI-ILWR.

These results suggest that SI-ILWR and DI-ILWR are capable of approximating this function to higher accuracy than a small ANN of comparable size for a fixed, small number of updates. However, a larger ANN can still represent the function with similar accuracy to ILWR, though it requires more updates. Using a small learning rate for 7,000,000 updates, an ANN with 20 neurons in the first hidden layer and 20 neurons in the second hidden layer achieved an evaluation of 50. Both SI-ILWR and DI-ILWR were found to be empirically stable in the long-term for at least 200,000 training points.

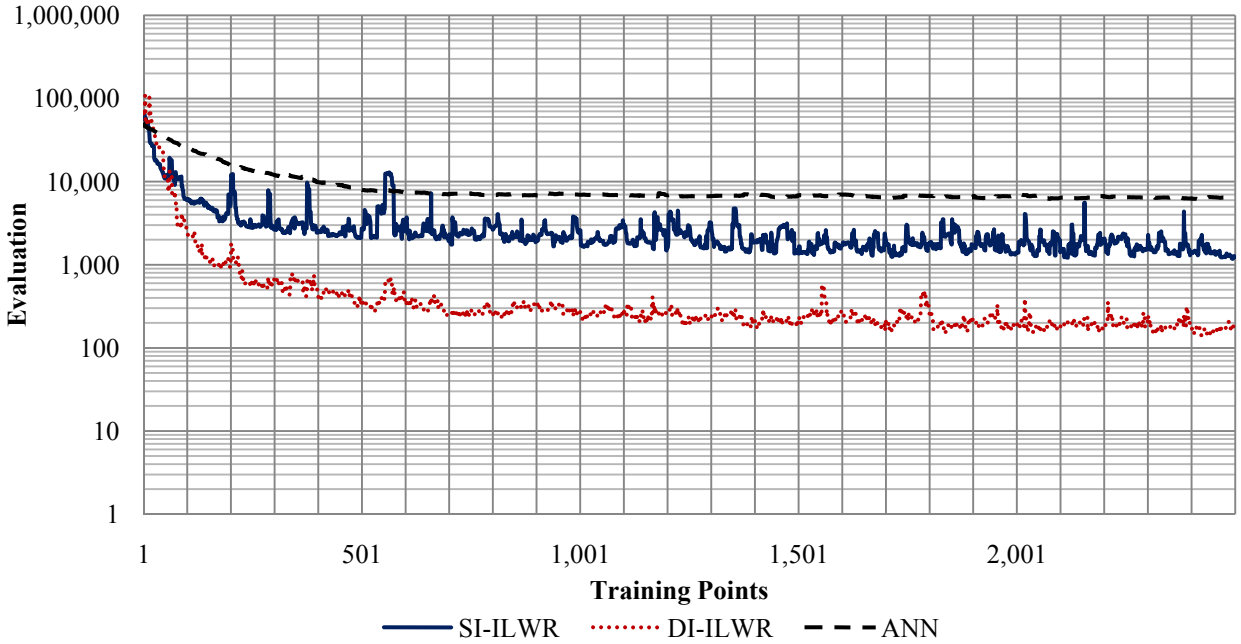


Figure 3.5: Learning curves for SI-ILWR, DI-ILWR, and an ANN on the Double Environment using the parameters from Table 3.1. SI-ILWR and DI-ILWR both used 10 randomly initialized knowledge points. Notice the logarithmic vertical axis.

For the second set of tests, the knowledge base was increased to 100 points and was initialized in a *Sukharev Grid* (Sukharev, 1971), covering the domain with a uniform density. The largest learning rate for which SI-ILWR was stable was found, through manual search, to be ten. With this learning rate, its evaluation breaks 1,500 after 39 training points. This same setup, using randomly initialized knowledge points rather than the Sukharev Grid, requires 482 training points. This suggests that initializing the points in a Sukharev Grid improves performance significantly. For DI-ILWR, the largest stable input learning rate found was .001, and the largest output learning rate was ten. DI-ILWR also required 39 training points before its evaluation reached 1,500.

The larger learning rates are required because, with more points close to each query point, the derivative of the output with respect to a single point is generally smaller. Figures 3.6

and 3.7 show the evaluations over time of these two methods. The learning curves are more similar than those of Figure 3.5, suggesting that, as the number of knowledge points increases to the point where the domain is well covered, the benefits of DI-ILWR over SI-ILWR are marginal.

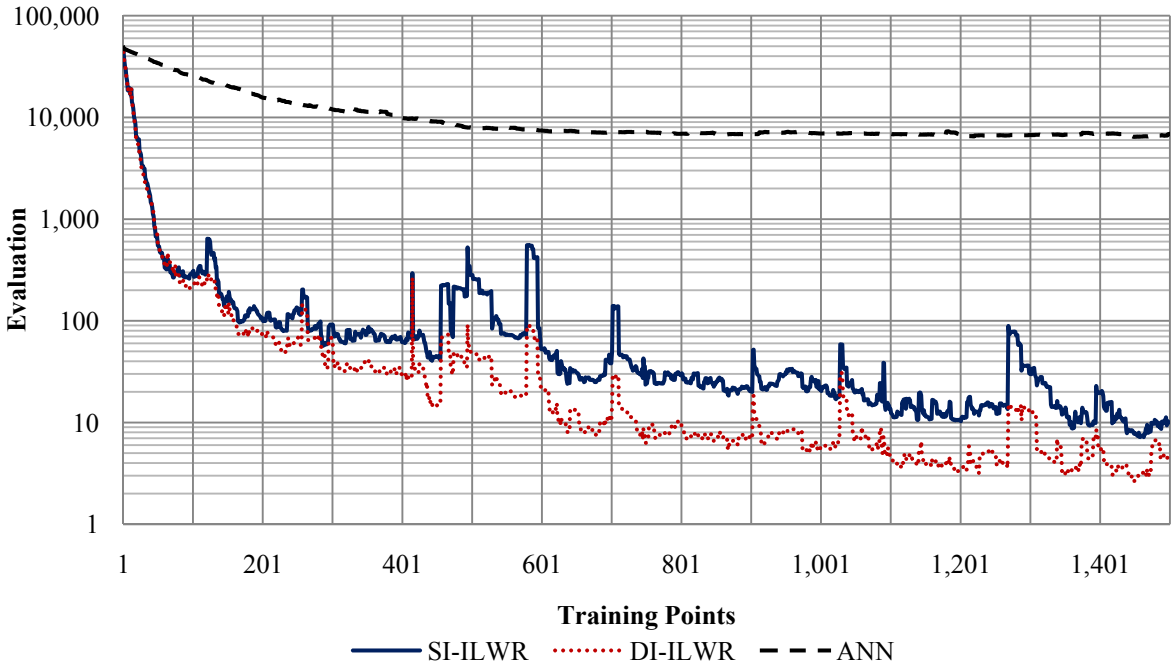


Figure 3.6: Evaluations over time for SI-ILWR, DI-ILWR, and an ANN in the Double Environment. The LWR algorithms have 100 knowledge points, an input learning rate of .001, and an output learning rate of 10. The ANN is the one found for Table 3.1, which has ten neurons in its hidden layer and a backpropagation learning rate of .002. The graph spans 1,500 training points.

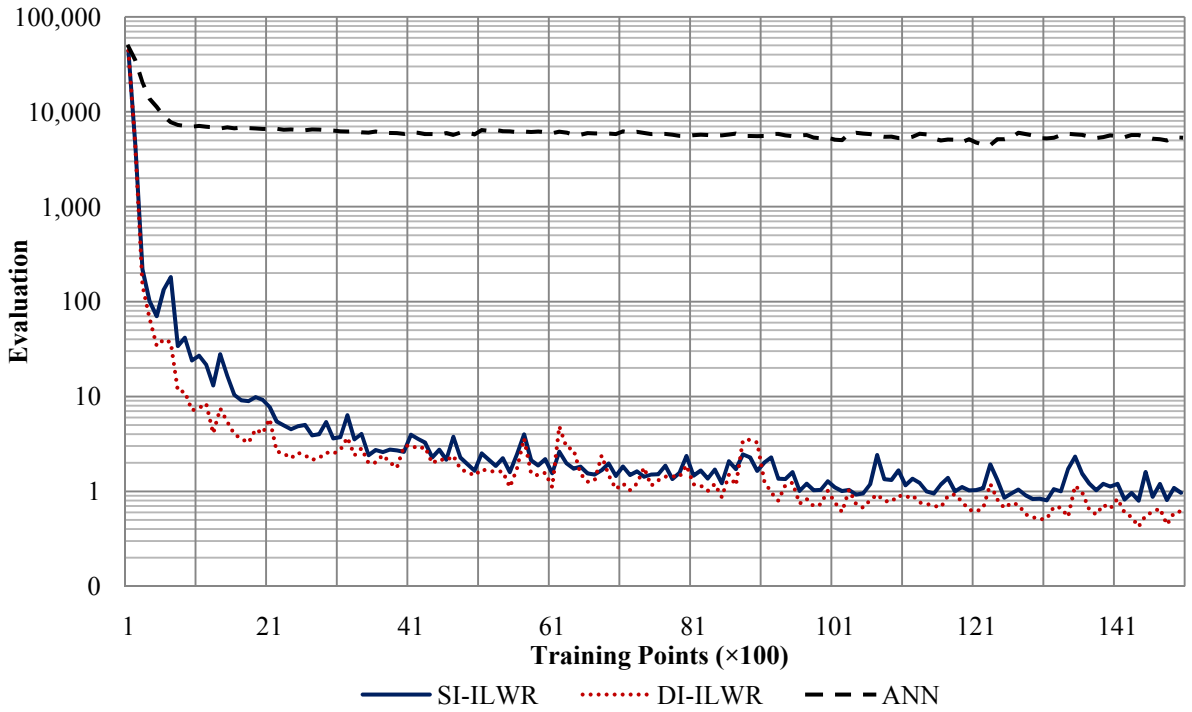


Figure 3.7: Long-term plot with same setup from Figure 3.6, except this graph spans 15,000 training points (notice that the horizontal axis is scaled by a factor of 100).

Figure 3.8 illustrates the initial and final knowledge point configurations of SI-ILWR and DI-ILWR when using the parameters from Table 3.1 for 15,000 training points. Subsection 3.1.3 further investigates the movement of knowledge points when using DI-ILWR.

These results are encouraging not only because the learning speeds and converged errors are superior to those of fully connected feed-forward ANNs with similar numbers of tunable parameters, but also because the points seem to cluster around the most interesting parts of the domain, though this is not completely clear when the domain is densely covered as in Figure 3.8. Therefore, the following subsection will further investigate whether the knowledge points move to the most interesting regions of the domain.

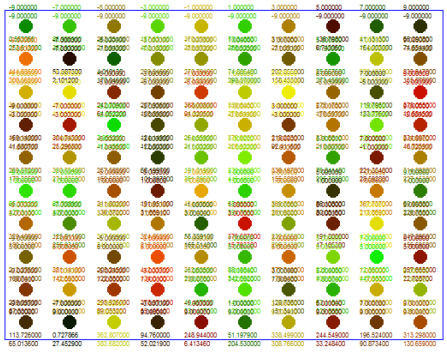
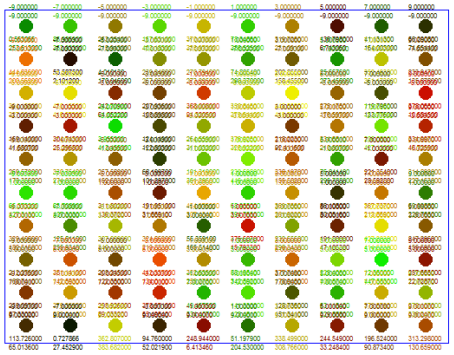
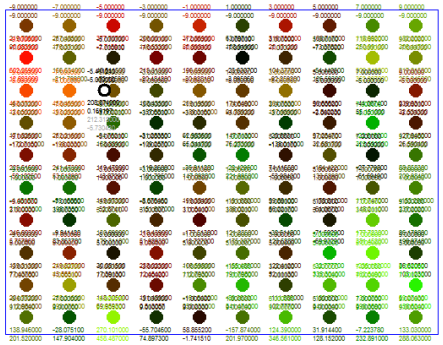
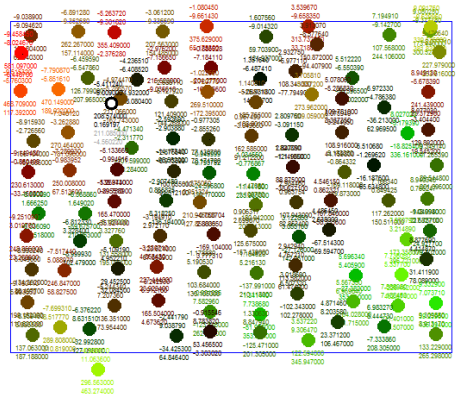
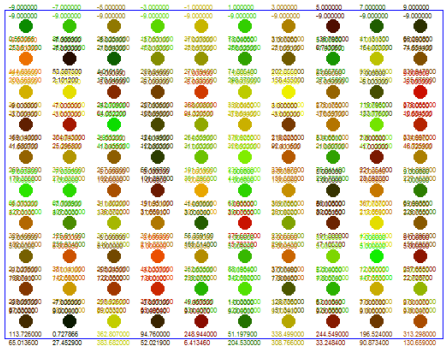
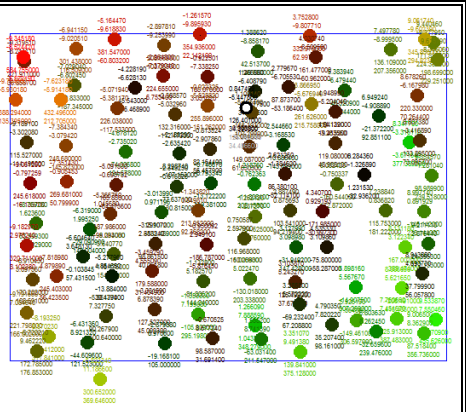
	SI-ILWR	DI-ILWR
Initial Distribution	 Evaluation = 43,868.3	 Evaluation = 43,868.3
After 100 Iterations	 Evaluation = 309.976	 Evaluation = 227.706
After 15,000 Iterations	 Evaluation = .776443	 Evaluation = .463469

Figure 3.8: Illustration of the movement of knowledge points during training using SI-ILWR and DI-ILWR. In all images, the origin is in the center of the blue box, which spans from –10 to 10. The more green a point is, the larger its value in the first output dimension. Similarly, the more red a point is, the larger its value in the second output dimension. A point with little red and green is displayed as being dark. Above each point are the input coordinates, and below are the output coordinates. The white circle with a black outline denotes the location of the latest training point.

3.1.3 FitzHugh-Nagumo Approximation (Accuracy)

This subsection is inspired by (Wedge, 2004), which focuses on using function approximators to approximate the *FitzHugh-Nagumo equations* (Izhikevich, 2007) for modeling a cell. V represents the electrical potential of a cell and W is a variable relating to sodium and potassium gating. Figure 3.6 of (Wedge, 2004) shows the values of V and W after one time unit of simulation with a small time step, given various initial V and W. We will focus on the values of V. Figure 3.13 of (Wedge, 2004) shows that, when points are distributed with a higher density around interesting regions, smaller errors can be achieved when using LWR. We wish to see whether DI-ILWR will find such a distribution. The FitzHugh-Nagumo approximation problem was chosen because of its real-world application and complexity.

The FitzHugh-Nagumo equations are, reproduced from (Wedge, 2004),

$$\frac{\partial V}{\partial t} = \frac{1}{\varepsilon}\left(V - \frac{V^3}{3} - W\right), \tag{3.11}$$

$$\frac{\partial W}{\partial t} = \varepsilon\left(V - \gamma W + \beta\right), \tag{3.12}$$

where $\varepsilon = .2,\ \beta = .7,$ and $\gamma = .8.$ For further description of these parameters, see (Wedge, 2004). Simulations were performed with a forward Euler approximation, with time step $\Delta t = .0001.$ The domain of initial conditions tested is $V \in [-2.1, 1.9]$ and $W \in [-.7, 1.0]$. The desired function is shown in Figure 3.9.

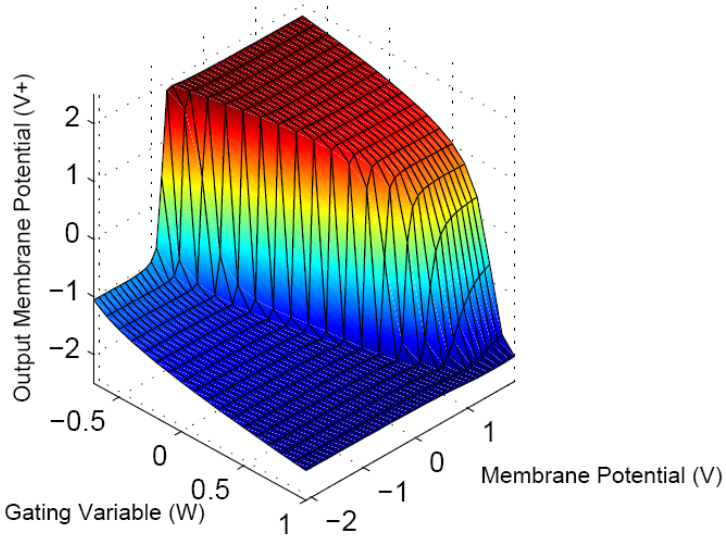


Figure 3.9: Result of simulating FitzHugh-Nagumo equations for one time unit, starting with initial conditions V, W. Image reproduced from (Wedge, 2004, Figure 3.6).

The first test in this environment, dubbed the *FitzHugh-Nagumo Accurate Approximation Task*, compares the final approximations of SI-ILWR and DI-ILWR after training with small learning rates for as many training points as are needed for convergence. The goal of this test is to show that DI-ILWR can reorganize the points in ways that result in smaller errors than is possible with SI-ILWR. The expectation is that the points will be most dense around the areas of the function with the largest second derivative. This test will also show whether, for this problem, the two methods are empirically stable in the long-term.

Learning rates were manually determined to achieve minimal converged error. For SI-ILWR, the output learning rate was 1. For DI-ILWR the output learning rate was 1 and the input learning rate .01. Both used a 10×10 grid of knowledge points, initialized randomly between –2 and 2. $\mathbf{D}$ was fixed as $\mathbf{D} = diag(5,5)$ for this subsection as well as the following, Subsection 3.1.4. Evaluations were computed as the normalized (divided by 10,000) sum of the squared error. Evaluation points were sampled in a 100×100 grid over the domain. Because the

evaluation of points on the surface is computationally expensive, 500,000 random points were pre-evaluated for training. Training points were then sampled randomly from these 500,000 points. Figure 3.10 shows the performance of both algorithms over 1,000,000 training iterations.

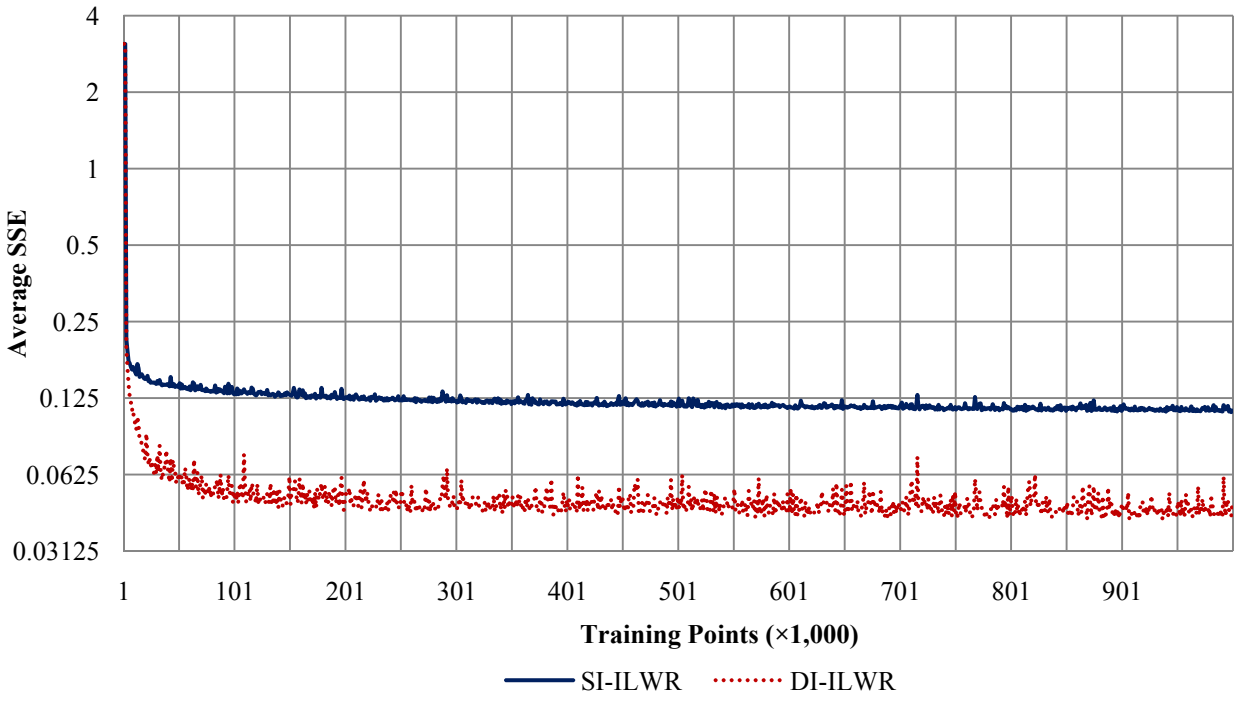


Figure 3.10: Performance of SI-ILWR and DI-ILWR on the FitzHugh-Nagumo Accurate Approximation Task. Notice the logarithmic scale of the vertical axis.

After 1,000,000 training points, DI-ILWR has reached a smaller error than SI-ILWR. Its knowledge points have been moved about the domain to increase the point density in areas with a high second derivative, as shown in Figure 3.11.

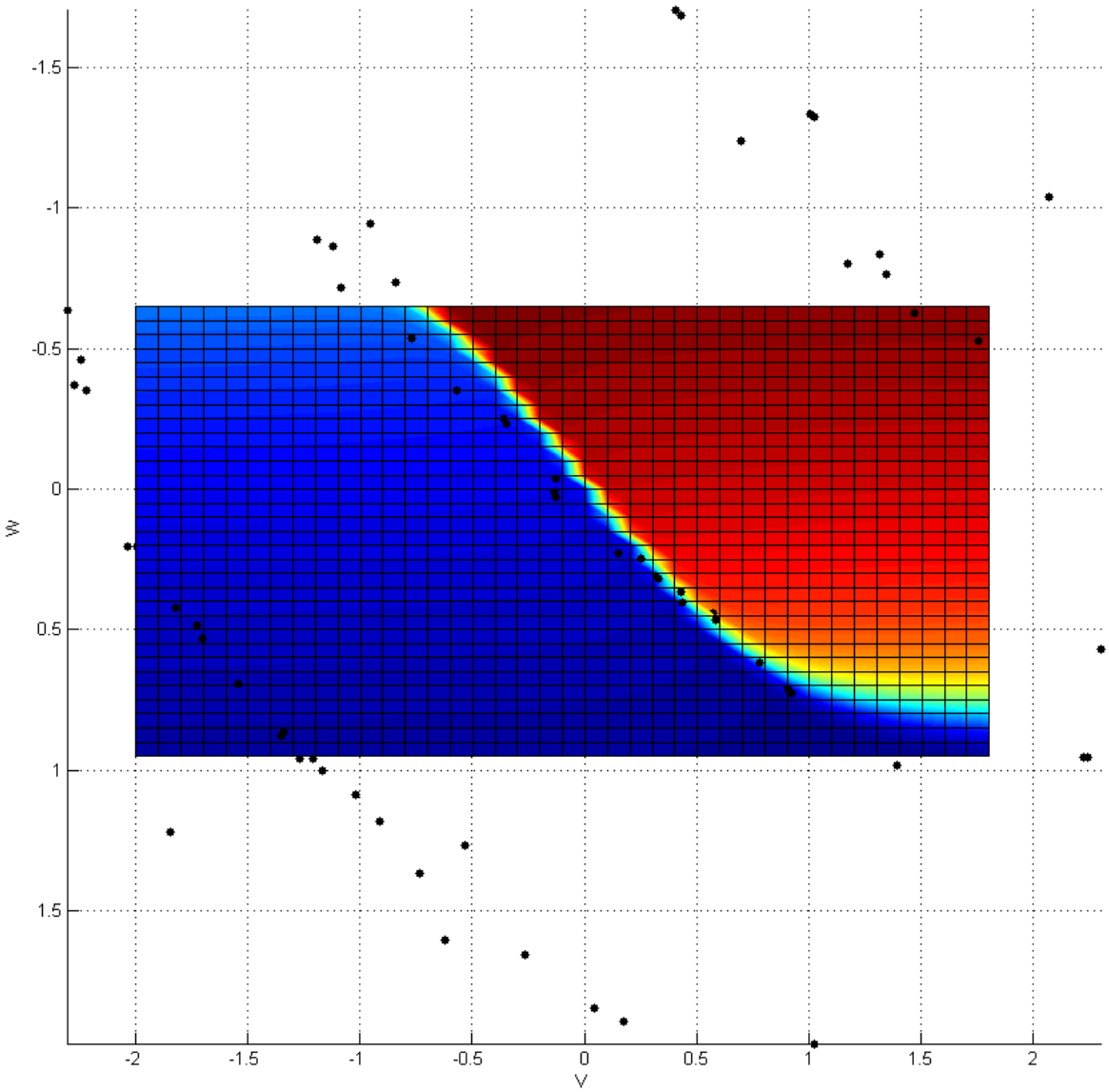


Figure 3.11: DI-ILWR knowledge point locations after 1,000,000 training points. Notice the high point density along the cliff from $V,W=(-.5,-.5)$ to $V,W=(1,.8)$, where the function has the largest second derivative. The remaining points have moved farther away from the cliff, creating a relatively flat surface away from the cliff. The "heat map" covers the domain over which the approximation is evaluated: $V\in[-2.1,1.9]$ and $W\in[-.7,1.0]$.

Figure 3.12 depicts the final surfaces generated by SI-ILWR and DI-ILWR. Notice that DI-ILWR has achieved better accuracy both at the cliff and over the relatively linear regions. Figure 3.13 depicts the error in the DI-ILWR and SI-ILWR approximations.

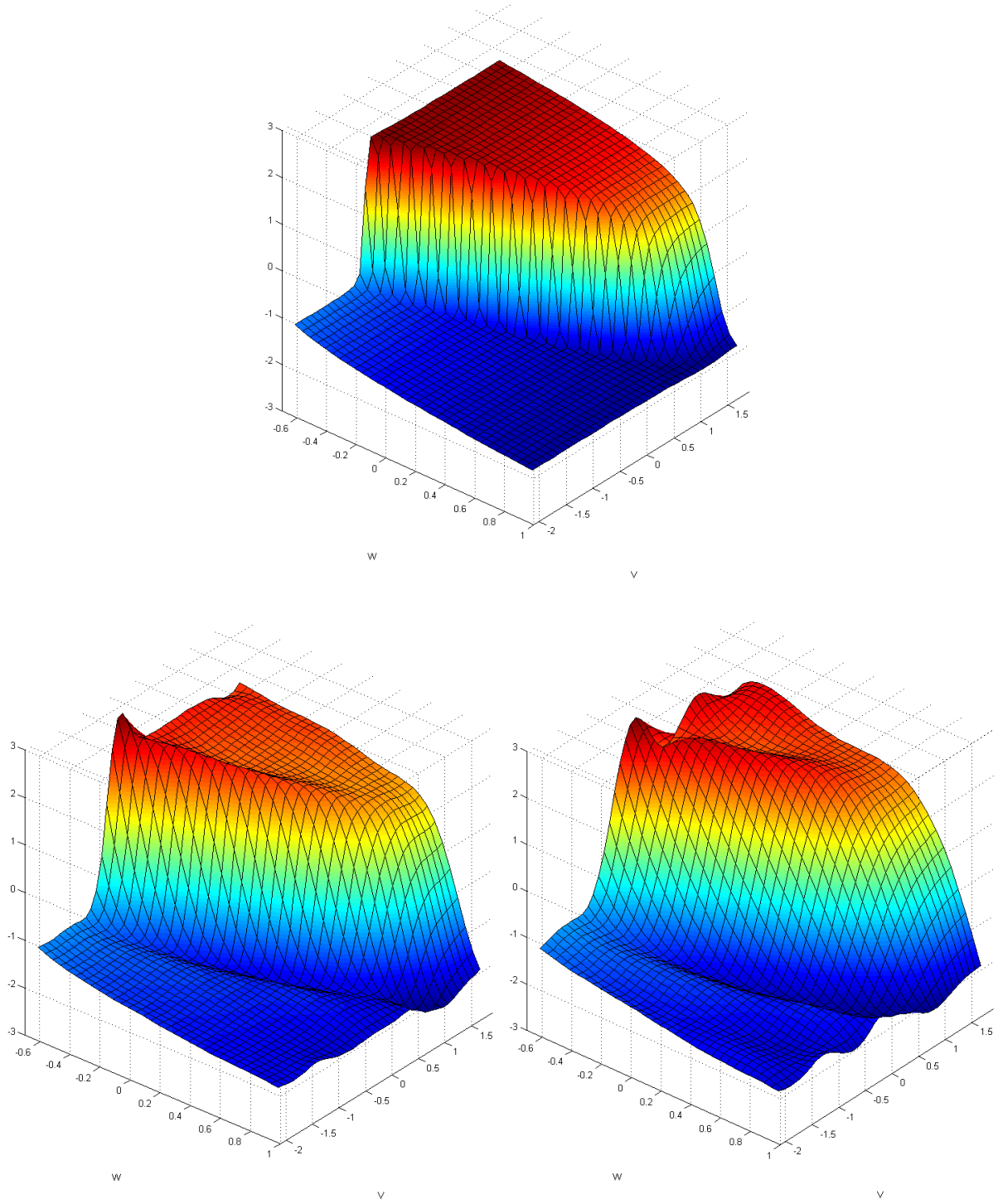


Figure 3.12: The target surface (top), reproduced to mimic Figure 3.9; the surfaces approximated by DI-ILWR (bottom left) and SI-ILWR (bottom right) after the run from Figure 3.10. Notice that, around the cliff, as well as over the linear regions of the target function, DI-ILWR performs significantly better than SI-ILWR.

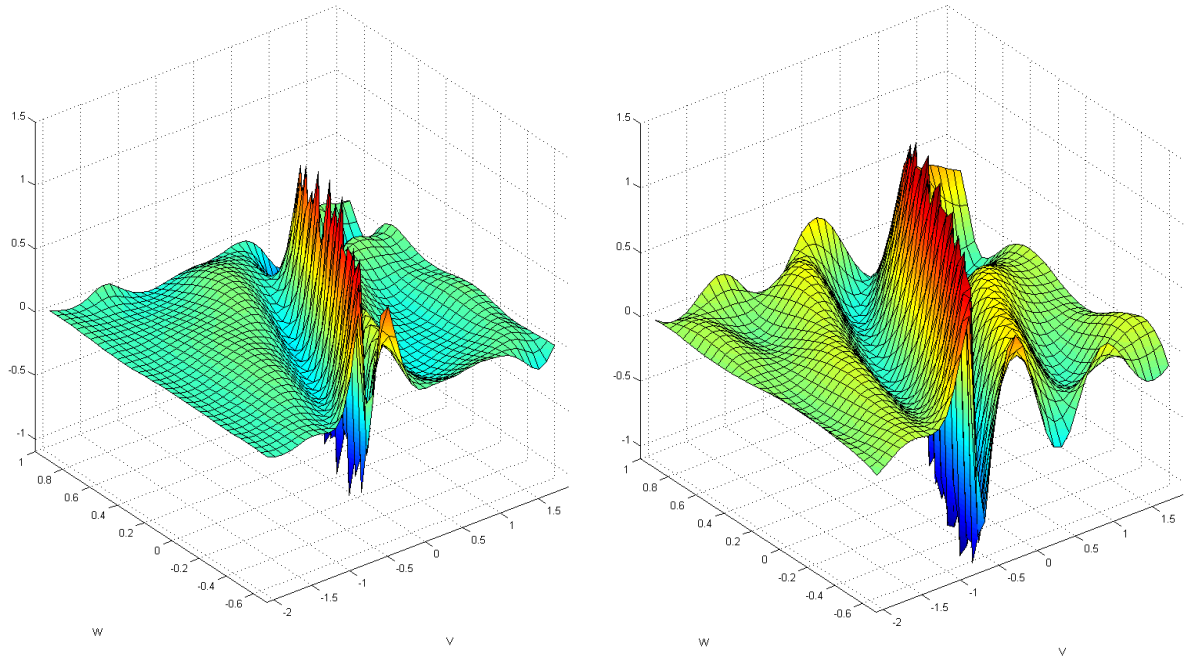


Figure 3.13: The difference between the DI-ILWR approximation and the target function (left), and the difference between the SI-ILWR approximation and the target function (right). Notice that the *W*-axis has been flipped to provide a better view of the surface. A positive value denotes that the target value was smaller than the approximation. Notice that the scales are same on both plots, making the larger error in SI-ILWR clearly visible.

From these results, we conclude that, on the FitzHugh-Nagumo Accurate Approximation Task, DI-ILWR does move the points to more interesting regions, as desired. This results in lower errors than SI-ILWR is capable of achieving.

3.1.4 FitzHugh-Nagumo Approximation (Learning Speed)

The second test in the FitzHugh-Nagumo environment, dubbed the *FitzHugh-Nagumo Rapid Approximation Task*, compares the speed of learning with DI-ILWR and ANNs with optimized learning rates and architecture sizes. The goal will be to achieve the smallest sum of squared error after 1,000 stochastic gradient descent training updates. For the ANN, this corresponds to 1,000 updates with the backpropagation algorithm.

Hidden layer sizes and learning rates for the ANN, as well as the number of knowledge points and learning rates for DI-ILWR, were optimized manually and using grid searches. The ANN sizes ranged from one neuron in one hidden layer to 20 neurons in two hidden layers (40 total hidden neurons). The learning rates ranged from .0001 to 110, growing by factors of two. During optimization, DI-ILWR's knowledge base ranged from 1 to 100 points, with the input learning rate ranging from 0 to 128, and output learning rate ranging from 1 to 50. A manual search was performed around the best parameters found by the optimizations to narrow down the granularity of the possible parameters. The best parameters found for the ANN and DI-ILWR are shown in Table 3.2. Though the ANN in Table 3.2 has fewer tunable parameters, recall that its morphology was optimized over sizes ranging from 5 to 501 tunable parameters.

Figure 3.14 shows the learning curves for these two parameter settings. DI-ILWR learns both faster and more smoothly. Based on these results, we conclude that, on the FitzHugh-Nagumo Rapid Approximation Task, DI-ILWR outperforms ANNs of all sizes, learning significantly faster, while remaining more stable.

First Hidden Layer Size	Second Hidden Layer Size	Learning Rate	Evaluation after 1,000 Training Points	Tunable Parameters
17	6	.4096	.238563	166

Number of Knowledge Points	Input Learning Rate	Output Learning Rate	Evaluation after 1,000 Training Points	Tunable Parameters
100	.05	5	.136878	300

Table 3.2: Best parameters for ANN (top) and DI-ILWR (bottom) for the FitzHugh-Nagumo Rapid Approximation Task. Evaluations were averaged over 10 trials.

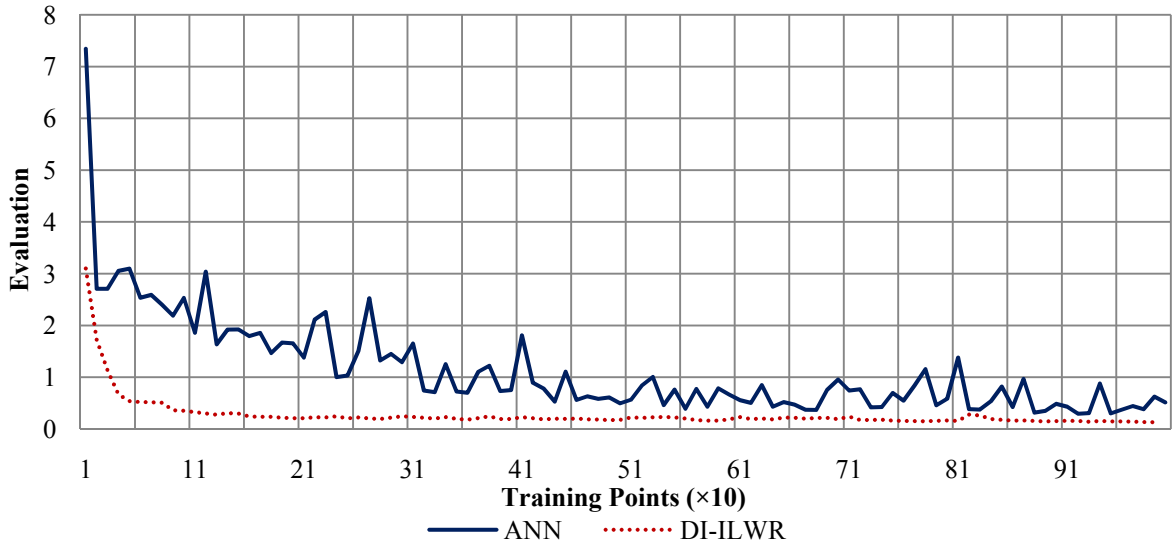


Figure 3.14: An ANN and DI-ILWR run using parameters from Table 3.2, for 1,000 training points on the FitzHugh-Nagumo Rapid Approximation Task. All points are averaged over three trials. Notice the horizontal axis is scaled by a factor of ten.

3.1.5 Non-Stationary Function

In reinforcement learning architectures, a function approximator is often required to approximate a non-stationary function (e.g. the critic in the actor-critic architecture). In this test, we evaluate the abilities of DI-ILWR and ANNs to track a simple non-stationary function. The function used is provided in Equation 3.13:

$$f(x,y,t) = e^{-\frac{1}{(x-\sin(t))^2+(y-\cos(t))^2}}. \tag{3.13}$$

The domain used was $x, y \in [-2,2]$ and $t \in [0,\infty)$, giving a range of $f(x,y,t) \in (0,1)$. As t increases, this function rotates around the origin, with a period of 2π. The following figures (3.15, 3.16) show f over the domain considered.

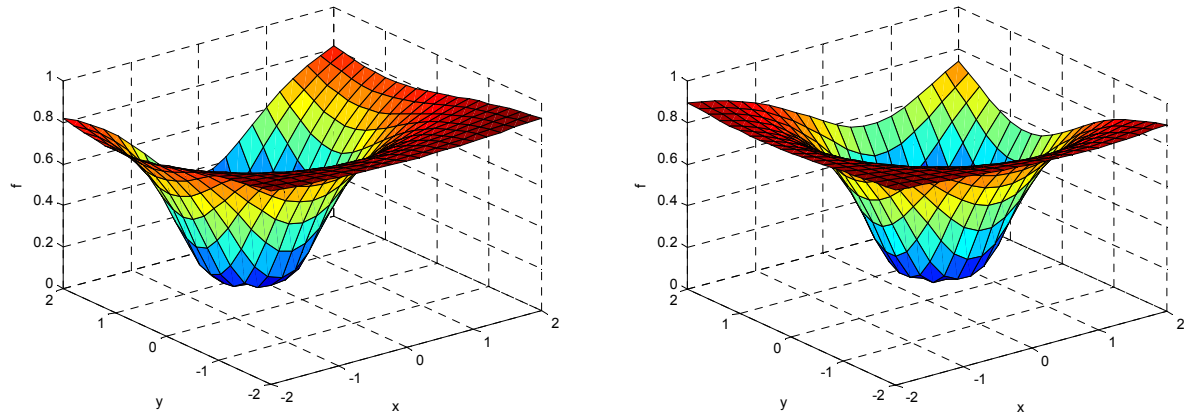


Figure 3.15: f shown for $t = 0$ on the left and $t = \pi/4$ on the right.

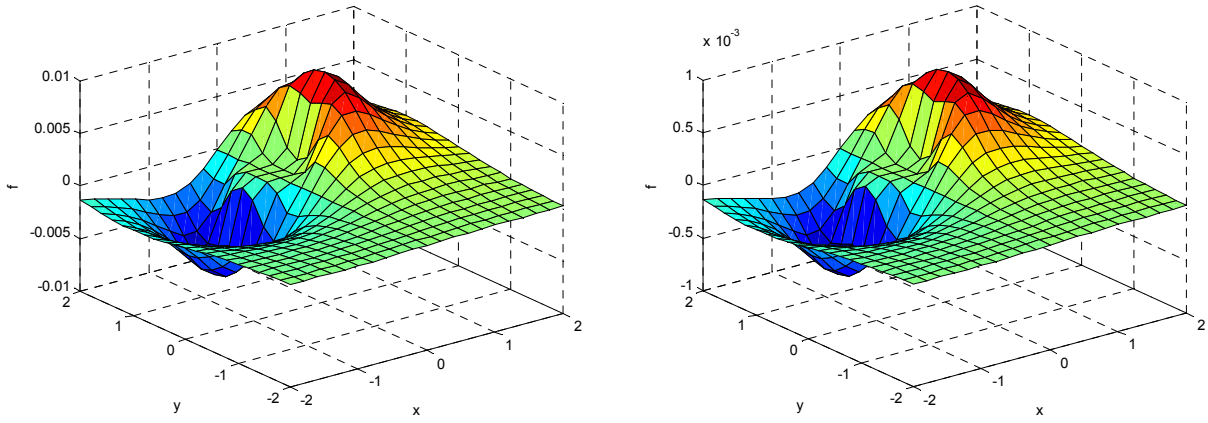


Figure 3.16: The difference between $f(x, y, 0)$ and $f(x, y, .001)$ is shown on the left, and the difference between $f(x, y, 0)$ and $f(x, y, .01)$ is shown on the right. Notice the magnitude of the difference with a time step of $\Delta t = .01$ is approximately ten times that with a time step of $\Delta t = .001$.

After each training point, t is increased by $\Delta t = .001$. Thus, the function completes a full revolution every 6,283.2 updates. The function approximators were allowed access only to x and y, making the function non-stationary. A grid-search optimization was run for ANNs and DI-ILWR to find the best sizes and learning rates, with $\mathbf{D} = diag(1,1)$. This optimization is identical

to that described in Subsection 3.1.4. Recall that this optimization allowed both to have similar numbers of tunable parameters. Evaluations are computed as the normalized (divided by 225) sum of the squared error between the approximation and the actual surface, over a 15×15 Sukharev Grid over the domain.

Each algorithm was run 10 times for 5,000 training points, and the final evaluations were averaged to create Table 3.3, which presents the best parameters found for each. Again, recall that, though the ANN has fewer tunable weights, the size presented was the best found during an optimization that allowed between 5 and 501 tunable parameters in the ANN. Figure 3.17 shows the evaluations of the ANN and ILWR over time. Figure 3.18 shows the actual surface at the end of the run that generated Figure 3.17, as well as the approximation thereof. Figure 3.19 shows the difference between the actual and approximated surfaces.

Tunable Parameters	**First Hidden Layer Size**	**Second Hidden Layer Size**	**Learning Rate**	**Evaluation after 5,000 Training Points**
83	20	1	.8192	.006076

Tunable Parameters	**Number of Knowledge Points**	**Input Learning Rate**	**Output Learning Rate**	**Evaluation after 5,000 Training Points**
400	100	10	10	.000701

Table 3.3: Best parameters for ANN (top) and DI-ILWR (bottom) for approximating the non-stationary function after 5,000 training points. Evaluations were averaged over 10 trials.

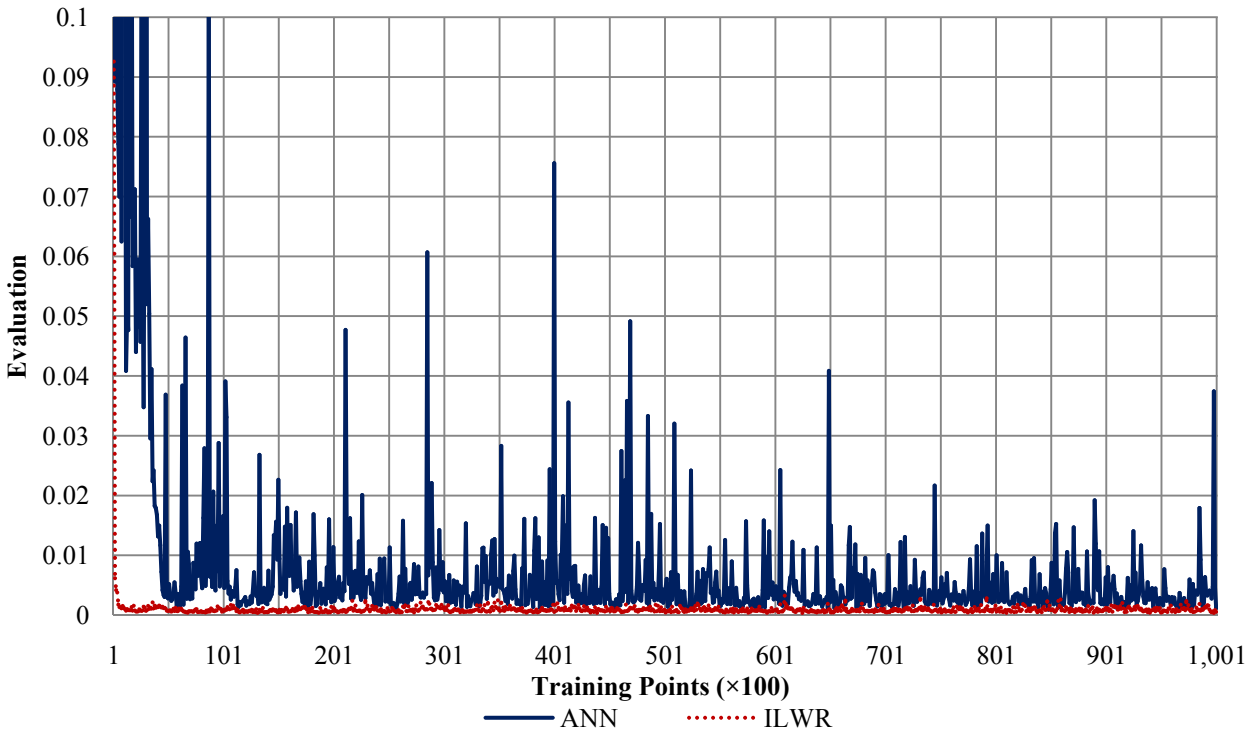


Figure 3.17: Evaluations of parameters from Table 3.3 in the non-stationary environment, over 100,000 training points (notice that the horizontal axis is scaled by a factor of 100). The average evaluations of the ANN and ILWR over the last 90,000 training points are .005 and .0009 respectively.

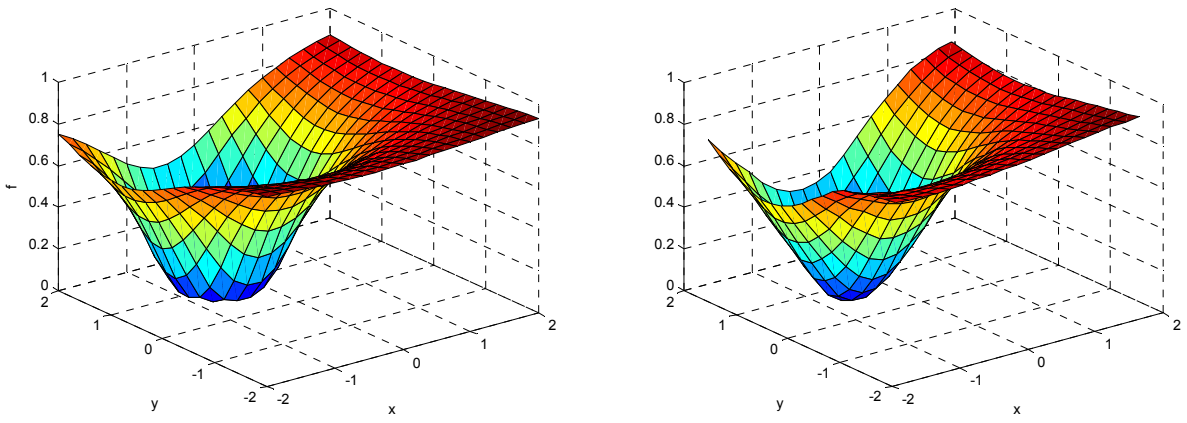


Figure 3.18: After the 100,000 training points of Figure 3.17, $t = 100$. At this point, the actual surface is shown on the left, and DI-ILWR's approximation is shown on the right.

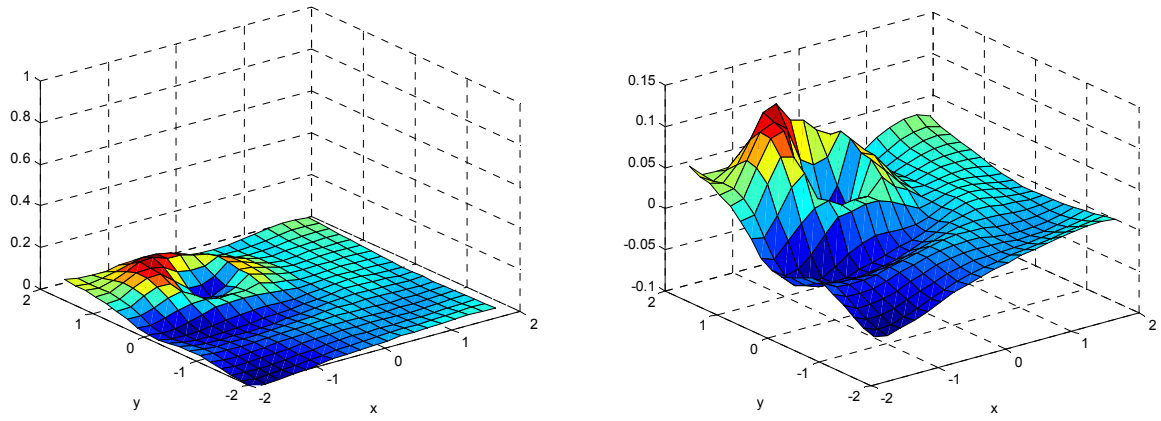


Figure 3.19: The difference (actual minus approximated) between the two plots in Figure 3.18, provided with the same axes scales as the previous plots (left), and a zoomed view (right).

Not only is DI-ILWR performing better than the ANN, but these figures show that it is tracking the function quite well, visually matching the target shape. Next, we will consider the effects of a more non-stationary function. To do this, the function will be accelerated to $\Delta t = .01$ without re-optimizing learning rates. The resulting learning curves for the ANN and ILWR are provided in Figure 3.20.

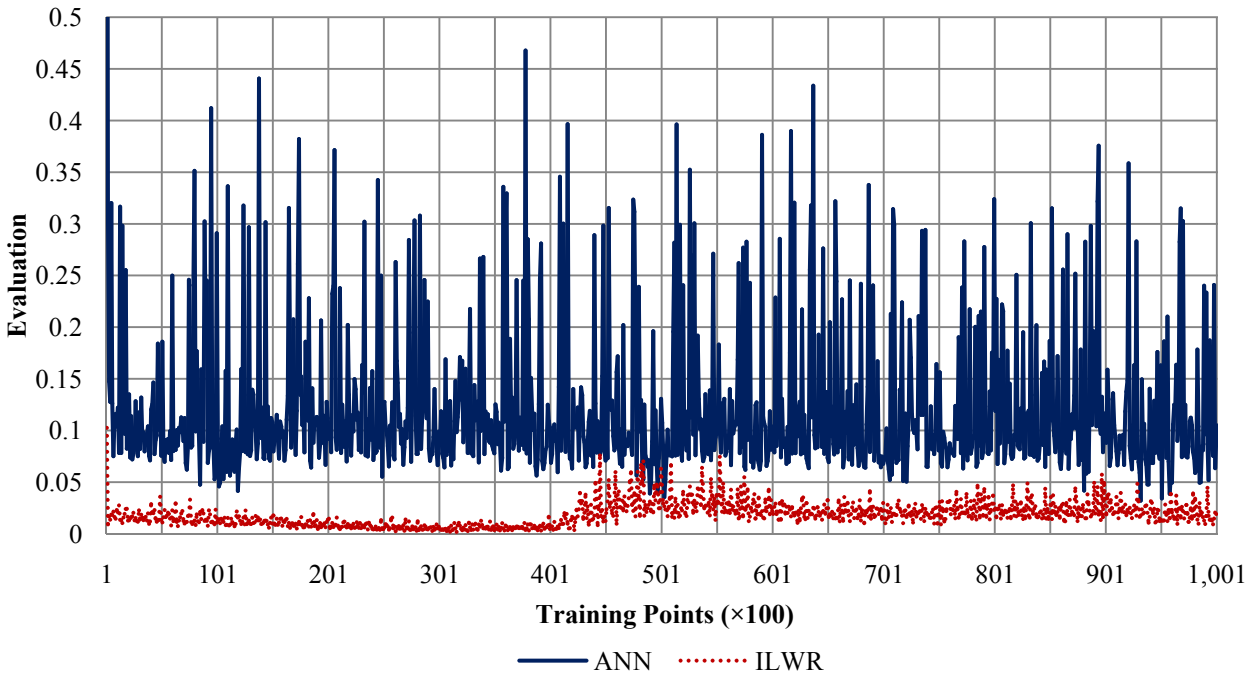


Figure 3.20: Evaluations of ANN and ILWR in the non-stationary environment with $\Delta t = .01$. Notice that the horizontal axis is scaled by a factor of 100.

3.2 Conclusion

The Sigmoid Environment showed that DI-ILWR is capable of more closely approximating functions than SI-ILWR because of its ability to move knowledge points freely in space. It also showed that SO-ILWR is not capable of accurately approximating functions, though moving knowledge point inputs may improve performance.

The Double Environment showed that DI-ILWR learns significantly faster than SI-ILWR when there are few knowledge points. With more knowledge points, DI-ILWR and SI-ILWR perform similarly. In architectures where a function approximator must be used to track a non-stationary target function, such as the critic in the actor-critic architecture, rapid convergence is

required. Thus, having a small set of knowledge points is beneficial, highlighting the benefits of DI-ILWR over SI-ILWR. Both outperform backpropagation on ANNs in all tests.

The FitzHugh-Nagumo Accurate and Rapid Approximation Tasks showed that DI-ILWR tends to distribute knowledge points with a higher density in areas with a larger second derivative. It also showed that DI-ILWR out-performs SI-ILWR, which is not surprising since the global minimum of DI-ILWR is guaranteed to be at least as good, because SI-ILWR is a special case of DI-ILWR where the input learning rate is zero. DI-ILWR also outperforms ANNs, learning faster and more smoothly than the best ANN.

The non-stationary function showed that DI-ILWR is capable of tracking a simple non-stationary function more accurately than an ANN. It also showed that DI-ILWR appears to be stable in the long-term when tracking a non-stationary function. These tests are relevant because the function the critic must approximate in the actor-critic architecture changes as the policy is refined.

Unlike ANN algorithms, DI-ILWR has local updates, which may be beneficial in certain reinforcement learning applications. This calls for further research into the performance of DI-ILWR in machine learning applications such as Q-Learning, the actor-critic architecture, and eligibility traces. We expect DI-ILWR to outperform ANNs when used with eligibility traces because of its ability to increase the eligibility locally, whereas increases to eligibilities in an ANN are global. This increased efficiency with eligibility traces may speed up the process of current rewards propagating back to previous states.

Further research into the ability of DI-ILWR to adapt to different **D** matrices (see Equation 2.73) is also warranted, though we expect that the choice of **D** does not have a significant effect on learning. Finally, further research should be conducted regarding the

potential benefit of splitting a function with more than one output (such as the Double Environment) into multiple sets of knowledge points, with one for each output.

CHAPTER 4:

DAS1 ARM SIMULATION EXPERIMENTS

The purpose of this chapter is to present the tests of the actor-critic's adaptive abilities on the DAS1 arm model of Section 1.2, which will be used in the subsequent chapters. Section 4.1 describes pre-training and evaluation. Sections 4.2 through 4.4 present tests of the actor-critic's adaptive abilities. Section 4.5 introduces a test to ensure that the system is robust to realistic sensor noise. Section 4.6 introduces an adaptation task developed specifically for use in the test described in Section 4.7.

Sections 4.8, 4.9, and 4.10 present tests that mimic complications that would arise if a human were to provide the reward signal. Notice that these are only preliminary tests to give researchers an idea of possible issues that may arise with humans giving rewards. Even though the actor-critic performs well on these tests, it does not mean that it will necessarily perform well when humans provide the rewards. There are several other complications that cannot be simulated well. A human subject may not be consistent in providing rewards, and may not be good at judging partial movements. Most significantly, the value function that the critic begins with may be unrepresentative of the value function for the human's reward system. The effects of these complications should be considered during human trials and the analysis of the subsequent results.

4.1 Pre-Training and Evaluation

In preliminary tests in which the actor and critic were both randomly initialized, the continuous actor-critic failed to converge to a desired solution, usually finding a policy that achieved one of the desired joint angles, but not the other. These tests included various function approximators and reward functions. These are likely two large local minima in policy space, which most policies converge to. In order to overcome this, we used supervised learning to pre-train the actor to mimic the PD controller discussed in Section 2.1. This corresponds to placing the policy near a minimum expected to be the global minimum or of similar utility. When the arm dynamics change, the policy will no longer be the optimal policy, but in many cases it will be close enough that the actor-critic's gradient descent in policy space will converge to the corresponding minimum.

This pre-training was executed differently when using ANNs and ILWR, so specifics for each are provided in Sections 5.1 and 7.1 respectively. For the tests described in this chapter, the actor-critic begins with an actor that is trained to mimic the PD controller.

In all tests, each episode lasts for two seconds and involves start and goal positions that have a combined joint angle difference of at least .6 radians, which requires movements to be significant. All initial states and target states have zero joint angle velocities. The reward signal used was always

$$r(t) = W\sum_i u_i^2 - d\left(\left(x, y\right), \left(x_{\text{Goal}}, y_{\text{Goal}}\right)\right), \tag{4.1}$$

where $W = -.016$, u is a vector of the requested muscle stimulations, x, y is the current position of the hand if both arm segments are one unit long (Equation 4.2), and $x_{\text{Goal}}, y_{\text{Goal}}$ is the target position of the hand if both arm segments are one unit long (Equation 4.3):

$$\left[x, y\right]^T = \left[\cos\left(\theta_1\right) + \cos\left(\theta_1 + \theta_2\right),\ \sin\left(\theta_1\right) + \sin\left(\theta_1 + \theta_2\right)\right]^T, \tag{4.2}$$

$$\left[x_{\text{Goal}}, y_{\text{Goal}}\right]^T = \left[\cos\left(\theta_{\text{Goal}_1}\right) + \cos\left(\theta_{\text{Goal}_1} + \theta_{\text{Goal}_2}\right),\ \sin\left(\theta_{\text{Goal}_1}\right) + \sin\left(\theta_{\text{Goal}_1} + \theta_{\text{Goal}_2}\right)\right]^T. \tag{4.3}$$

The value for W was computed such that the evaluation of the PD controller is identical to the evaluation when using the reward signal in Equation 2.57. Also, critics trained using the reward function in Equation 2.57 were found to be equally accurate when using the reward signal in Equation 4.1. This suggests that the two reward signals are almost identical. The reason for the switch in reward signals is that the muscle forces are not directly measurable in practice, though the requested muscle stimulations are.

To evaluate the actor-critic's performance, we use the average total reward per episode, computed over 256 fixed two-second episodes involving large motions over the state space. The integral of the reward signal over time was approximated using a backward Euler approximation. Recall that the larger the reward received, the better. The larger an evaluation is, the better it is. All rewards are negative, so the average reward per episode must be negative. For comparison throughout, the evaluation of the PD controller, from Chapter 2, is $-.18$. Visually, this policy appears near optimal, and at no time has any controller, including the PD and PID, achieved an evaluation above $-.17$, suggesting that this evaluation may be near that of the optimal policy.

4.2 Control Test (CT)

The first test was the *Control Test* (CT), in which the dynamics of the DAS1 arm are unchanged, allowing the actor-critic to further adapt to the standard arm model. Appendix E contains the parameters used in the DAS1 model for the CT. The PD controller's evaluation on this test is $-.18$. Because the PD controller is a linear controller and the actor is capable of

representing nonlinear functions using either an ANN or ILWR, the actor may be able to learn a policy that is superior to that of the PD controller.

4.3 Baseline Biceps Test (BBT)

The second test was inspired by PD controller human trials in which the subject had spasticity of the biceps brachii, causing the biceps to exert a constant low level of torque on both joints. This *Baseline Biceps Test* (BBT) involved adding 20% of the maximum stimulation (100%) to the stimulation requested by the controller (clipped to 100%) in order to simulate spasticity. In the BBT, when using the PD controller or the actor-critic trained on it, the steady state of the arm is counterclockwise of the goal state at the point where the controller's requested triceps stimulation balances out the baseline biceps stimulation. The PD controller's evaluation on the BBT is $-.41$.

4.4 Fatigued Triceps Test (FTT)

The third test, the *Fatigued Triceps Test* (FTT), simulates the effects of a muscle being severely weakened. In this test, the triceps stimulation used is 20% of the requested triceps (long head) stimulation. Thus, when a controller requests full triceps stimulation, only 20% will be applied. Unlike the BBT, this does not change the steady state when using the PD controller, though it does induce overshoot if the initial configuration is clockwise of the goal. This occurs because the biceps is used to pull the arm towards the goal, and the triceps is used to stop it at the

goal configuration. With the triceps weakened, the PD controller does not exert enough torque to overcome the arm's angular momentum.

The PD controller's evaluation on the FTT is $-.19$. This evaluation is high compared to the BBT because the FTT has no steady state error. The extra negative rewards accrued during the brief overshoot of the target state when starting clockwise of the goal, and the slower motion to the goal when starting counterclockwise of the goal result in only a small difference in the integral of the reward signal used for evaluations. However, especially when starting clockwise of the goal, there is a clear visual degradation of performance between the CT and FTT. When evaluating the actor-critic's performance, rather than focusing on the numerical evaluation, it will therefore be useful to visualize the magnitude of the overshoot as training progresses.

4.5 Noise Robustness Test (NRT)

In practice, the exact joint angles and their velocities are not known. Sensors can directly measure joint angles, though there is always minor error, which can be simulated as noise. The joint angle velocities are approximated using the difference between successive joint angle measurements, run through a low pass filter. Thus, the error in joint angle approximations can result in even larger error in the velocity calculations. The fourth test, the *Noise Robustness Test* (NRT), attempts to model this noise and test the robustness of the controller on the BBT in a noisy environment. Normal Gaussian noise was added to both the joint angle measurements, $\theta(t)$, and the joint angle velocity measurements, $\dot{\theta}(t)$, scaled by the constants σ_θ and $\sigma_{\dot{\theta}}$ respectively. Realistic values for these two parameters are $\sigma_\theta < .1$ and $\sigma_{\dot{\theta}} < .3$. The PD controller's evaluation on this test combined with the BBT with $\sigma_\theta = .1$ and $\sigma_{\dot{\theta}} = .3$ is $-.45$.

Another version of the NRT involves adding a bias to the sensor readings. This corresponds to a sensor not being properly calibrated and having a constant level of error in it. This bias need only be added to the joint angle measurements, as it will factor out during velocity computations. The scale and direction of the bias to each joint angle measurement is static and defined as μ_B.

4.6 Fatigued Biceps Test (FBT)

The fifth test, the *Fatigued Biceps Test* (FBT) was developed for use in the Toggling Test, which is described in the next section. The FBT was developed for the TT because the BBT and FBT require different policies for improved performance. The FBT is identical to the FTT, except that rather than modifying the triceps, the biceps is weakened. Thus, the biceps stimulation used is 20% of the requested biceps stimulation. When a controller requests full biceps stimulation, only 20% will be applied. The PD controller's evaluation on this test is $-.19$.

4.7 Toggling Test (TT)

In the *Toggling Test* (TT), the environment switches cyclically between various other tests (e.g., BBT and FTT) after a fixed number of episodes. The agent must continuously adapt to changing dynamics. However, the BBT and FTT require similar policies for improved performance. In both, increased triceps stimulation and decreased biceps stimulation will improve performance. The FBT was created so that policies that performed well on the BBT would be expected to perform poorly on the FBT. In the remainder of this thesis, whenever the TT is run, it will toggle between the BBT and FBT. The exact details of when toggling will occur

will be provided prior to discussing the results of the TT. These details are not presented in this section because the necessary results to justify these decisions have not yet been revealed.

4.8 Delayed Reward Test (DRT)

Dr. Antonie van den Bogert and Kathleen Jagodnik, two researchers at the Cleveland Clinic Lerner Research Institute (LRI), are interested in trials using the reinforcement learning controller derived from this thesis for human trials. Trials are scheduled to begin during Summer 2009 in which able-bodied human subjects will provide the reward signal. Researchers are interested in whether humans will give rewards in such a way that the agent is encouraged to learn more natural strategies. Having humans provide the rewards presents two challenges not inherent to the model. These are simulated in the Delayed Reward Test and the Discrete Reward Test (Section 4.9).

In the standard setup, the agent is given the reward signal and state after every .02 seconds. During trials in which a human provides the reward, the state can still be measured every .02 seconds, but a human may not be capable of accurately providing rewards at a rate of 50Hz. This would be especially difficult for a patient suffering from SCI (Section 1.1), who may be using an unnatural sensor, such as a sip-and-puff sensor (e.g. the Breeze™ sip/puff switch that is in use at the LRI and is made by Origin Instruments of Grand Prairie, Texas), to provide input to the computer system.

In the *Delayed Reward Test* (DRT), the reward signal is only provided once every $.02k_\tau$ seconds, where k_τ is an integral time scaling constant. This can be simulated two ways. First, the reward at time t could be presented at all subsequent .02 second updates until the reward is

updated after $.02k_\tau$ seconds. Second, the sum of the rewards over the $.02k_\tau$ interval could be given every $.02k_\tau$ seconds, with the actor-critic only updating eligibility traces during the other .02 second state updates. We decided to use the latter method because it better approximates how we expect a human to present rewards. The DRT is combined with the BBT to judge adaptive ability with delayed rewards. Any controller's evaluation on the combination of the BBT and the DRT is initially the same as the BBT because the DRT affects learning, not initial performance.

4.9 Discrete Reward Test (DiRT)

A human subject using a sip-and-puff sensor would have trouble representing a real-valued reward signal accurately while simultaneously providing values rapidly enough for the system to learn. To simplify the task, researchers at the LRI decided to discretize the reward signal. We performed trials with various granularities of discretization, and discretization into five values was found to still allow for accurate learning. SCI patients can be expected to achieve five different values quite easily with a sip-and-puff. Thus, the results for discretization into five values will be presented for this test. This test is called the *Discrete Reward Test* (DiRT).

When discretizing the reward signal, we wish to do so into sections representative of various regions. For example, if there are 3 possible discrete rewards to which the real-valued reward signal will be mapped, one of the three discrete rewards should not correspond to a rare real-valued reward interval. Rather, we prefer that all three rewards occur frequently and preserve as much information from the real-valued reward as possible. In order to determine how best to discretize the reward signal, consider Figure 4.1, which depicts the reward signal plotted over 10 random episodes on the CT. From this plot, the discretization into the 5 values provided in Table 4.1 was derived.

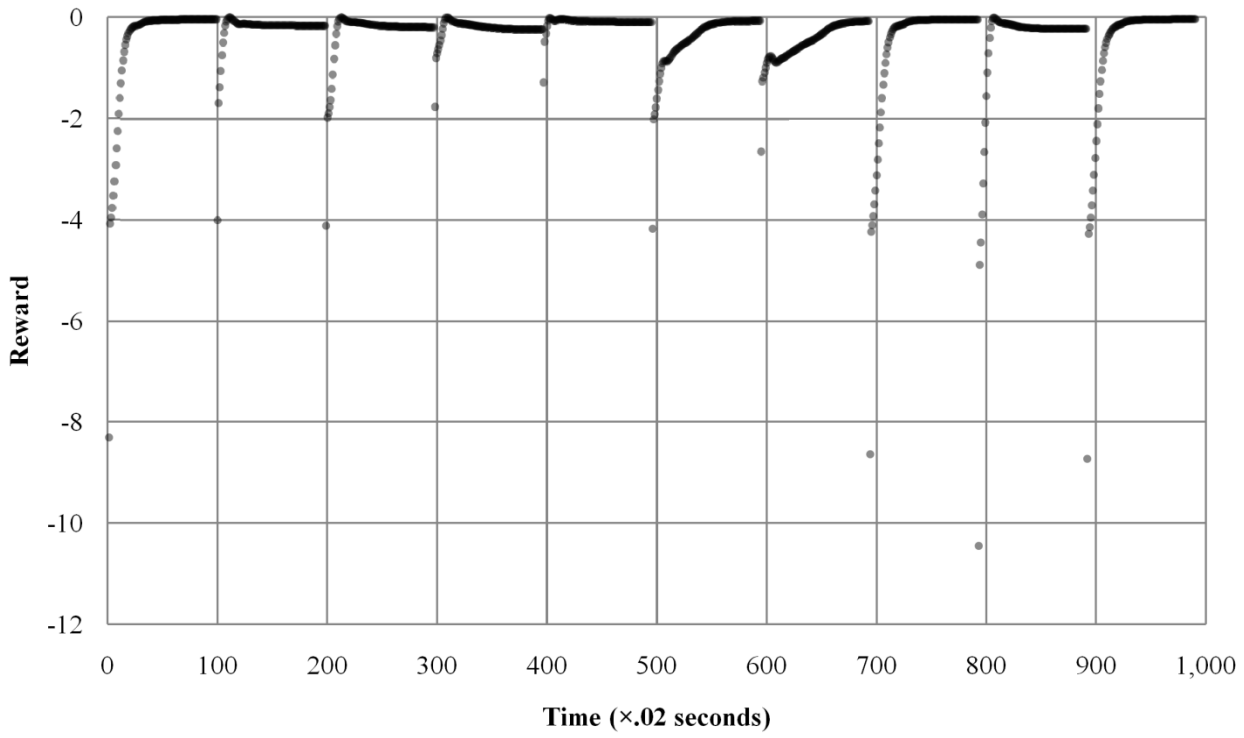


Figure 4.1: Reward signal plotted over 10 random episodes using the PD controller on the BBT, with the reward evaluated every .02 seconds. Each episode lasts two seconds, resulting in 1,000 points. Transparency was used to show the density of points.

Reward Range:	Reward ≥ –.25	–.25 > Reward ≥ –.5	–.5 > Reward ≥ –1	–1 > Reward ≥ –2	–2 > Reward
Meaning:	At the target state	Very near the target state	Approaching the target state	Not near the target state	Far from the target state
Discrete Mapping:	–.25	–.5	–1	–2	–5

Table 4.1: Reward discretization into 5 values.

Similar discretizations with seven values and three values were also tested. Performance with seven values was similar to that of five values. The system still learned when using only three values, though there was enough of a difference to warrant the use of five values.

4.10 Continuous Learning Modification (CLM)

When applied to an actual subject with SCI, individual episodes will not be entirely independent: each subsequent movement will begin where the previous episode ended. In all other tests, each episode began at a random initial state with zero joint angle velocity. With the *Continuous Learning Modification* (CLM), the initial state of each episode is the same as the terminal state of the previous episode. Though it would appear that this would not significantly impact performance, it reduces the amount of the state space that is explored over a fixed amount of time, which may slow learning. The CLM is run in conjunction with the BBT, testing the actor-critic's adaptive abilities when the episodes are relatively continuous. The word "continuous" in the CLM is a slight misnomer, as the state of the environment is not quite continuous over episodes. If the arm does not terminate at the target position, it will still begin the next episode at the target position. However, this can be viewed as being relatively continuous when compared to non-CLM trials in which subsequent episodes have random initial states.

CHAPTER 5:

DAS1 ANN ACTOR-CRITIC RESULTS

This chapter is broken into 11 sections. In Section 5.1, the actor and critic are pre-trained to mimic the PD controller and its corresponding value function, respectively. In Section 5.2, the parameters of the continuous actor-critic architecture are optimized and four parameter sets are selected for further study. Parameter sets A and B both learn rapidly, though A has excessive exploration and B has little exploration. The Fast and Slow parameter sets both have a practical amount of exploration, between those of parameter sets A and B. The Fast parameters learn quickly, though they are unstable. The Slow parameters learn slowly, but remain stable.

Sections 5.3 through 5.10 present the results for the tests described in Chapter 4. Plots of joint angle trajectories are provided for the CT, BBT, and FTT to provide a better understanding of different evaluations on each test. Chapter 5 concludes with Chapter 5.11, which describes an unexplained phenomenon that was discovered during trials of the continuous actor-critic on the BBT.

5.1 Pre-Training

As described in Section 4.1, supervised learning was used to train the actor ANN, with ten neurons in its only hidden layer, to mimic the PD controller. To do this, the actions for 550,000 training points and 170,000 testing points, each consisting of the state and corresponding action generated by the PD controller, were run through the inverse sigmoid,

generating training pairs for the actor ANN, $A\left(x(t);w^{A}\right)$ from Subsection 2.2.8. The actor ANN was then trained using the error backpropagation algorithm with a learning rate of .001 (Russell and Norvig, 1995). After 2,000 epochs of simulation, each of which consisted of training once on each of the 550,000 training points, the actor converged to a policy qualitatively similar to the PD controller's policy. This policy has an evaluation of $-.21$ on the standard arm model (CT), which can be compared to the PD controller's evaluation of $-.18$. The performance of this pre-trained ANN actor on the CT, BBT, and FTT are depicted in Figure 5.1.

Trials with as few as 5 hidden units and as many as 100 hidden units in two hidden layers did not result in significant improvements in learning speed nor long-term stability. The policy that the ANN must represent is nonlinear due to the inverse sigmoid function applied to the PD controller's policy. Achieving significantly improved accuracy would require adding many neurons, which would in turn result in more weights. This increase in tunable parameters increases the dimension of the policy space in which the actor-critic performs gradient descent, resulting in slower learning. We therefore desire a small number of neurons to allow for more rapid learning, while keeping enough neurons that the *inductive bias* (Mitchell, 1997) does not result in poor policies.

The critic ANN was then trained using the actor-critic architecture with the previously trained actor. The actor's policy was fixed while the critic was trained. Two critics were created, the first with 20 neurons in its hidden layer, and the second with ten. Both achieve average TD-error magnitudes of .1 on the unmodified arm (CT) when using the ANN actor. These two ANNs will be referred to as ANN critic-20 and ANN critic-10, respectively. ANN critic-20 was initially created and used with parameter sets A and B, defined in Table 5.1. It was then observed that 10 neurons were sufficient for the hidden layer, and ANN critic-10 was used for subsequent tests

with the Fast and Slow parameter sets, defined in Table 5.2. ANN critic-20 was trained to be accurate for the actor ANN without exploratory noise. ANN critic-10 was trained to be accurate for the actor ANN with exploratory noise.

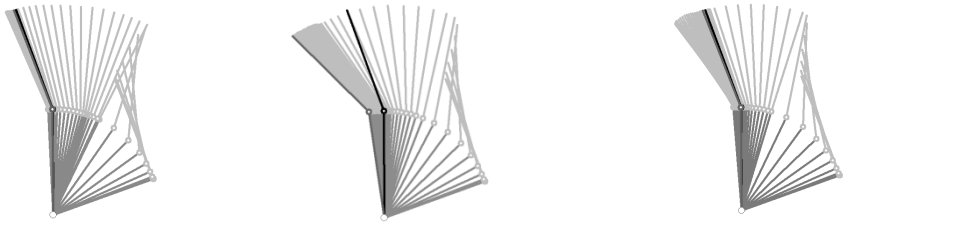


Figure 5.1: Initial actor ANN's performance on a particular motion for the three adaptation tests. The black state is the goal state (90°, 20°), the medium grey state is the final state after two seconds of simulation, and the light grey states are snapshots of the arm location taken every 20ms. The initial condition is the clockwise-most trace (20°, 90°). In the BBT, the final state is the counterclockwise-most trace, while in the control test and FTT the final state partially obscures the goal state.

When both were trained, it was found that adding a cap on the magnitude of the TD-error, δ, from Equation 2.39, can improve stability. Therefore, the TD-error magnitude was capped at .5 during training and for all subsequent trials, unless otherwise specified. Trials were performed with critics of sizes ranging from 5 hidden units in one hidden layer, to 100 across two hidden layers. None performed better, with respect to learning speed and stability, than the 10 hidden neuron ANN critic-10. The remainder of this thesis therefore focuses on ANN critic-10.

For each two-second episode, when training the critic, the start and goal were randomly selected with the sum of the difference in joint angles (in radians) between the initial and goal configurations being greater than .6. This constraint removed episodes in which the arm did not have to make a significant motion. After each episode, the eligibility traces were all set to zero. All further training was performed with the same episode duration and constraints.

The actor-critic thus begins all of the following tests with an actor ANN that is a close approximation of the PD controller, and an accurate critic for the CT. When the arm dynamics change, the critic will not be accurate, but must reconverge.

5.2 Parameter Optimization

The parameters for the actor-critic model, η_A, η_C, τ_N, κ, and σ, defined in Equations 2.42, 2.41, 2.44, 2.40, and 2.43 respectively, were initially optimized using *Random Restart Hill Climbing Search* (RRHCS), defined in (Russell and Norvig, 1995). This optimization was performed using the pre-trained ANNs described in Section 5.1. The parameters Δt and τ were fixed to $\Delta t = .02$ and $\tau = 1.$ The optimization was run for learning on the BBT, with the gradient sampled at 90% and 110% of the current value for each learning parameter. The ANN actor's initial evaluation (Section 4.1) on the BBT is $-.65$. Each parameter set's learning ability was measured as the average evaluation after training for 100, 200, 500, and 1000 random episodes. Random restarts used a "logarithmic distribution" half the time, and a linear distribution the other half of the time in order to better explore the extremes and full range of the parameter space. Points for the logarithmic distribution were sampled as

$$e^{\text{random}(\ln(min),\ln(max))}, \tag{5.1}$$

where random returns a random number with uniform distribution over the range $[min, max]$.

In this section (5.2), references to the evaluations of parameter sets refer to this evaluation scheme, while all other sections only refer to the evaluation of a specific policy (Section 4.1). Values for these two evaluation schemes should not be confused. For example, a

policy with an evaluation of $-.28$ would be poor, while a parameter set with an evaluation of $-.28$ could be quite good.

This optimization could lead to overfitting of the learning parameters for the task of learning on the BBT, so generalizability is evaluated later via the FTT.

Of the 4,460 learning parameter sets examined by RRHCS, 1,363 had evaluations greater than $-.3$. However, many of the best learning parameter sets found by the optimization did not have stable evaluations because the training episodes and exploration are stochastic. For example, the best parameter set received an evaluation of –.22 during the optimization, though further tests found its average evaluation was –.33 with a standard deviation of .15 (N=100). P*arameter set A* and *parameter set B*, defined in Table 5.1, were selected for further inspection due to their consistently good evaluations, as well as their different characteristics with respect to exploratory noise.

Parameters	η_A	η_C	τ_N	κ	σ	τ	**Mean Evaluation**	**Std. Dev.**
A	.001	.0001	.55	.55	74.5	1	–.267	.01
B	99.5	34.4	2500	71.5	7991	1	–.286	.09

Table 5.1: Two of the best parameter sets found from optimization. Means and standard deviations of the evaluations were calculated with a sample size of N=30 evaluations.

These two parameter sets both learn well on the BBT and FTT, though they use significantly different exploration. Parameter set A uses a massive amount of exploratory noise, allowing it to fully explore the state and action spaces, while Parameter set B exploits the current knowledge, with subtle exploration injected into the policy. In a typical episode on the control test, the average sum of the squared joint angle noise injected into the policy for parameter set A

was four orders of magnitude larger than that of parameter set B. Figure 5.2 depicts the difference between the exploration of the two parameter sets. For further discussion of these parameters and the effects of varying exploratory noise magnitude, see Section 5.6 and (Thomas et al., 2008a).

It was later realized that parameter set A does not conform to the constraint $0 < \kappa \leq \tau$. Experiments in which τ_N was changed in parameter set A to $\tau_N = 1$ led to results that were almost identical to those when $\tau_N = .55$.

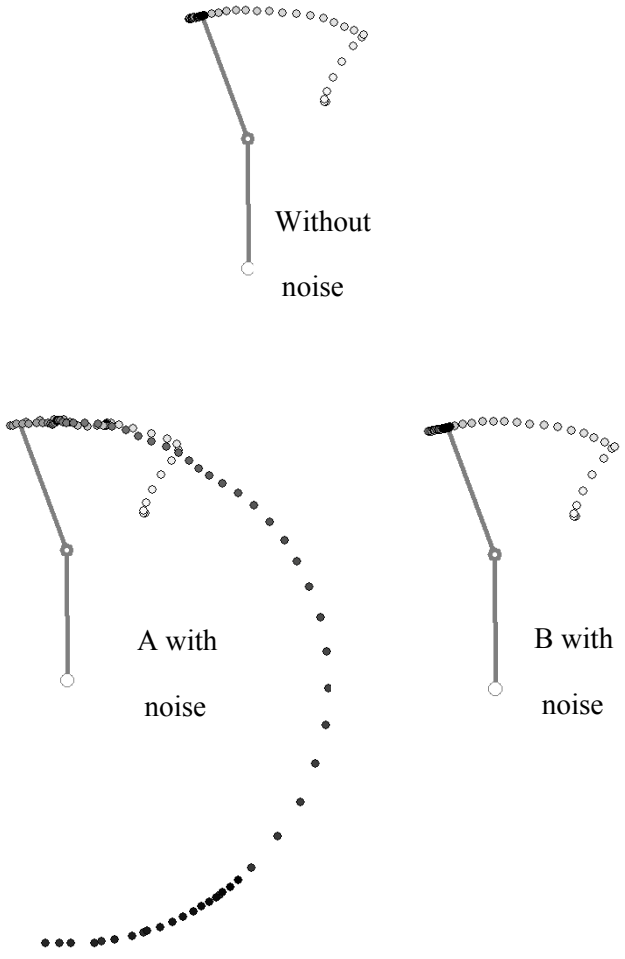


Figure 5.2: Plot of the hand position when using learning parameter set A or B without noise (top), A with noise (bottom left), and B with noise (bottom right). All are attempting the same motion to the grey goal state. Dots, starting white and fading to black, map the hand position every 20ms.

Though parameter set B exhibits little exploration and parameter set A exhibits excessive exploration, human trials would require exploration between the two. Current research by Kathleen Jagodnik and Dr. Antonie van den Bogert at the Lerner Research Institute involves the application of the controller developed herein to human trials in which the subject gives the rewards. For such a test to succeed, the explorational noise must be large enough for a human subject to discern the difference between a motion with and without exploration, and small enough that it does not largely obscure the motion of the current policy.

The explorational noise is defined by the parameters σ and τ_N. The values $\sigma = 9,000$ and $\tau_N = 2,400$ were found to produce exploration as desired. In order to ensure conformity with the constraint $0 < \kappa \leq \tau,$ the values $\tau = .1$ and $\kappa = .1$ were selected. The remaining parameter sets were found to fit into two categories: parameters that result in fast initial learning but an unstable system that diverges in the long-term; and parameters that learn slowly but remain stable in the long-term. Two prototypical examples of such parameters, the *Fast parameters* and the *Slow parameters*, are provided in Table 5.2. These parameters are analyzed in the following sections as well as in (Thomas et al., 2009a). Notice that the critic's learning rate for the Fast parameters is zero. The Fast parameters likely utilize the shape of the reward function and pre-trained critic for initial learning, though future work should be done to determine why the continuous actor-critic learns with a fixed critic.

Parameters	η_A	η_C	σ	τ_N	τ	κ
Slow	**10**	**.344**	9,000	2,400	.1	.1
Fast	**70**	**0**	9,000	2,400	.1	.1

Table 5.2: One of the best parameter sets found from optimization with manual tuning, the Fast parameters, and a derivation thereof, the Slow parameters.

Because these parameters have a more practical magnitude of exploration than parameters A and B, parameters A and B will only be considered in Section 5.6, and not included in results for the other tests.

5.3 Control Test (CT)

Using the Fast parameters on the CT, the system improves its evaluation before becoming unstable. Qualitatively, the arm movements begin to oscillate around the goal state within the first 1,000 episodes. Using the Slow parameters, learning is significantly slower, though stable. Figure 5.3 shows the short and long-term performance of both parameter sets on the control test. Notice the logarithmic scale of the horizontal axis, representing training time in terms of episodes. We desire rapid initial learning over the first few hundred episodes, as well as stability in the long-term, represented by the evaluation out to 10,000 episodes. This logarithmic scale of the horizontal axis will be utilized in all of the learning curves shown, except where emphasis is placed on the short-term performance. The initial evaluation is $-.21$.

Figure 5.4 compares the joint angle trajectories on the CT for one movement before training and after training, which will be useful for comparison to other tests.

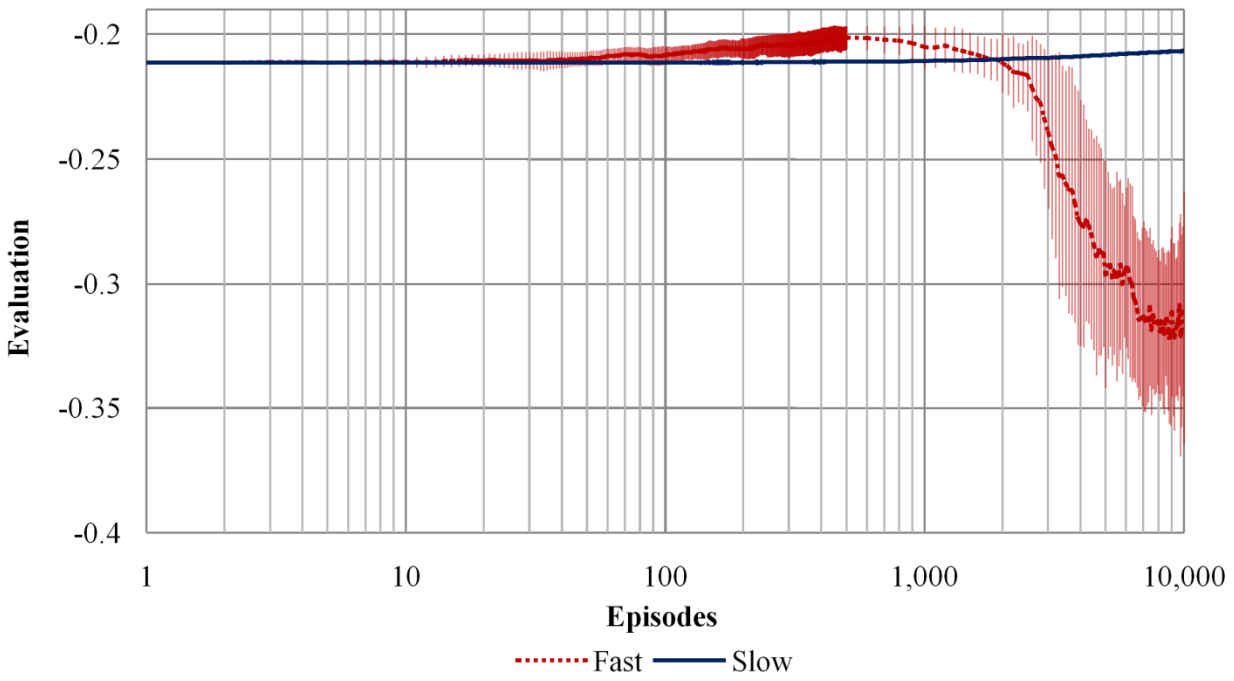


Figure 5.3: The actor-critic's mean evaluation (N=10) on the control test using both the Fast and Slow parameters with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis.

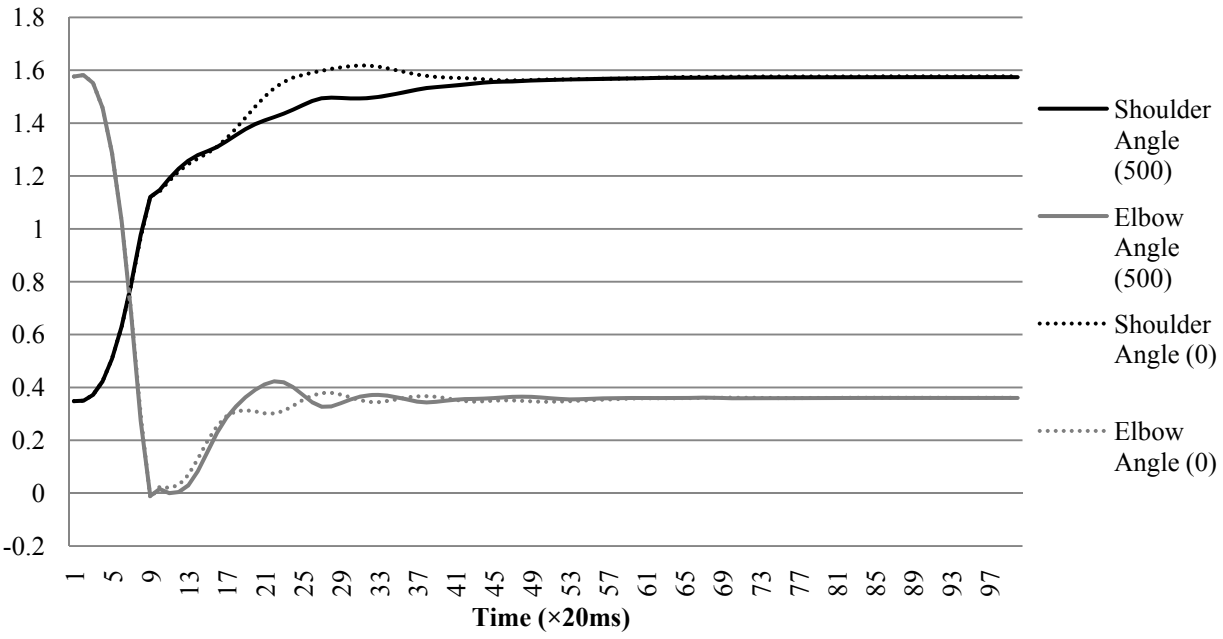


Figure 5.4: Joint angle trajectories before training (dotted) and after 500 training episodes (solid). The target movement is from $\theta = (.35, 1.57)$ to $\theta_{\text{Goal}} = (1.57, .35)$. The horizontal axis spans one episode.

5.4 Baseline Biceps Test (BBT)

Because the learning parameter sets were optimized using the BBT, the Fast parameters perform well on the BBT, quickly removing overshoot of the goal when the initial configuration is clockwise of the goal configuration, and generating a steady state close to the goal state. Once again, the Fast parameters are unstable in the long-term, though the Slow parameters remain stable, as shown in Figure 5.5. Trials using the Slow parameters for several million episodes confirmed empirically that the Slow parameters are stable in the long-term.

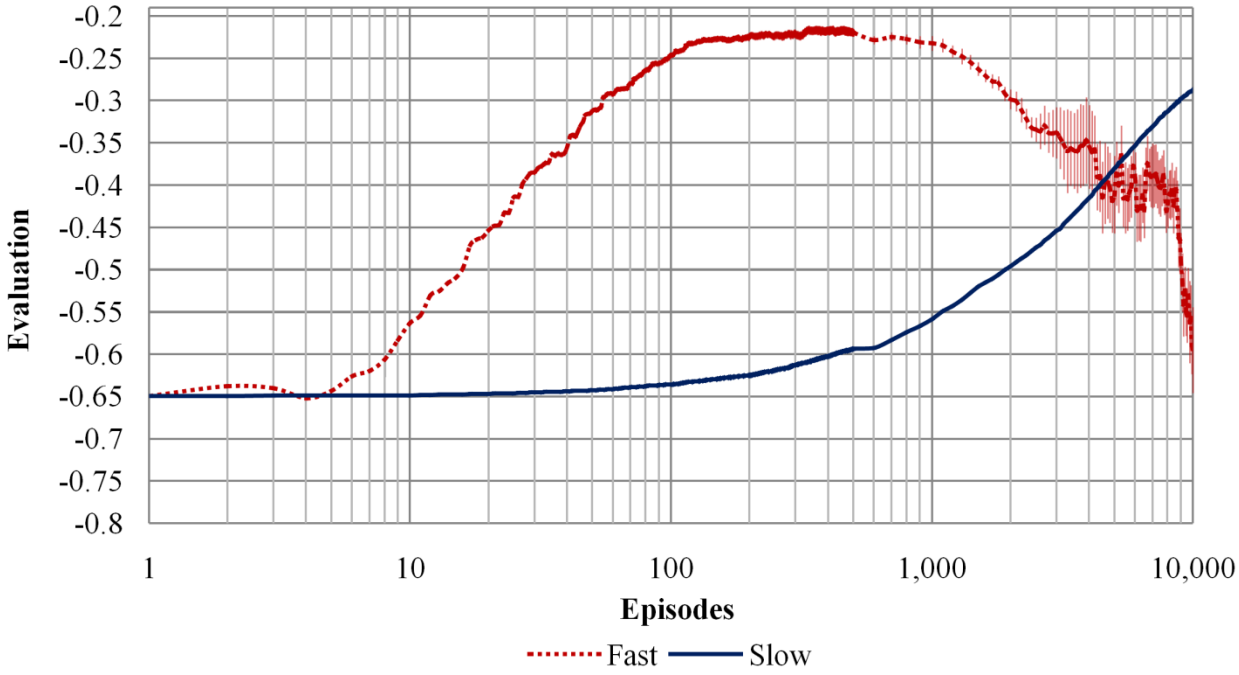


Figure 5.5: The actor-critic's mean evaluation (N=10) on the BBT with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis.

Figure 5.6 compares the joint angle trajectories on the BBT for the same movement before training and after training. The desired movement is identical to that of Figure 5.4. Notice the significant improvement in the elbow joint angle trajectory after training.

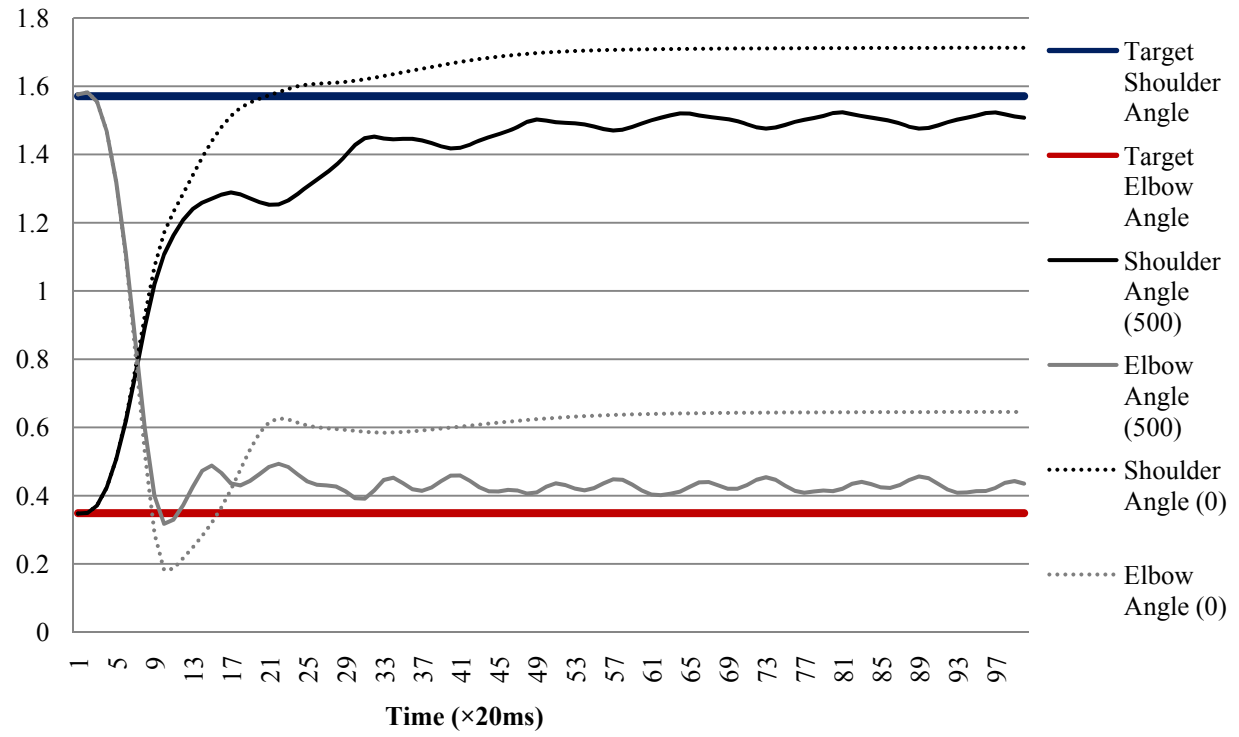


Figure 5.6: Joint angle trajectories before training (dotted) and after 500 training episodes (solid) on the BBT. The target shoulder and elbow angles are provided as the thick blue and red lines near .35 and 1.57, respectively. The horizontal axis spans one episode.

5.5 Fatigued Triceps Test (FTT)

The learning parameter sets' ability to adapt to changing dynamics was then tested using the FTT. Because the parameters were optimized using the BBT, the FTT serves as a test of their generalizability to other changes in dynamics. The Fast parameters remove the overshoot within

200 episodes. Performance is consistent with the previous tests, with the Fast parameters initially learning rapidly, then diverging, while the Slow parameters learn more slowly, but remain stable as shown in Figure 5.7. Trials using the Slow parameters have empirically shown that the system remains stable after 1 million training episodes (not shown).

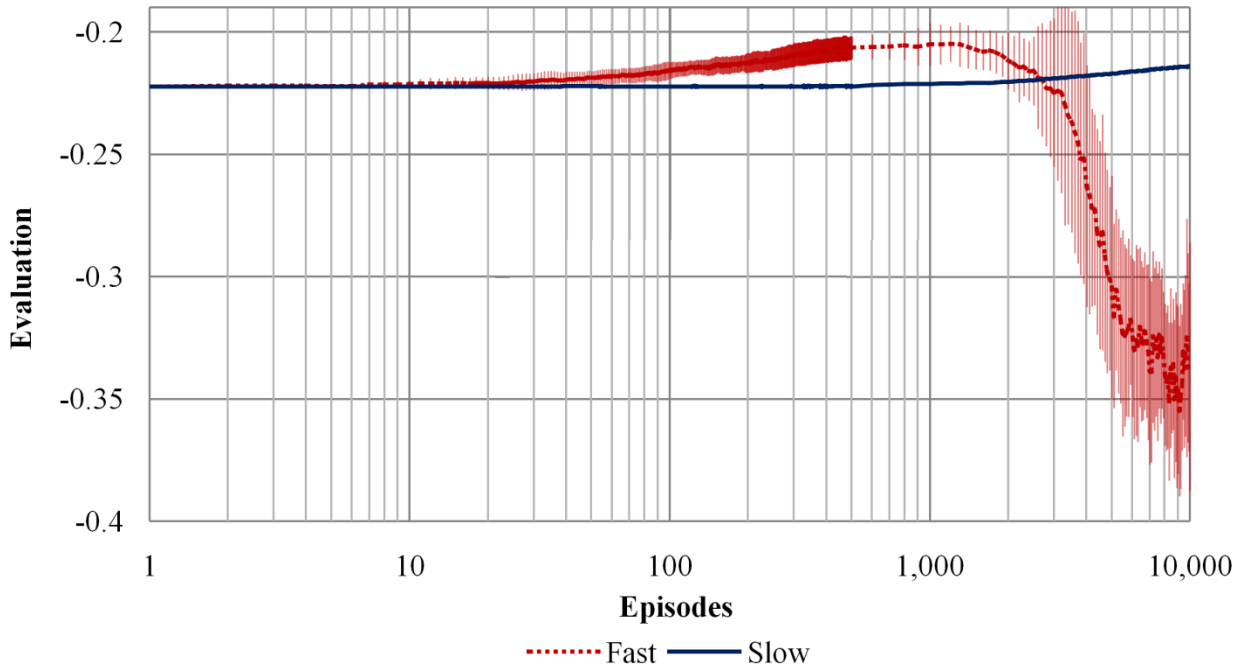


Figure 5.7: The actor-critic's mean evaluation (N=10) on the FTT with standard deviation error bars provided. Evaluations represent those just prior to the episode number marked on the horizontal axis. As with Figure 5.3, notice the logarithmic horizontal axis.

Figure 5.8 compares the joint angle trajectories on the FTT for the same movement before training and after training. The desired movement is identical to those of Figures 5.4 and 5.6. Notice that, after training, the overshoot of the elbow joint angle has been significantly reduced.

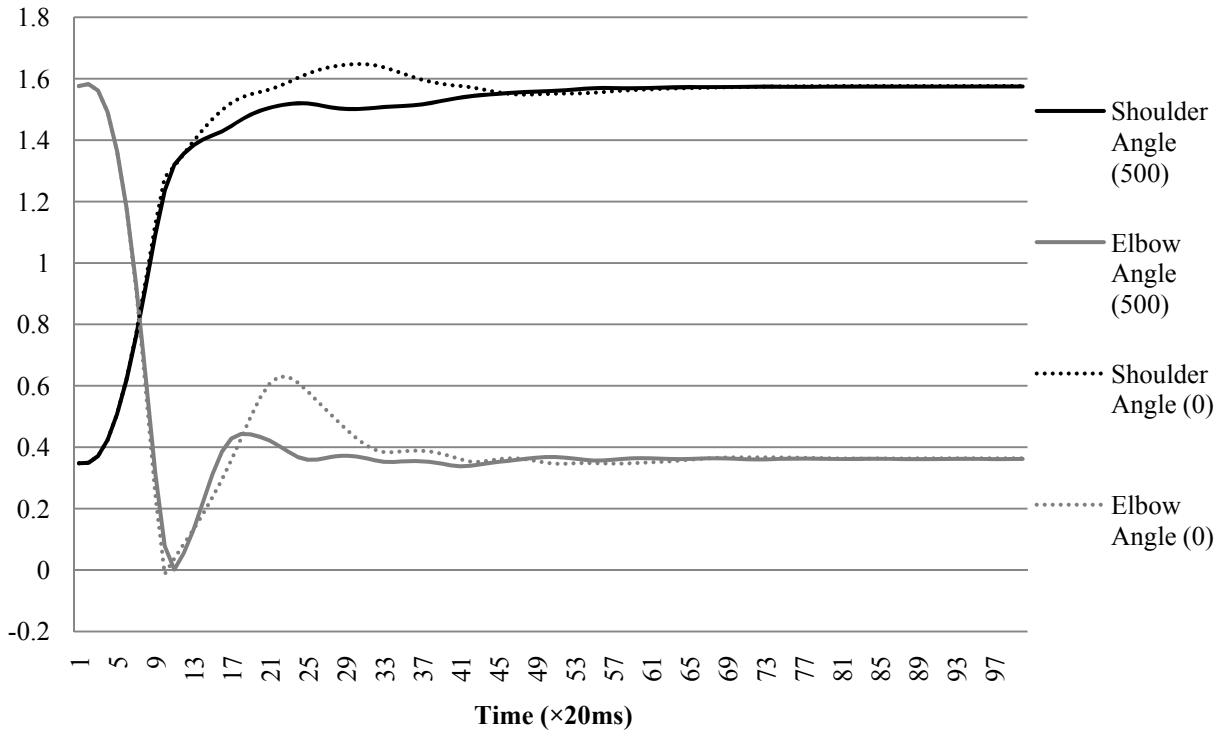


Figure 5.8: Joint angle trajectories before training (dotted) and after 500 training episodes (solid) on the FTT. The horizontal axis spans one episode.

5.6 Effects of Exploration

This section discusses the results on the CT, BBT, and FTT when using parameter sets A and B, defined in Table 5.1. As discussed in Section 5.2, parameter set A includes significant exploration, while parameter set B has minor exploration. Both were found to perform similarly to the Fast parameters of Table 5.2 on the CT, BBT, and FTT. Neither is stable in the long-term. Though long-term plots are not provided, this instability will be evident in the following short-term plots. Figure 5.9 shows performance on the CT, Figure 5.10 shows performance on the FTT, and Figure 5.11 shows performance on the BBT.

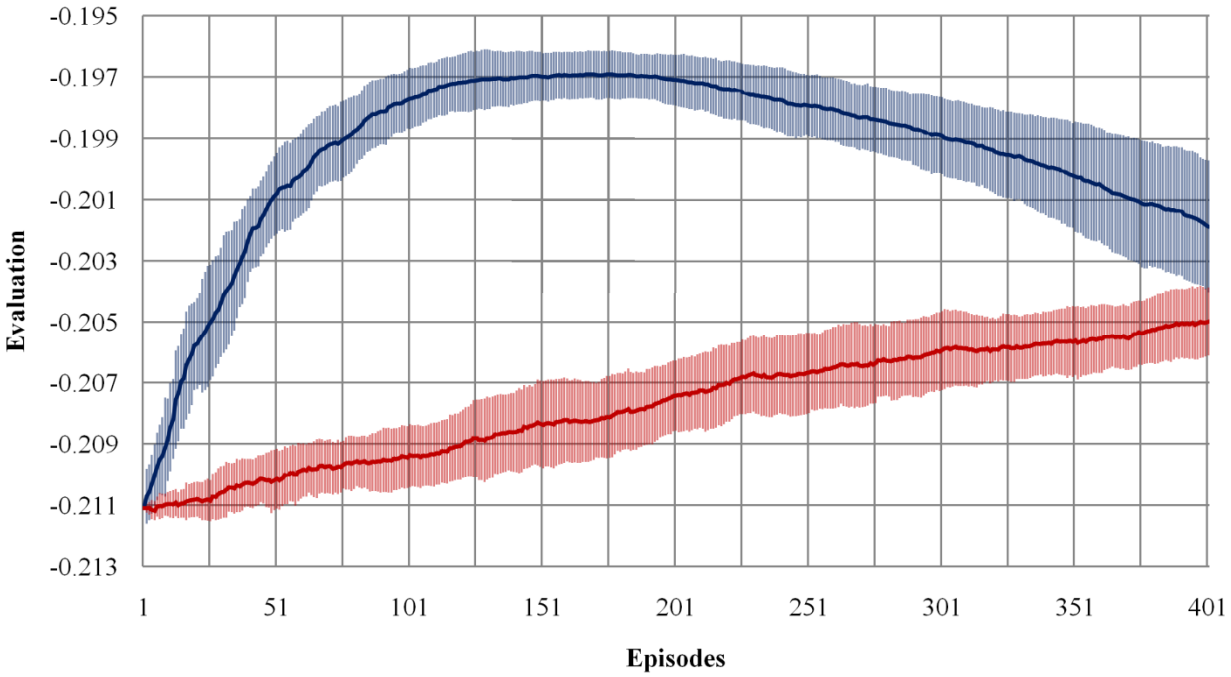


Figure 5.9: Mean performance (N=16) of parameter sets A and B on the CT with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis.

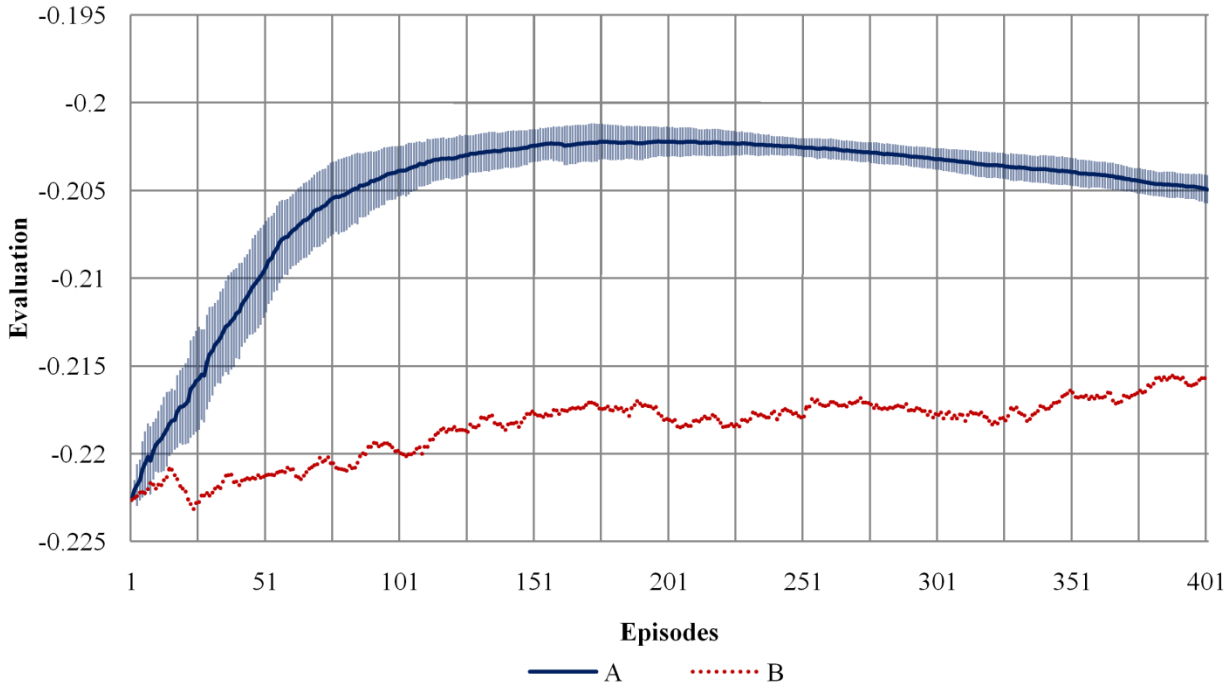


Figure 5.10: Mean performance (N=16) on the FTT with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis.

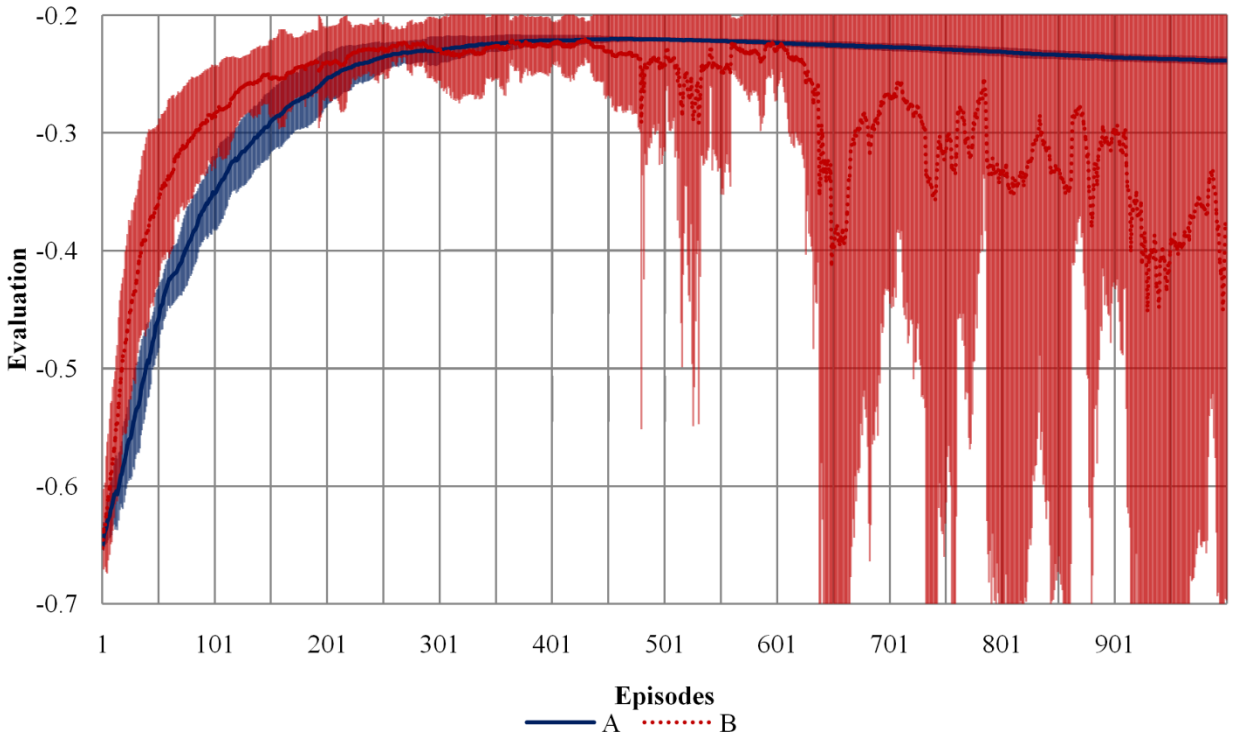


Figure 5.11: Mean performance (N=16) on the BBT with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis.

Notice that parameter set B performs similarly to A on the BBT, but worse on the CT and FBT. This is likely because the optimization of the parameter sets judged parameter utility based on performance on the BBT. Parameter set B may have over fit the problem of learning on the BBT.

The ability of the actor-critic to learn well with various amounts of exploration on the simulated arm is encouraging and potentially useful in clinical application. When used with a human arm, there will be unintentional noise introduced to the system by sensors, as in the NRT. Parameters for exploration ought to be chosen to have just enough exploration that the agent can distinguish between the intended exploratory noise and the unpredictable noise inherent to real-world experiments. In the NRT, we only hypothesize about what such experimental noise will

be, while these tests show that parameter sets with more exploration, which may be required in a noisy environment, are also able to adapt rapidly to changing dynamics.

5.7 Noise Robustness Test (NRT)

The system performs well on the NRT using the Fast parameters, without significant changes to learning speed with noise in the inputs representative of those expected in real-world experiments. As with the Fast parameters on the BBT without sensor noise, the system is not stable in the long-term. Figure 5.12 shows the rapid initial learning of the Fast parameters on the NRT without bias, combined with the BBT.

Adding a bias of size $\mu_B = .05$ to both joint angle measurements improves the initial evaluation on the BBT, but does not have a significant impact on learning, as shown in Figure 5.13. Tests (not shown) have suggested that larger biases will often cause the system to immediately diverge. Researchers should therefore carefuly calibrate sensors prior to use.

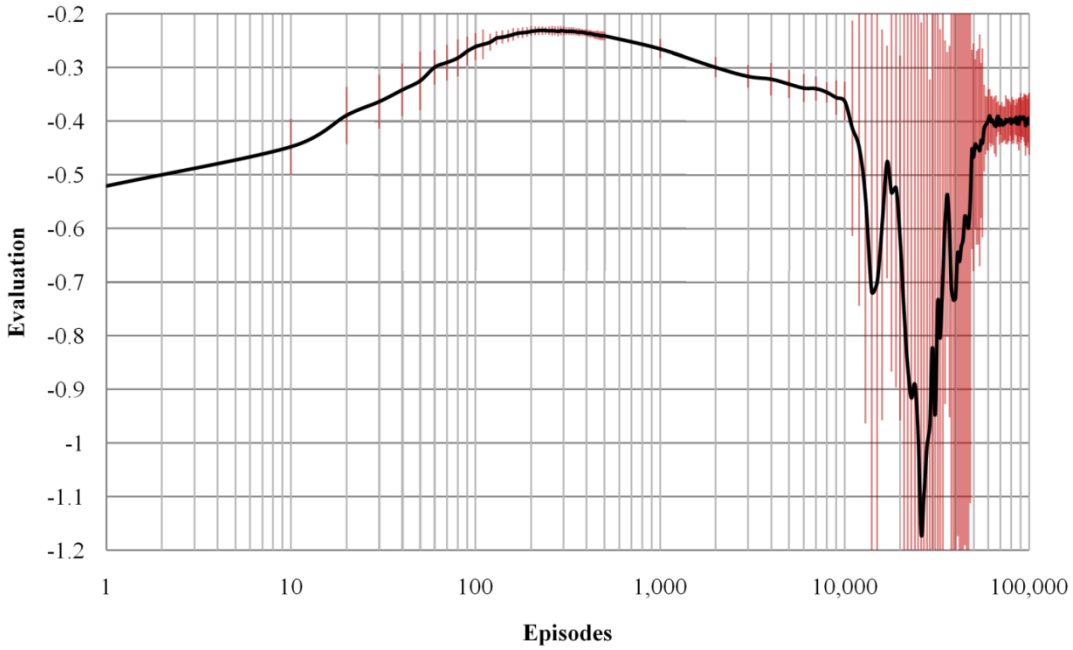


Figure 5.12: The actor-critic's mean evaluation (N=16) on the NRT with $\sigma_\theta = .1$, $\sigma_{\dot{\theta}} = .3$, and $\mu_B = 0$, with standard deviation error bars provided. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

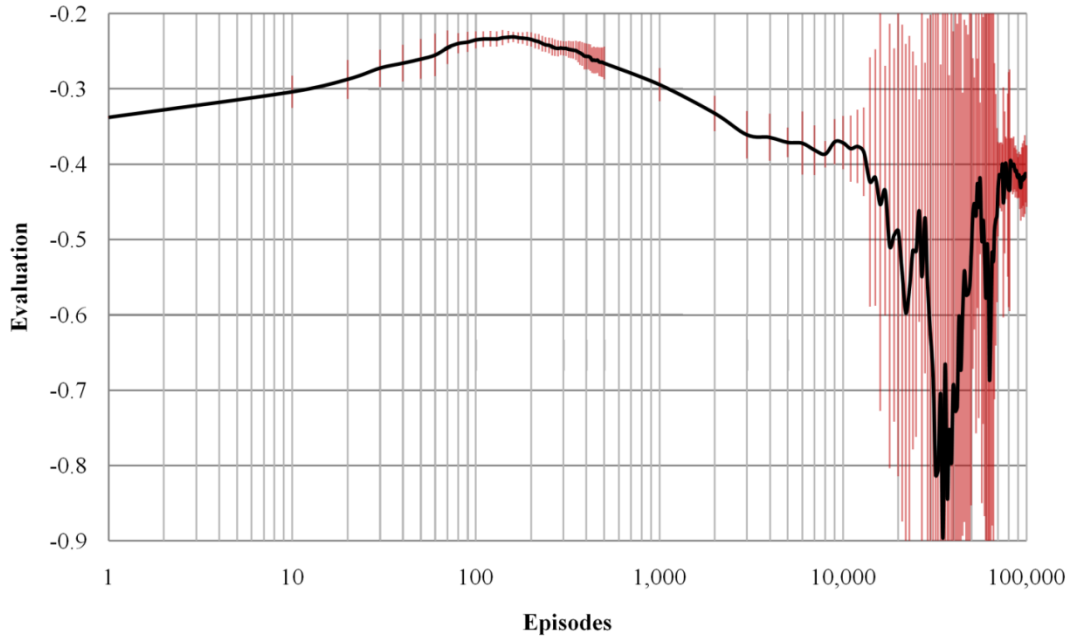


Figure 5.13: The Fast parameters' mean performance (N=16) on the BBT combined with the NRT including a bias, with standard deviation error bars provided. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

5.8 Delayed Reward Test (DRT)

Running the DRT with $\kappa_\tau = 20$ simulates a setting in which the user is only able to give rewards once every .4 seconds. For some amount of time, *t*, in seconds, the actor and critic would usually be updated $\left\lfloor \frac{t}{\Delta t} \right\rfloor$ times, however, on the DRT there would only be $\left\lfloor \frac{t}{\kappa_\tau \Delta t} \right\rfloor$ updates, so it is expected that learning will be slower on the DRT. To some extent, this can be combated using larger learning rates in the actor, as shown in Figure 5.14.

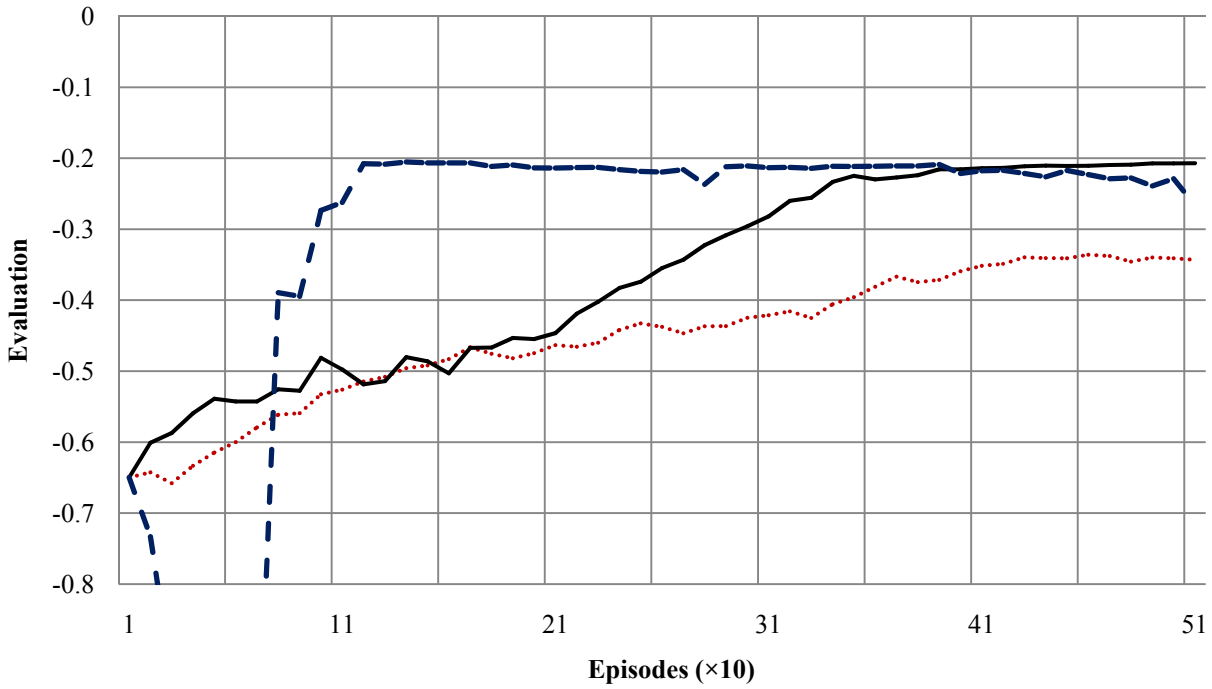


Figure 5.14: Typical runs of the Fast parameters on the DRT combined with the BBT using $\kappa_\tau = 20$. Notice that the horizontal axis is episodes times ten, giving the plot a duration of 500 episodes. Evaluations represent those just prior to the episode marked on the horizontal axis. The red (dotted) line is the actual Fast parameters ($\eta_A = 70$), the black (solid) line is the Fast parameters with $\eta_A = 140$, and the blue (dashed) line is the Fast parameters with $\eta_A = 1,400$. The latter reaches a minimum at -1.3 after 80 episodes (not shown).

In conclusion, we expected and observed slower learning when using delayed rewards, though this can be partially avoided by increasing the learning rates in the actor. During trials in which a human subject provides the rewards to the system, either the rewards must be provided frequently, or learning will be slow. One possibility for achieving rapid learning while maintaining human influence in the reward signal would be to allow the human to augment the computer signal, which is still provided every .02 seconds.

5.9 Discrete Reward Test (DiRT)

The actor-critic performs well on the DiRT combined with the BBT, described in Section 4.9. Figure 5.15 shows how learning is quite similar to that of the BBT alone when using the Fast parameters, though in the long-term the system diverges to an even lower evaluation on the DiRT.

Figure 5.15: Mean performance (N=16) of the Fast parameters on the BBT combined with the DiRT for 100,000 episodes, with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

5.10 Continuous Learning Modification (CLM)

When training on the CLM, the eligibility traces in the critic were reset after each movement. This was done because the values of subsequent states are independent of the actions taken and states crossed during the previous movement. Figure 5.16 shows performance on the CLM combined with the BBT. Though the system is still learning, it is not as reliable as with the Fast parameters. Visualizing the policies after 500 episodes of training, they are not as smooth as those found without the CLM, and also include more oscillation about the target state. Though the actor-critic has accomplished its goal of increasing the accumulation of reward, the resulting policies would likely be considered worse by a human observer.

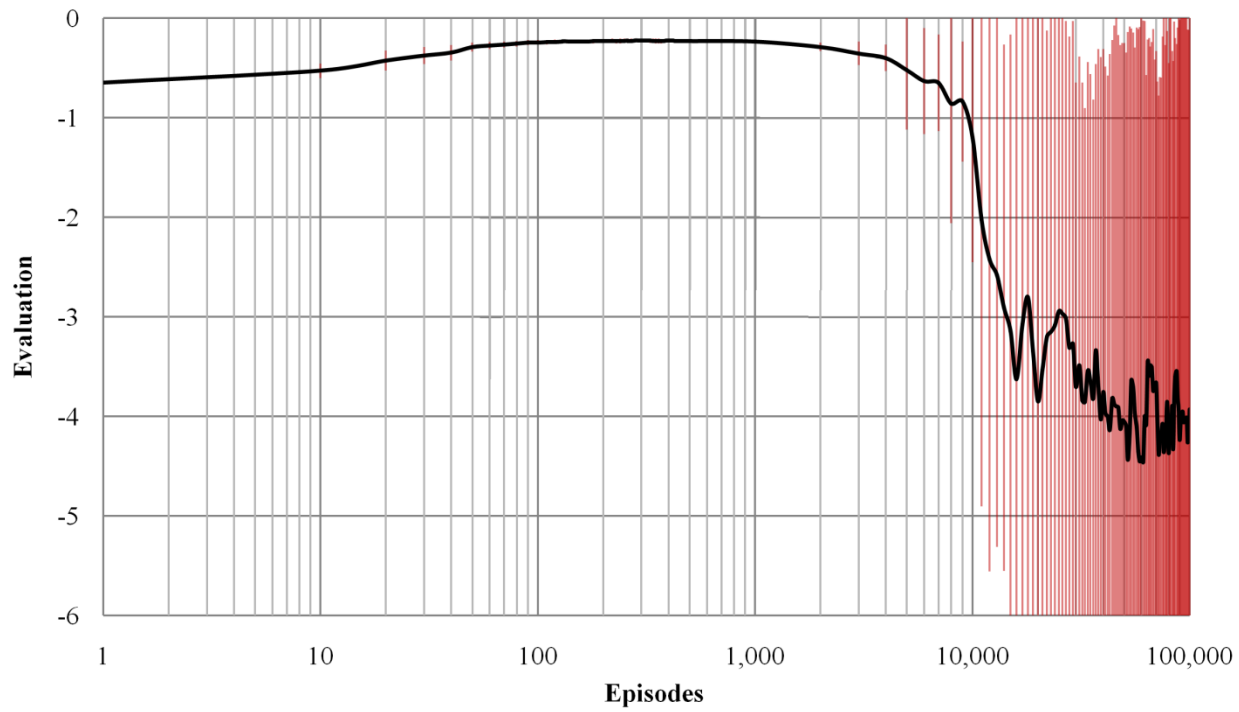


Figure 5.16: Mean performance (N=16) of the Fast parameters on the BBT with the CLM over 100,000 episodes, with standard deviation error bars. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis. The maximum evaluation of $-.23$ occurs after approximately 500 episodes.

5.11 An Unexplained and Unexpected Phenomenon

It was accidentally observed during trials using the Fast parameters on the BBT (Section 5.4) that the system learned on the BBT and FTT whenever the TD-error (δ from Equation 2.39) is negative, though learning is less stable than when δ is computed using Equation 2.39. To exemplify this, consider Figures 5.17 and 5.18, in which the Fast parameters were run on the BBT (Section 4.3). In Figure 5.17 the TD-error was selected randomly over the interval $\delta \in [-.5, 0]$, while Figure 5.18 is the unmodified actor-critic (similar to Figure 5.5, except using 16 new trials and a different horizontal scale).

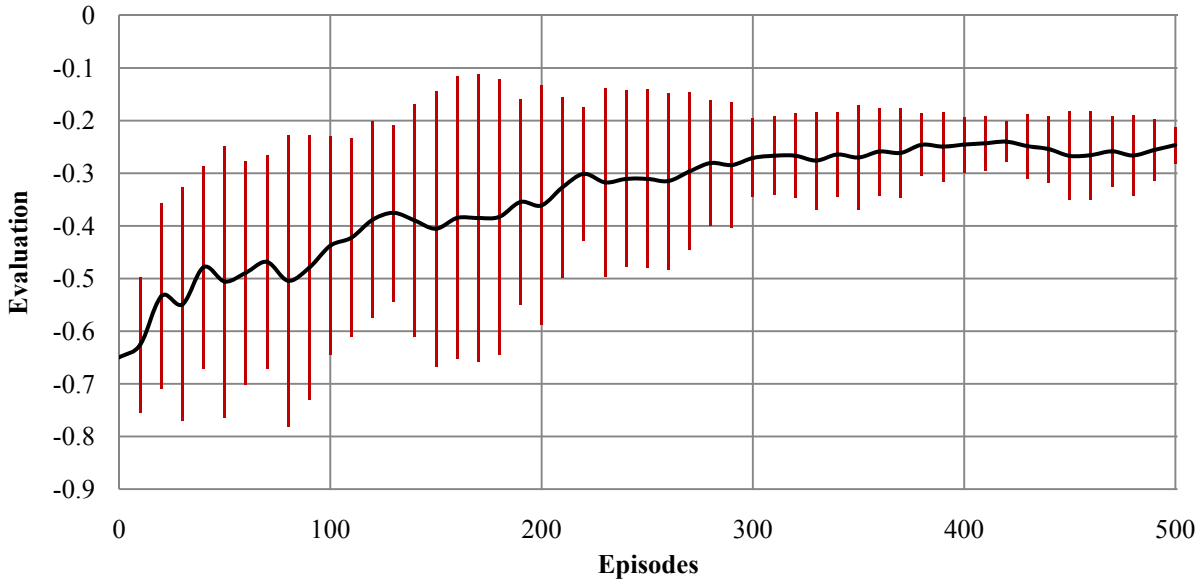


Figure 5.17: Mean performance (N=16) of the continuous actor-critic on the BBT using the Fast parameters with random negative TD-error. Standard deviation error bars are provided.

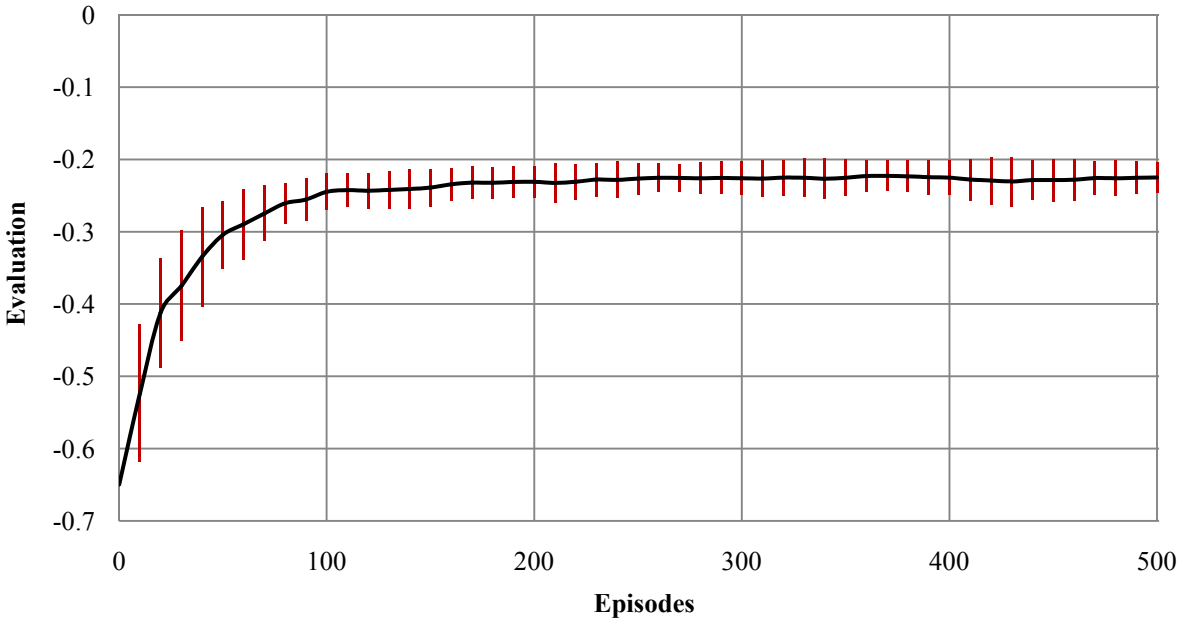


Figure 5.18: Mean performance (N=16) of the continuous actor-critic on the BBT using the Fast parameters with TD-errors computed as described in Subsection 2.2.8. Standard deviation error bars are provided.

Figure 5.19 shows that the TD-error initially has a negative bias during a typical run of the unmodified actor-critic using Fast parameters on the BBT (e.g., a run from Figure 5.5 or Figure 5.18). Over time, the bias decreases. Figure 5.19 spans 10,000 TD-error computations, which equates to 100 episodes.

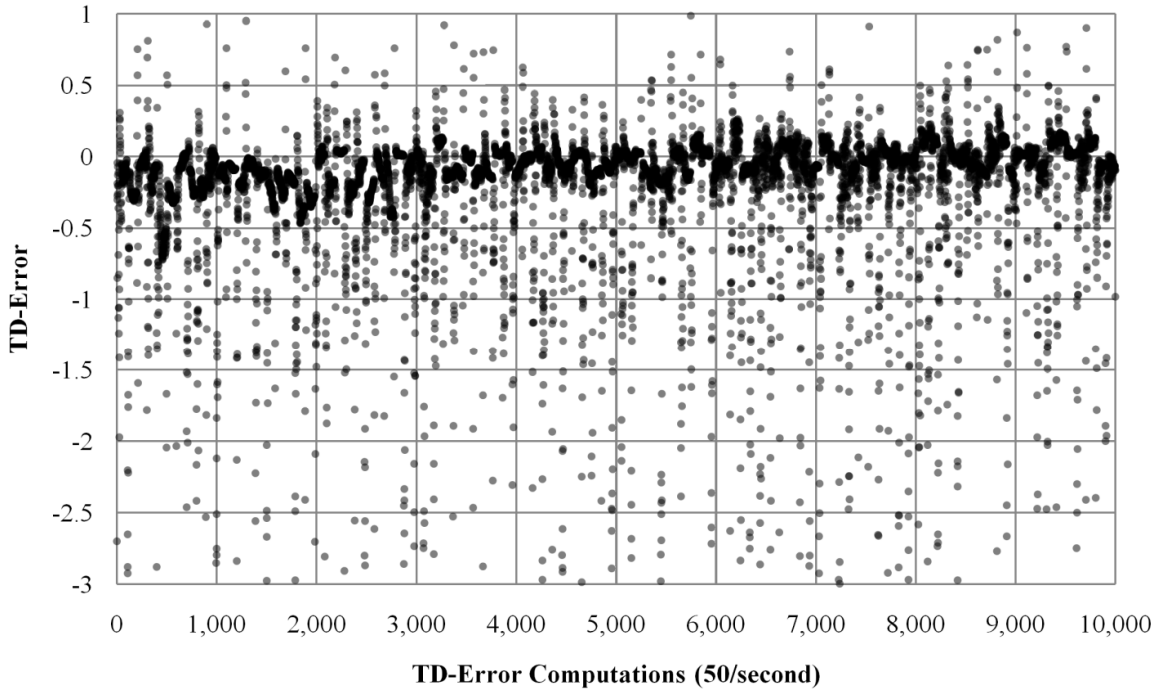


Figure 5.19: TD-errors from a typical run of the unmodified actor-critic using Fast parameters on the BBT.

Trials in which the TD-error was random with no bias or had a positive bias all diverged rapidly. We observed that the exploration has a slight positive bias because the requested stimulation of each muscle is near zero for most of each episode. When these small values are combined with the zero-bias exploration and put through the Sigmoid function to obtain the action (Equation 2.43), the result is a slightly positive bias in the exploration applied to the

muscle stimulations. One would expect this positive bias, combined with negative TD-errors, to result in a general decrease in muscle stimulation, which may improve performance on the BBT and FTT by reducing overshoot. However, empirical tests revealed that the sum of the muscle stimulations from each episode increased for all muscles but the triceps (short and long head) on the BBT when using the unmodified Fast parameters.

It remains unknown why the system learns when the TD-error is negative and randomly selected.

CHAPTER 6:

LONG-TERM STABILITY

In all of the previous tests from Chapter 5, parameters were found to either be fast and unstable or slow and stable. This chapter presents several techniques applied in an attempt to achieve both in one controller. It begins in Sections 6.1 through 6.4 with simple modifications to the actor-critic architecture and implementation. Section 6.5 presents a system that achieves both rapid initial learning and long-term stability. Section 6.6 concludes the chapter with a summary of the results.

6.1 TD-Error Cap

In the previous tests, the magnitude of the TD-error was capped to .5 during training because larger TD-errors, though uncommon, could cause the critic to become unstable. This occurs because the magnitude of the TD-error scales the learning rate for the actor and critic linearly (see Equations 2.41 and 2.42). By lowering this cap, the system is forced to make smaller updates. This improves stability, but slows learning.

Figure 6.1 shows that the tradeoff between stability and learning speed was not significantly improved by changes to the TD-error cap. Notice that the horizontal axis is scaled by 1,000, showing 100,000 episodes.

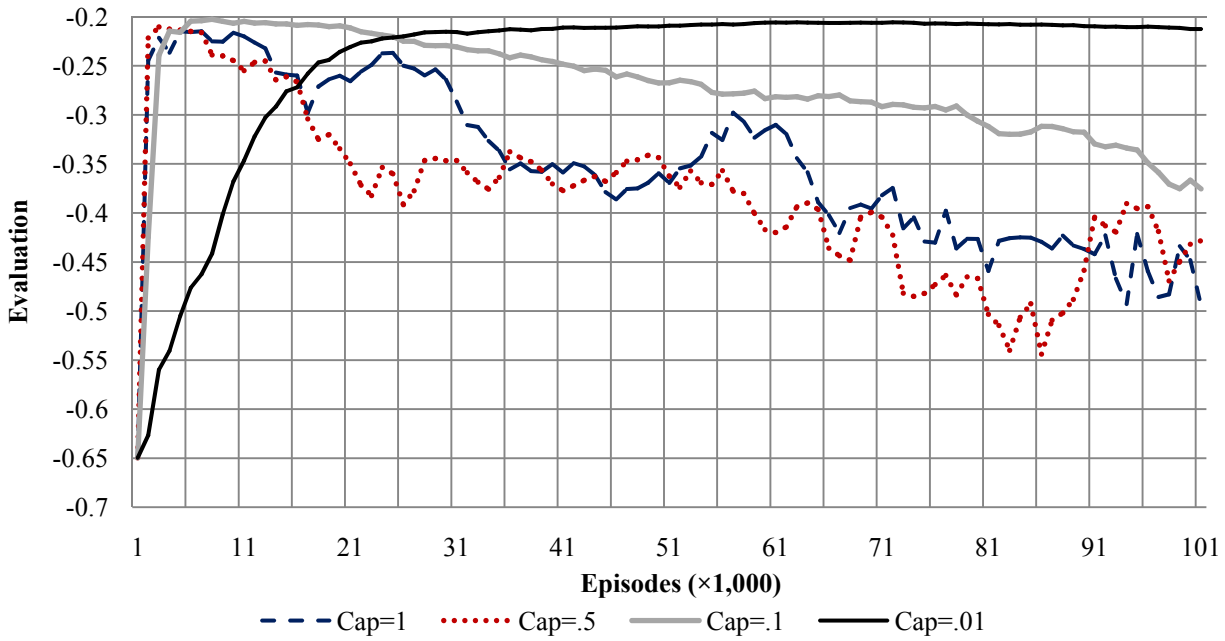


Figure 6.1: Evaluation on the BBT for 10,000 episodes using the Fast parameters with various TD-error magnitude caps. After 100,000 episodes, the curve for Cap=.01 reaches −.3. The horizontal axis is scaled by 1,000.

A cap small enough to ensure stability would have to be lower than .01, which already adapts too slowly for practical applications. There is not a significant difference between a cap of 1 and .5 because TD-errors were rarely larger than .5 in our trials.

6.2 Muscle Force Weight

When the system is diverging, it first begins to oscillate at high frequency around the goal state. A possible cause is an improper weighting of the squared muscle activation, which relates to muscle forces, in Equation 4.1. The constant W was changed to $W' = kW$, with $k \in \{.8,1,1.2,1.5,1.75,2,3,4,5,10\}$. The larger values result in a higher weighting on muscle forces, which encourages slower motions that require less muscle force, which may reduce the

jitter that precedes divergence. Figure 6.2 presents the performance with these *k*. Notice that none of the systems are stable in the long-term.

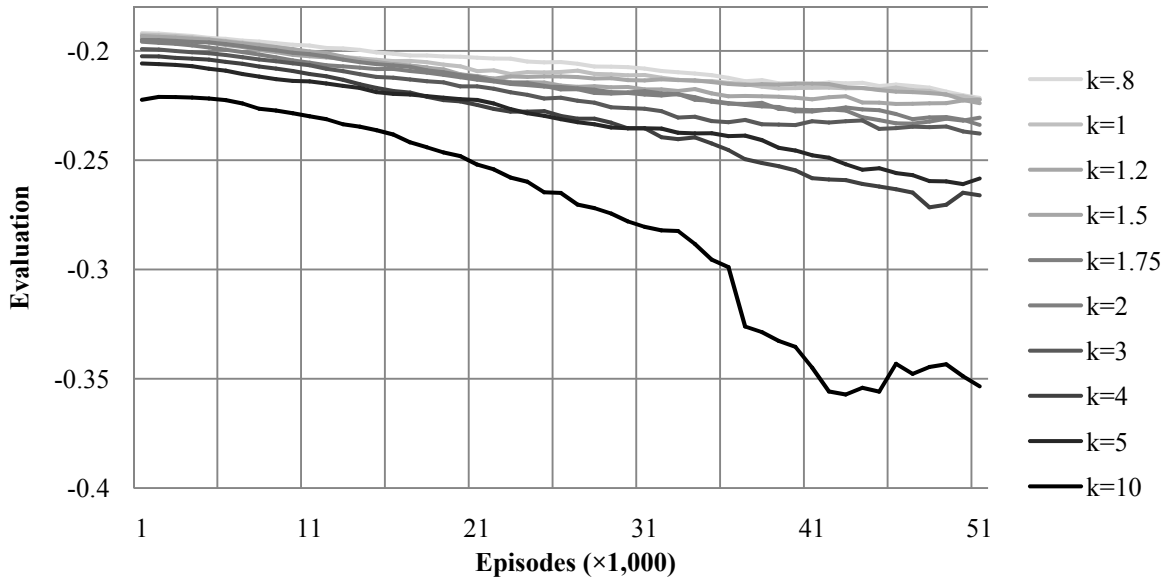


Figure 6.2: Performance on the BBT with Fast parameters and with various muscle weight constants in the reward function. Notice that the line colors are graduated with respect to *k*, with darker lines corresponding to larger *k*. The horizontal axis is scaled by a factor of 1,000.

If the constant, *k*, is made too much smaller, the evaluation may increase, but performance may not, as the controller will become more like a PD controller with excessive gains. Also, notice that the evaluation metric in Figure 6.2 differs for each curve. With a larger constant multiplying the muscle forces, the rewards are all more negative. Because the evaluation is based on the integral of the reward signal, this results in a worse evaluation for an identical policy. Initially all of the curves start with the same policy, so the difference in their evaluations is due to this discrepancy in evaluation.

In conclusion, changes to this constant were found to visually influence the magnitude and frequency of the jitter, though its onset was relatively constant and divergence properties remained unchanged, as seen in Figure 6.2. The critic was not pre-trained again for each *k*, which could be adversely affecting performance.

6.3 Monitor Critic

Assuming that divergence occurs because of error in the value function, a possible solution is to only update the actor only when the TD-error over the previous *k* updates has been less than a manually tuned constant, Δ. Tests showed the standard system to be relatively stable on the BBT when TD-errors had magnitude less than .1, suggesting $\Delta \approx .1$. The trade-off between stability and learning speed was again not significantly changed. For small Δ and large *k*, the system was more stable, though learning was slow, while larger Δ and smaller *k* learned faster but were more unstable.

The learning curves in Figure 6.3 use a combination of the Fast and Slow parameters, in which $\eta_A = 70$ and $\eta_C = .344$, when run on the BBT with varying Δ and $k = 20$. Though the smallest Δs are more stable, they require over 2,000 training episodes for initial adaptation, far more than the target 200 episodes. In addition, they too are unstable in the long-term. This is only a small sample of the trials run, which included various *k* and ANN sizes.

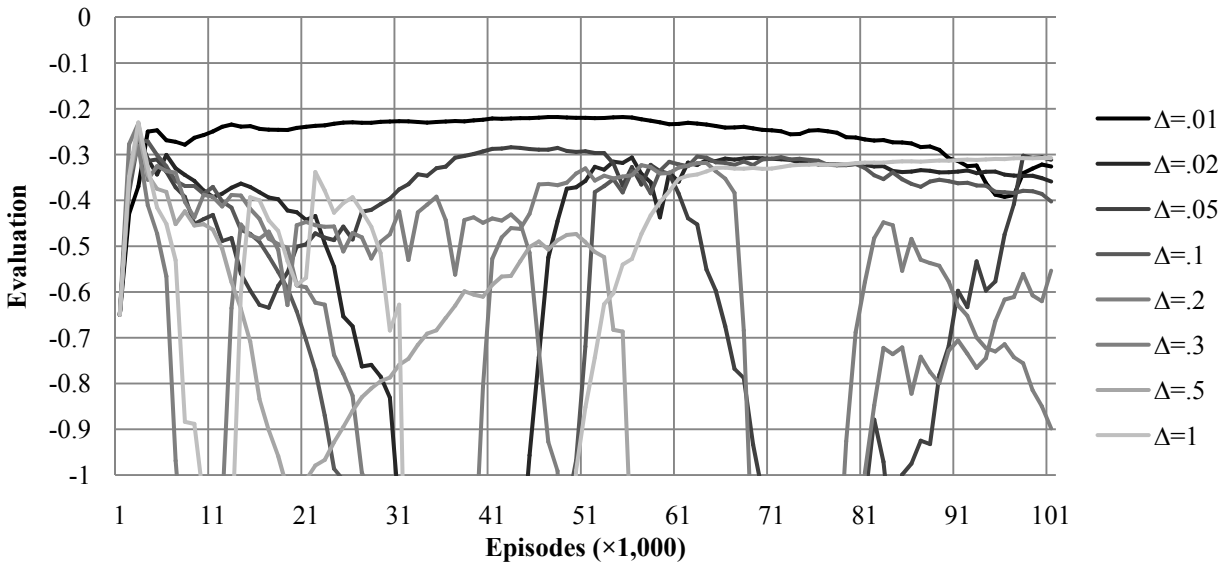


Figure 6.3: Performance of the merged Fast and Slow parameters on the BBT with various TD-error caps and $k = 20$. Notice the color gradient follows the value of Δ, and the horizontal axis is scaled by a factor of 1,000. The line with the best evaluation over most episodes corresponds to $\Delta = .01$.

A possible reason for the failure of this approach is the variance in TD-errors depending on arm motions and positions in the state space. A small average TD-error over an entire episode could either mean that the critic is accurate over the entire state space or that the critic is accurate only for the motion in the previous episode (e.g. a relatively small movement). However, a small TD-error over a duration less than an episode could mean that the critic is accurate only for certain regions of the state space (e.g., when the arm is near the goal, giving expected future reward near zero). To overcome these hurdles would require values of k that average TD-errors over more than one episode.

Figure 6.4 shows results with $k = 200$, which equates to 4 seconds, or two arm movements. As with the smaller value of k, the system remains unstable in the long-term.

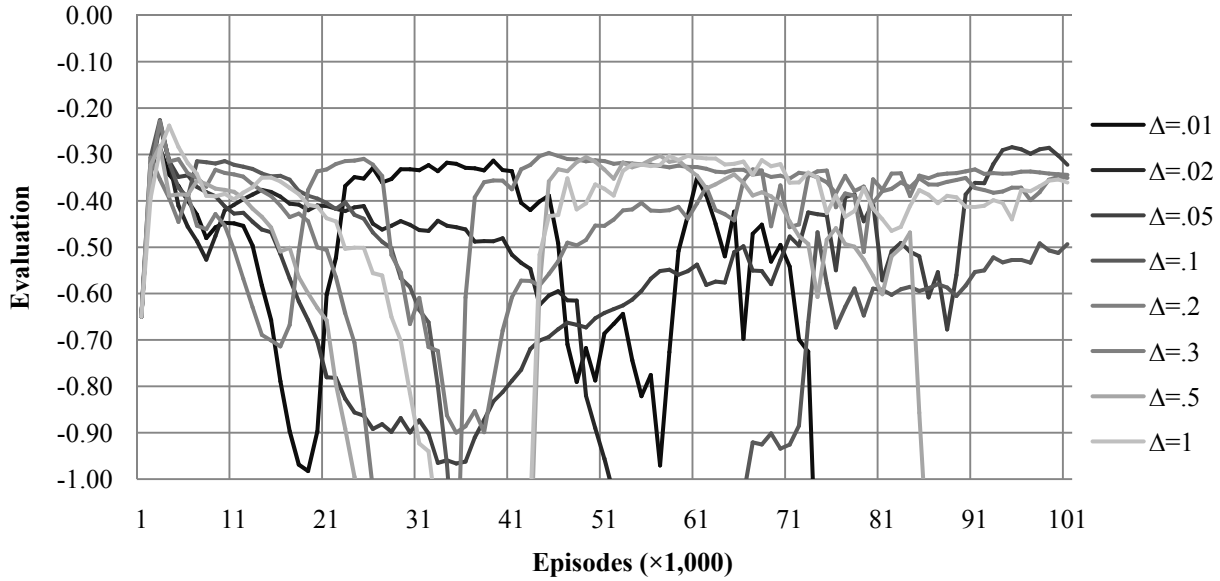


Figure 6.4: Performance of the merged Fast and Slow parameters on the BBT with various TD-error caps and $k = 200$. Notice the color gradient follows the value of Δ and the horizontal axis is scaled by a factor of 1,000.

6.4 Weight Decay Term

It is common to add a weight decay parameter to the objective function when training function approximators in order to improve generalization (Mitchell, 1997). This can be approximated for the continuous actor-critic by augmenting the actor and critic update equations (2.42 and 2.41 respectively) to

$$\dot{w}_i = \eta_C \delta(t) e_i(t) - k_C w_i^2, \tag{6.1}$$

and

$$\dot{w}_i^A = \eta_A \delta(t) n(t) \cdot \frac{\partial A\left(x(t); w^A\right)}{\partial w_i^A} - k_A \left\| n(t) \right\| \left(w_i^A\right)^2, \tag{6.2}$$

respectively, where k_C and k_A are weighting constants. In some implementations, the expected magnitude of the vectors $n(t)$ and $N(t)$ are decayed over time. If the magnitude of the explorational noise term were not included in the weight decay term of Equation 6.2, it would dominate the equation as the magnitude of the exploration goes to zero, taking all weights to zero.

Previous tests can be thought of as having k_C and k_A both set to zero, resulting in no weight decay term. The General parameters are defined in Table 6.1 as a combination of the Fast and Slow parameters, with the addition of $k_A = .0000002$ and $k_C = .000002$. These values were found through experimentation to result in weight magnitudes several orders smaller than those derived from training without a weight decay parameter, while the overall performance was not significantly changed on the control test.

η_A	η_C	k_A	k_C
70	.344	2E – 7	2E – 6

Table 6.1: General parameters. Those not listed are identical to the Fast parameters, provided in Table 5.2.

A new actor and critic were pre-trained using these parameters on the Control Test for 10,000 episodes. The resulting actor and critic were then used as a starting point for the CT, BBT, and FTT. The General Parameters were given their name because the resulting policy of the newly pre-trained actor generalizes well to different arm dynamics. The pre-trained ANN's performance is provided in Table 6.2.

Test	CT	BBT	FTT
Evaluation	−.192	−.22	−.197

Table 6.2: General parameters' evaluations immediately after pre-training.

This result is expected, as weight decay terms are known in machine learning to improve generalization. These parameters use larger muscle forces, similar to a PD controller with larger gains, which improves initial performance on the BBT and FTT. Though these are mostly desirable traits, the long-term stability of the system remains unchanged.

Figure 6.5 depicts the General parameters' evaluation on the CT. Notice that the initial evaluation is −.19, which is similar to that of the PD controller on the CT (−.18). Figure 6.6 depicts the General parameters' performance on the FTT. Notice that the initial evaluation of the General parameters on the FTT (−.197) is better than that of the original pre-trained ANN of Chapter 5 for the Fast and Slow parameters on the CT (−.21). Figure 6.7 depicts the General parameters' performance on the BBT. Once again, notice that the initial evaluation of the General parameters (−.22) is only slightly worse than that of the pre-trained ANN for the Fast and Slow parameters on the CT (−.21).

The good initial evaluations in Figures 6.5, 6.6, and 6.7 support the notion that the General parameters generalize well to variations in arm dynamics, though all figures display divergent properties after approximately 1,000 episodes. Due to the lack of improvement in stability, the General parameters were not investigated further.

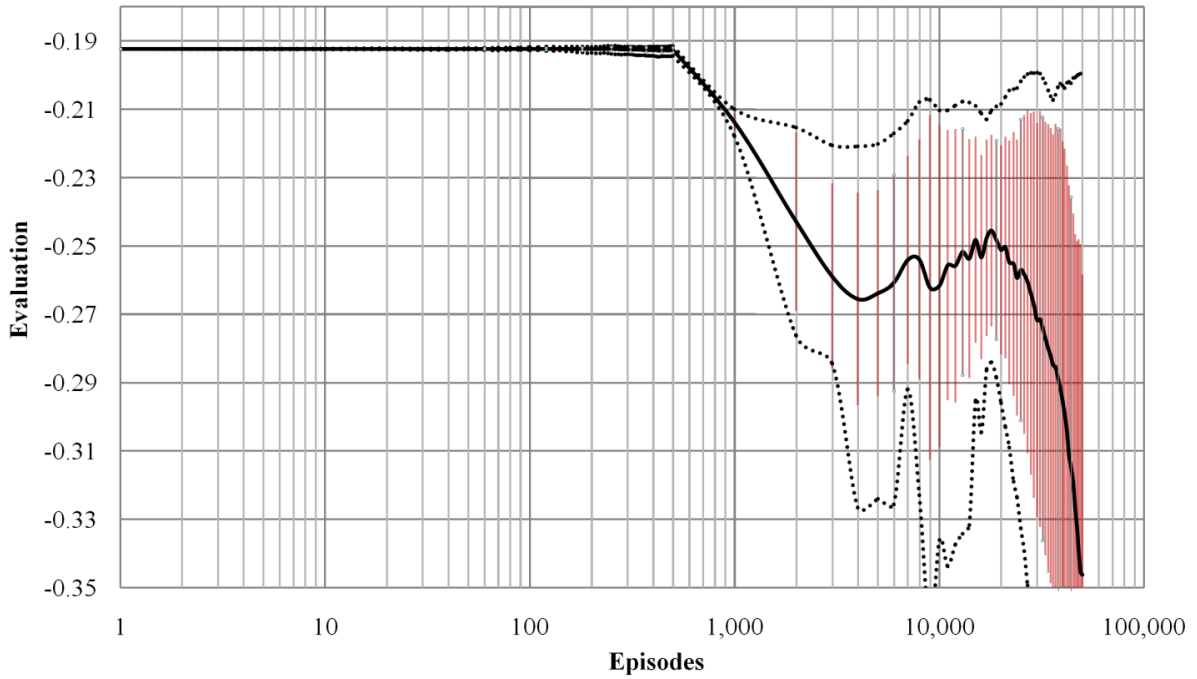


Figure 6.5: General parameters' mean performance (N=16) on the CT with standard deviation error bars provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis.

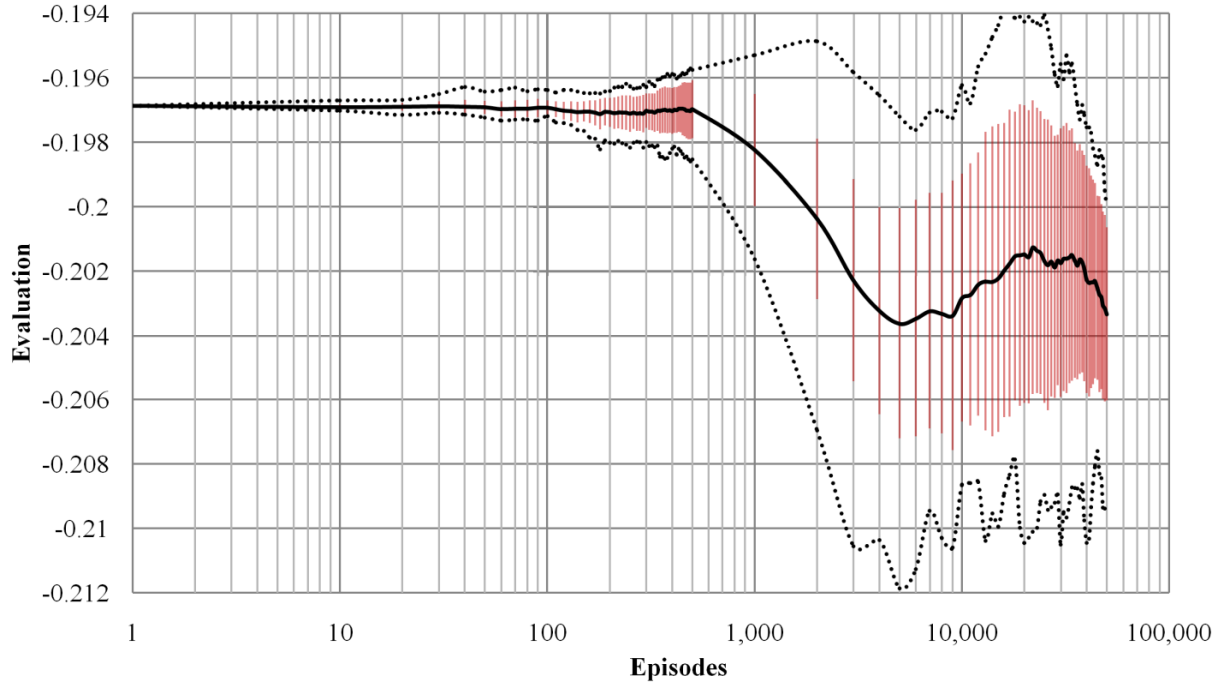


Figure 6.6: General parameters' mean performance (N=16) on the FTT with standard deviation error bars provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis. Also, notice the similarity of the line to that in Figure 6.5, and the differences in standard deviation.

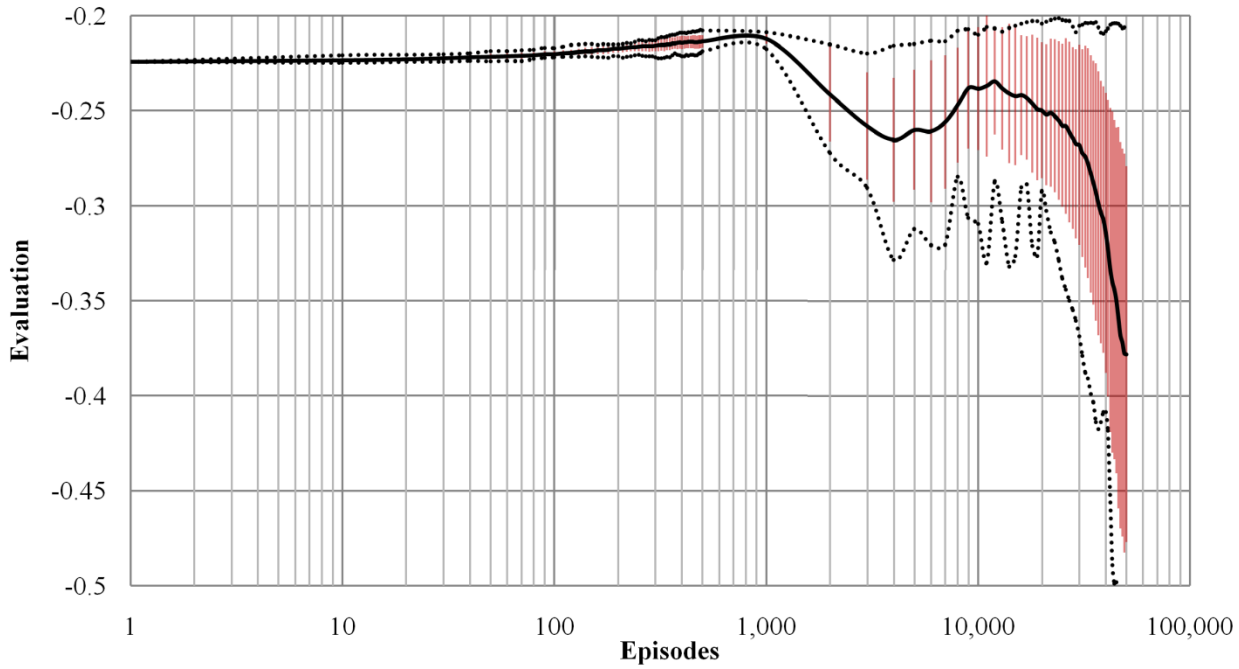


Figure 6.7: General parameters' mean performance (N=16) on the BBT with standard deviation error bars provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis.

6.5 Hybrid Controller Achieving Fast Learning and Long-Term Stability

For the FES control task, the agent must continuously adapt to changes in its environment, so decaying learning rates is not a viable option. However, the rapid initial learning with an inaccurate critic can be combined with the slower accurate-critic learning using a hybrid controller that toggles between learning styles. The Fast parameters (Table 5.2) can be used for rapid initial learning. Once learning has reached a plateau, or if performance begins to decrease, the agent can switch to the Slow parameters (Table 5.2). During this phase, the critic corrects errors in the value function, again becoming accurate. Whenever performance deteriorates beyond some performance threshold, the system can again switch to the Fast parameters for another burst of rapid learning.

We are primarily interested in determining whether or not the Slow parameters will remain stable after beginning training with the Fast parameters, which results in an inaccurate critic. Also, in order to continuously adapt to changing dynamics, the system must be able to switch back to the Fast parameters and continue to learn when performance degrades. This toggling system will be referred to as the *Hybrid Controller*.

In practical applications, the Fast parameters can be used for initial adaptation when the agent is first used on a subject, after which the Slow parameters can be used to maintain stability (e.g. Figures 6.8, 6.9, and 6.10). At any point, if a subject notices deteriorated performance of his or her arm due to muscle fatigue or other changes, the subject could activate a short-term switch to the Fast parameters to improve performance via shape-greedy learning.

Figure 6.8 shows that this hybrid controller can learn quickly and remain stable in the long-term on the CT and FTT. The results of these two tests were combined into one figure because of their similar evaluation magnitudes. Notice that, on the FTT, the actor-critic's final evaluations are better than those of the pre-trained ANN on the CT.

Figure 6.9 shows that the Hybrid Controller also performs well on the BBT with rapid initial learning and long-term stability. The slight decrease in performance around 10,000 episodes is consistent throughout the tests, as shown by the error bars, but remains unexplained.

Figure 6.10 shows the performance of the Hybrid Controller on the NRT ($\mu_B = .05$). The rapid initial learning is identical to that of the Fast parameters, however, unlike Figures 6.8 and 6.9, the system is not completely stable in the long-term, though it remains stable for nearly 10,000 episodes. For real-world FES applications, instabilities that arise after 10,000 episodes

are irrelevant. However, the instability beyond 10,000 episodes is interesting from an academic standpoint, evincing that the system is not completely stable.

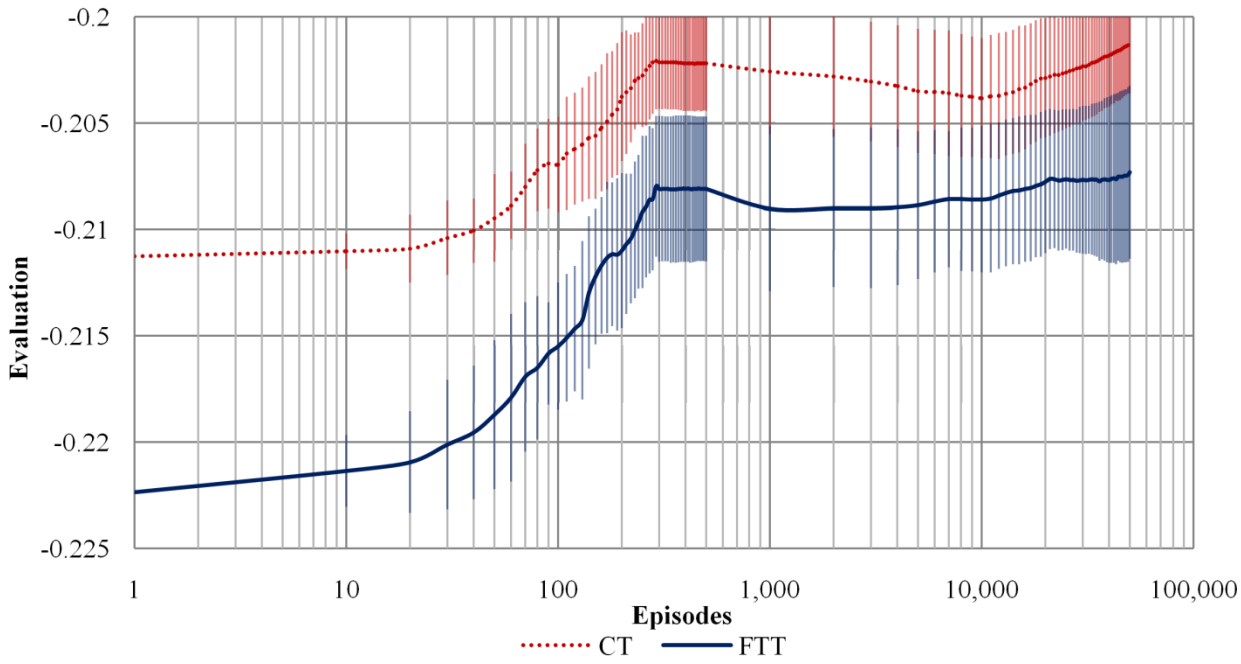


Figure 6.8: The actor-critic's mean evaluation (N=16) over 50,000 episodes on the CT and FTT using the Fast parameters for the first 300 episodes and the modified Slow parameters thereafter, with standard deviation error bars provided. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

For the Toggling Test (TT, Section 4.7), the parameters were switched to the Fast parameters whenever the environment switched dynamics between the BBT and the FBT. Figure 6.11 shows how the system can rapidly converge to a policy with a reasonable evaluation on both the BBT and FBT while remaining stable. Before the switch to the Slow parameters, the learning curve for the Hybrid Controller is identical to that of the Fast parameters. For a better view of initial learning than is provided, refer to Figure 5.5, which presents the results of the Fast parameters on the BBT.

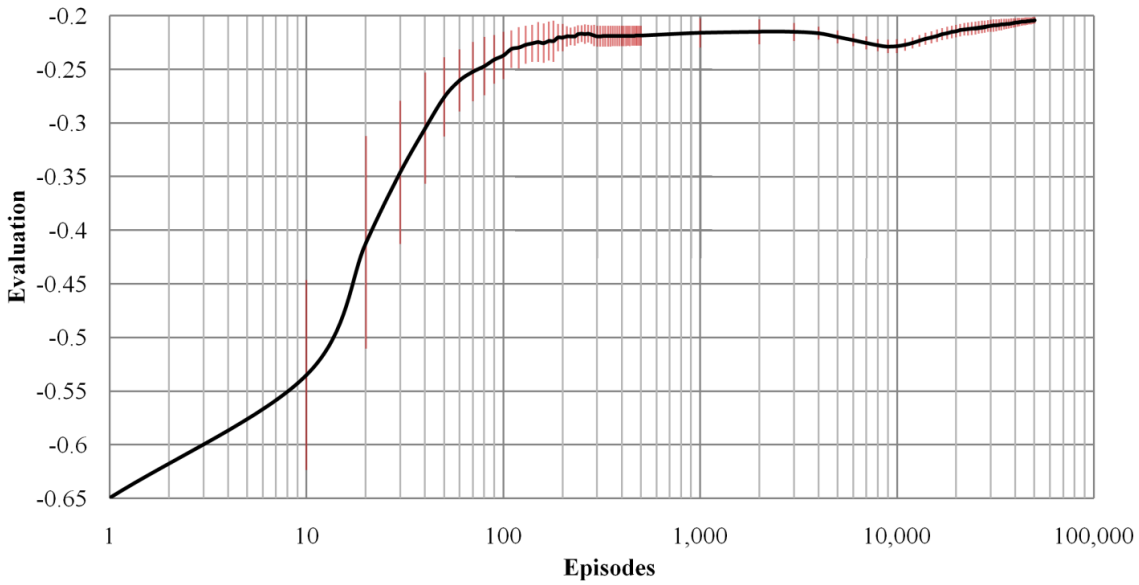


Figure 6.9: The actor-critic's mean evaluation (N=16) over 50,000 episodes on the BBT using the Fast parameters for the first 300 episodes, and the modified Slow parameters thereafter, with standard deviation error bars provided. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

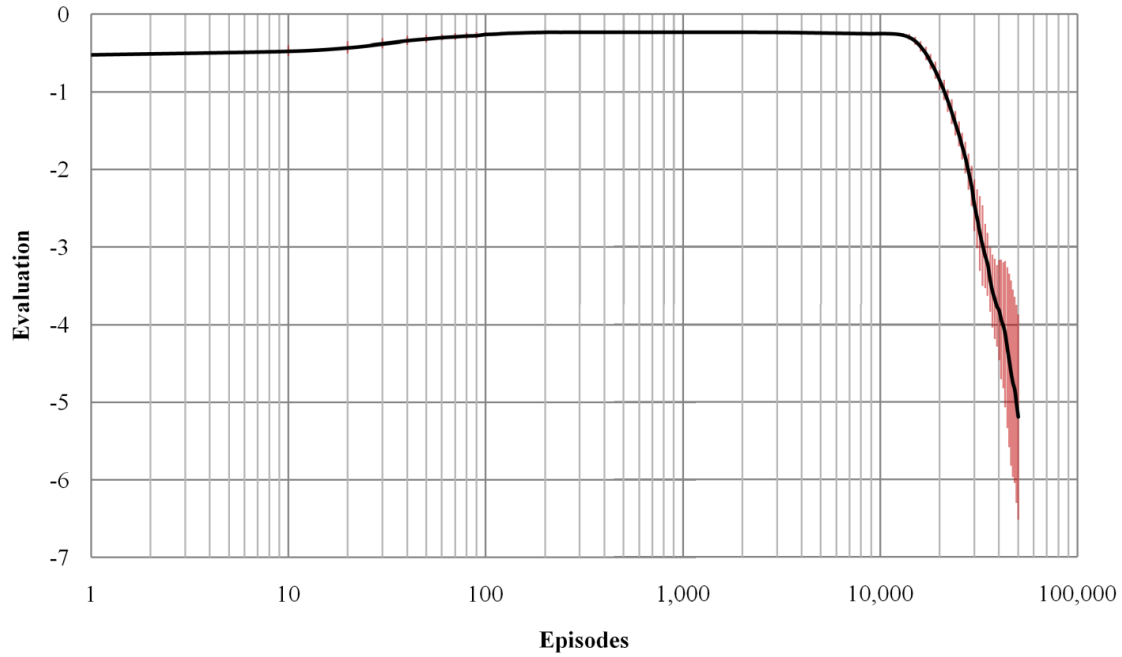


Figure 6.10: The actor-critic's mean evaluation (N=16) over 50,000 episodes on the NRT combined with the BBT, with standard deviation error bars. It used the Fast parameters for the first 300 episodes, and the modified Slow parameters thereafter. Evaluations represent those just prior to the episode number marked on the horizontal axis. Notice the logarithmic scale of the horizontal axis.

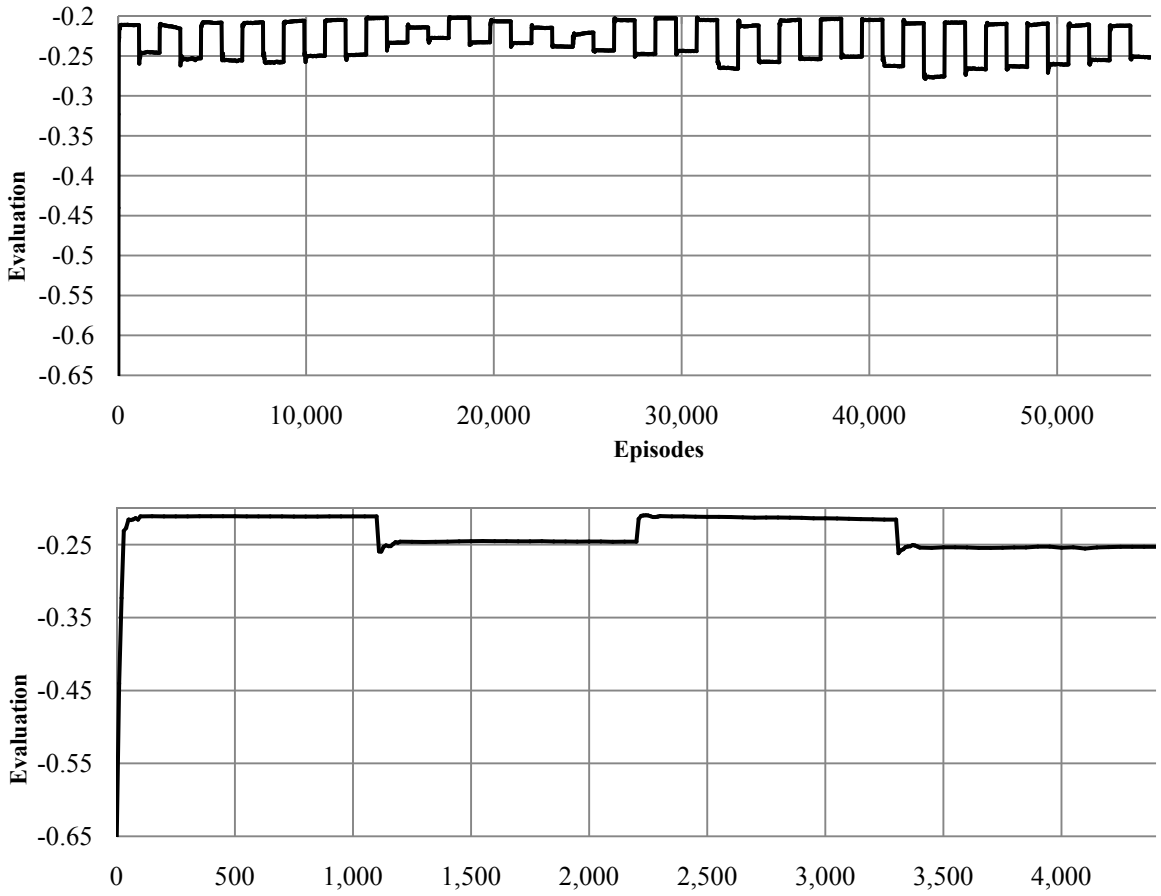


Figure 6.11: The hybrid controller's evaluation, where the environment starts as the BBT, then switches to the FBT after 1,100 episodes, then back to the BBT after 2,200 episodes, etc. The parameters also switch from the Fast parameters for the first 100 episodes on each test to the modified Slow parameters for the remaining 1,000 episodes on each test. The top plot shows long-term performance while the bottom shows short-term performance.

6.6 Conclusion

The initial attempts in Sections 6.1 (TD-Error Cap), 6.2 (Muscle Force Weight), 6.3 (Monitor Critic), and 6.4 (Weight Decay Term) all failed to improve long-term stability, though the addition of the weight decay term did result in improved generalizability of the policy to environments with variations in dynamics. In Section 6.5, we devised the Hybrid Controller,

which takes advantage of both the rapid initial learning of the Fast parameters and the long-term stability of the Slow parameters.

CHAPTER 7:
DAS1 ILWR-CRITIC RESULTS

When using ANNs for both the actor and the critic, applied to the Adaptive RL FES Controller Task (Section 1.2), the continuous actor-critic failed to accomplish the necessary conditions for success. We require the controller to have both rapid initial learning, as well as long-term stability. In Chapter 5, where we presented the results when using ANNs for both the actor and the critic, only one of these two conditions could be satisfied at a time. In Section 6.5, we accomplished both by modifying the continuous actor-critic to allow for well-timed switches between two different parameter sets.

The long-term instability of any one parameter set, when using ANNs for the actor and critic, may stem from the critic's inability to remain accurate as the actor changes. The critic's accuracy can be improved by changing the critic function approximator to one that is better able to accurately track a non-stationary function. Results from Chapter 3 suggest that Incremental Locally Weighted Regression (ILWR) would perform better than an ANN as the critic, allowing for more rapid changes to the actor with the critic remaining accurate. This chapter therefore focuses on implementing ILWR as the continuous actor-critic's critic, while using the same pre-trained ANN actor from Chapter 5.

Local actor updates may also improve the critic's ability to accurately represent the value function because changes to the policy would better reflect the local exploration. Though tests

with an ILWR actor were out of the scope of this thesis, it would be an interesting topic for future research.

7.1 Pre-Training

The ILWR critic was pre-trained to be accurate for the policy of the ANN actor from Section 5.1, which was trained via supervised learning to mimic the PD controller. To train the ILWR critic, the actor-critic was run using the *ILWR-Pretrain parameters*, provided in Table 7.1, with the actor's learning rate set to zero for 100,000 episodes on the CT. SI-ILWR was used due to constraints on computational time. Recall from Equation 2.56 that a typical state of the arm is $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, \theta_{\text{Goal}_1}, \theta_{\text{Goal}_2})$, which resides in $[-2,4]\times[-1,4]\times[-5,5]\times[-5,5]\times[-2,4]\times[-1,4]$. The initial knowledge points were distributed in a Sukharev Grid over this domain with 4 points across each dimension, resulting in $4^6 = 4,096$ total points. The initial output values for the knowledge points were chosen with a uniform distribution over $[-10, 0]$.

η_A	η_C	**D**	σ	τ_N	τ	κ
0	.1	$diag(.5,.5,.3,.3,.5,.5)$	9,000	2,400	.1	.1

Table 7.1: ILWR-Pretrain parameters. The critic's learning rate is the output learning rate. The input learning rate was zero.

All points with a weight larger than .1 were included in the regression, so long as no fewer than 10 points and no more than 1,000 points were included in each regression. The

average TD-error magnitude (Equation 2.39) over the second 50,000 episodes was .22, which is larger than the ANN critic from Section 5.1's average of .1. Figure 7.1 depicts the average TD-error magnitude over each of the first 32,000 of the 100,000 training episodes, and suggests that the system was nearly converged well before 50,000 episodes.

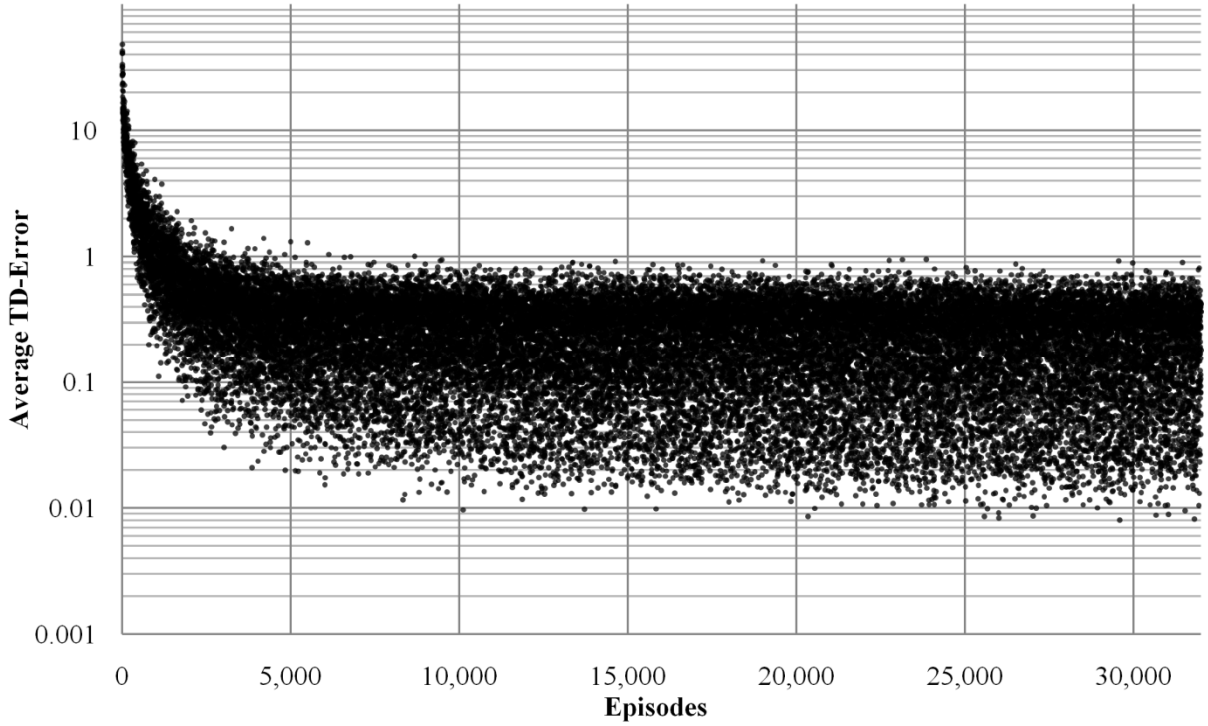


Figure 7.1: Average TD-error magnitude over the first 32,000 episodes when pre-training the ILWR critic. Notice the logarithmic scale of the vertical axis.

7.2 Parameter Optimization

Parameter optimizations for η_A, σ, τ_N, τ, and κ were performed in Section 5.1, and are relatively independent of the function approximator chosen for the critic. Their values were

therefore left unchanged, as in Table 7.1. The values for η_C, **D**, and the number of knowledge points were tuned manually. Unlike the optimizations of Chapter 5, here we are interested in finding parameters that result in both rapid initial learning as well as long-term stability.

There is no clear heuristic to allow a minimization algorithm to optimize both short and long-term performance because the tradeoff between the two is not yet known when using ILWR for the critic. Therefore, the optimizations were done manually by initially selecting reasonable parameter values and observing the system's behavior. Given the results of several tests, we determined what parameter changes would most likely result in improved performance. The best parameters found, called the *ILWR parameters*, are provided in Table 7.2.

η_A	η_C	**D**	σ	τ_N	τ	κ
70	.1	$diag(.5,.5,.3,.3,.5,.5)$	9,000	2,400	.1	.1

Table 7.2: ILWR parameters. The critic's learning rate, η_C, is the output learning rate. The input learning rate was zero.

Figure 7.2 gives some meaning to the values selected for **D**. If the relative weighting between dimensions were significantly off, discrete steps would appear. If the overall magnitude were too large, only a few points would have significant weights, and if it were too small, too many points would have significant weights. In order for the matrix inversion in LWR to succeed, at least 6 independent points must be included in each regression. In order to ensure that the system maintains real-time, we desire less than 1,000 points be included in each linear regression. As previously stated, points were included in the regression if their weights were above .1, so we desire between 6 and 1,000 points have weight above .1.

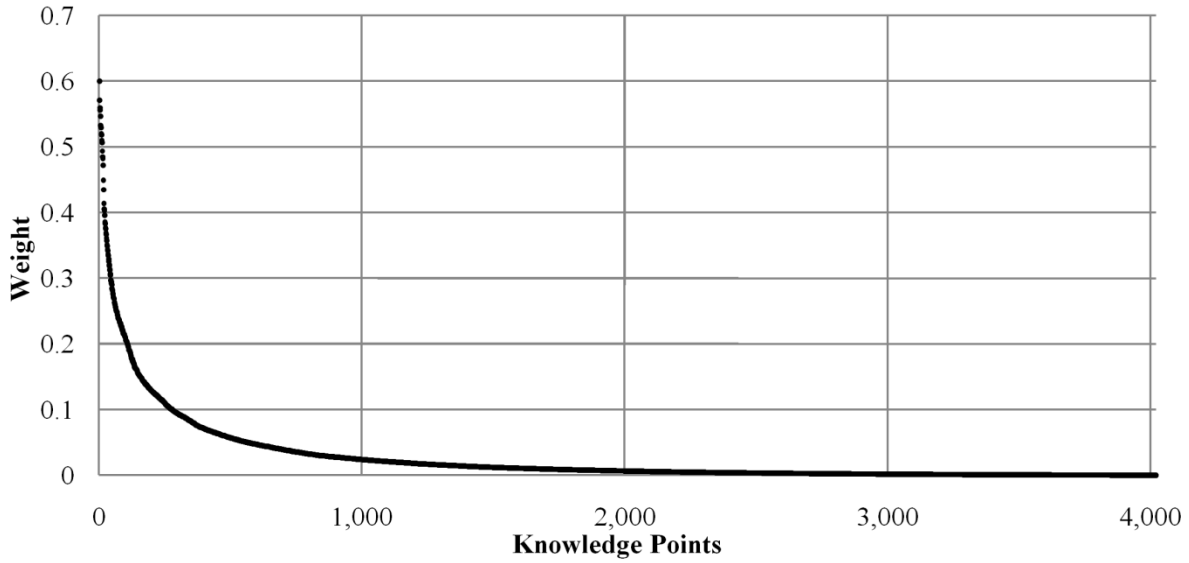


Figure 7.2: The weight of each knowledge point in the ILWR critic for the state (.82, .52, –.25, .78, .38, 1.5), which resides near the center of the query space. The knowledge points have been sorted along the horizontal axis from largest weight to smallest.

7.3 Control Test (CT)

The continuous actor-critic, when using the ANN-actor (Section 5.1) and ILWR-critic (Section 7.1) with the parameters from Section 7.2, achieves rapid initial learning on the CT, as well as improved long-term stability over the Fast parameters of Chapter 5. As mentioned in Section 6.5, long-term stability for practical applications requires observing performance out to 10,000 episodes. Stability after this point is interesting only as a case study of the continuous actor-critic. Figure 7.3 depicts performance on the CT. Notice that the system maintains improved performance out to 10,000 episodes, after which the system continues to improve on average, though some trials became unstable.

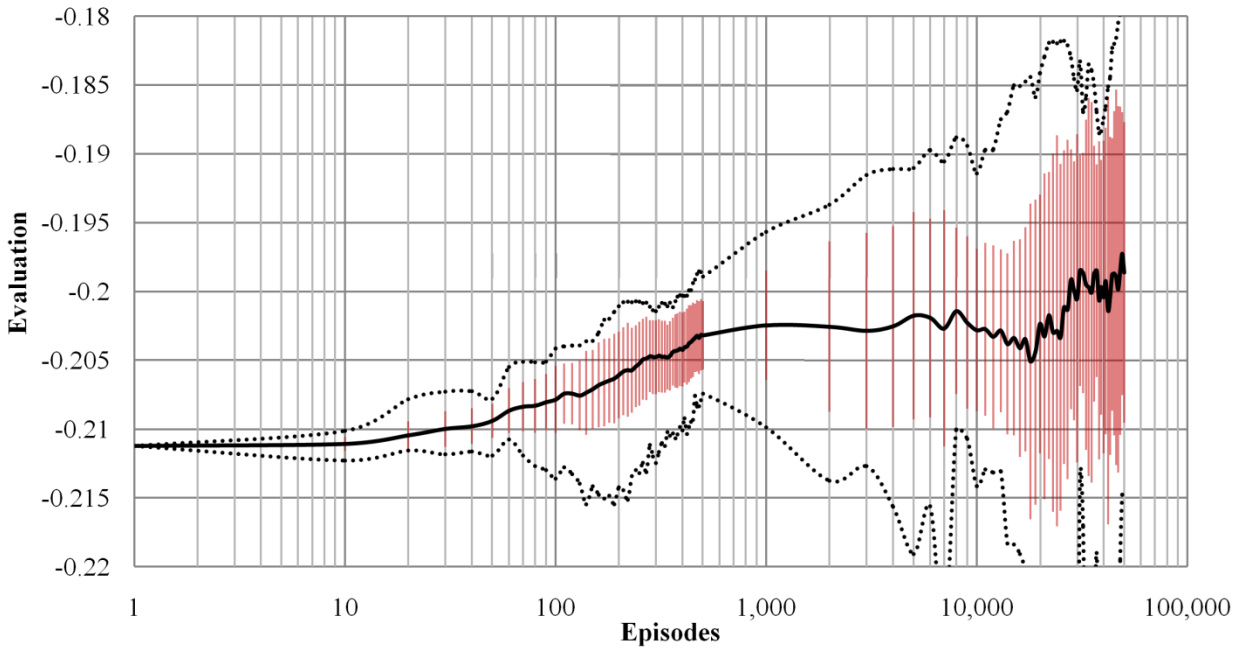


Figure 7.3: Mean performance (N=16) of the actor-critic with the ILWR-critic on the CT. Standard deviation error bars are provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis.

7.4 Baseline Biceps Test (BBT)

The actor-critic, when using the ANN-actor and ILWR-critic, excels on the BBT, achieving both rapid initial learning and significantly improved stability relative to the Fast parameters of Chapter 5. Performance is depicted in Figure 7.4. Although the system is not entirely stable out to 50,000 episodes, it is a vast improvement over the Fast parameters (Figure 5.5). Further observation of the 16 trials to create Figure 7.4 revealed that the runs were split into two classes: in one, the system remained completely stable with an evaluation around $-.2$; in the other, the system diverged to a final evaluation no worse than $-.35$. The majority of runs ($N \approx 11$) fell into the former category, with the remainder in the latter.

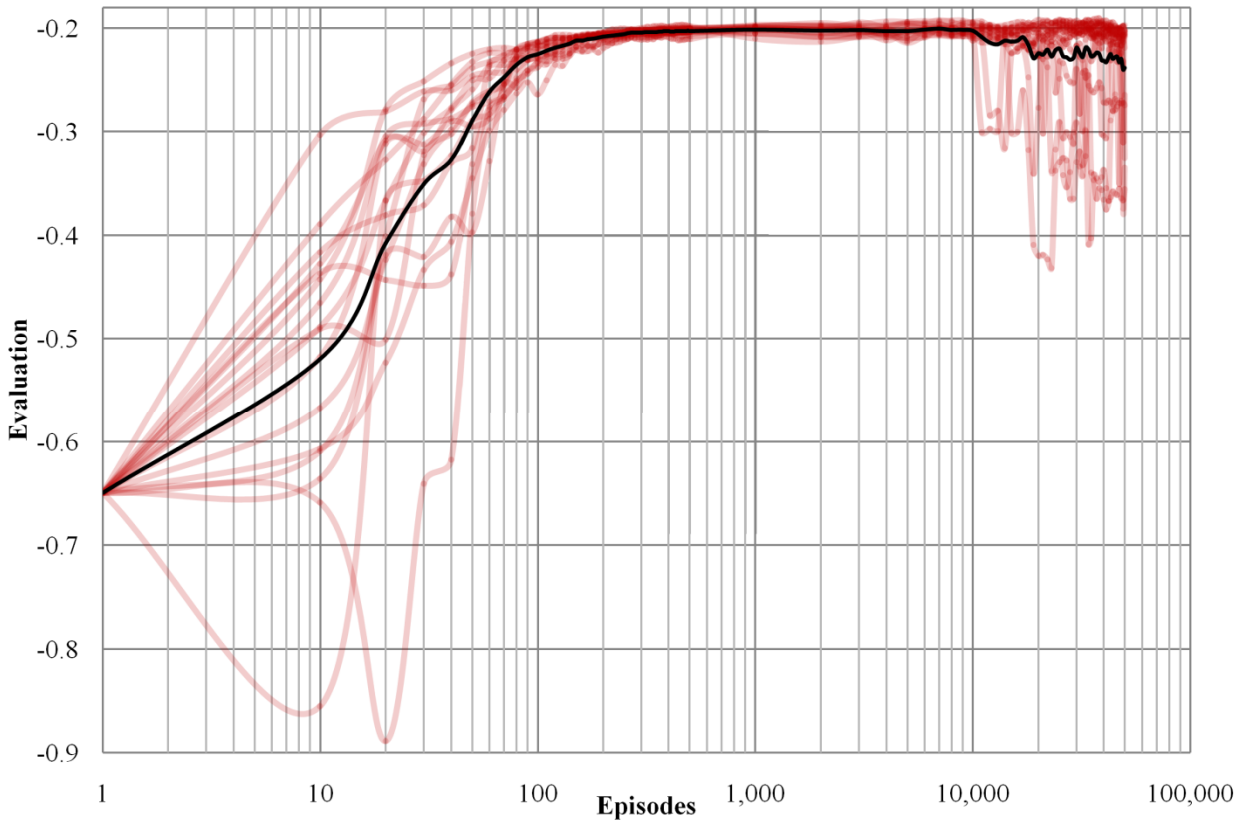


Figure 7.4: Mean performance (black solid line; N=16) of the actor-critic with the ILWR-critic on the BBT. In order to emphasize that most trials remained stable, all 16 trials are displayed as transparent red lines. Notice the dark red resulting from the majority of the trials maintaining an evaluation around –.2 after 50,000 training episodes. Standard deviation error bars are **not** provided. Notice the logarithmic scale of the horizontal axis.

7.5 Fatigued Triceps Test (FTT)

The actor-critic, when using the ANN-actor and ILWR-critic, excels on the FTT as well, achieving both rapid initial learning and significantly improved stability relative to that of the Fast parameters of Chapter 5. Performance is depicted in Figure 7.5. All 16 trials to create Figure 7.5 terminated with evaluations above $-.22$.

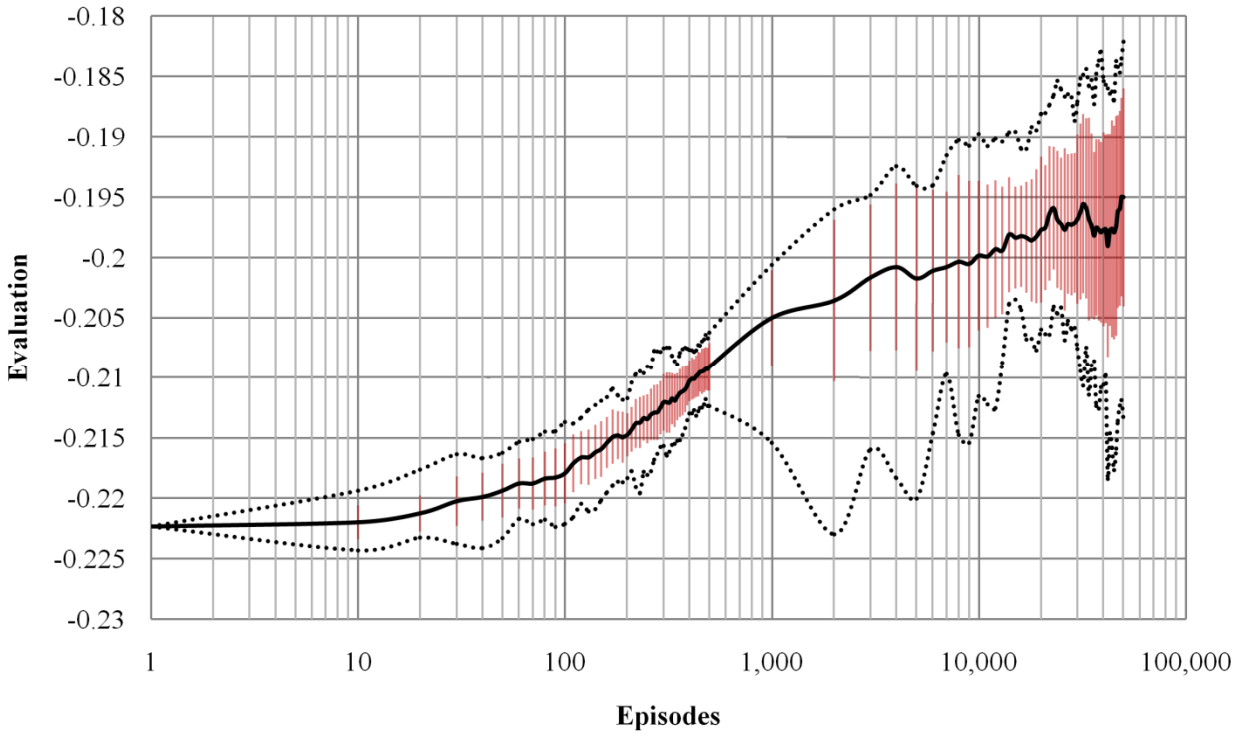


Figure 7.5: Mean performance (N=16) of the actor-critic with the ILWR-critic on the FTT. Standard deviation error bars are provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis.

7.6 Noise Robustness Test (NRT)

Performance of the actor-critic, when using the ANN-actor and ILWR-critic, is mediocre on the NRT with bias ($\mu_B = .05$). Though rapid initial learning is preserved, the maximum evaluations achieved are significantly diminished compared to the BBT without the NRT. It is possible that the noise added to sensor readings makes it impossible to perform better, though this is not known. The learning curve, provided in Figure 7.6, is similar to that of the BBT, with rapid initial learning and a significant improvement in long-term stability over the ANN-only actor-critic of Chapter 5. This system is also more stable than the Hybrid Controller on the NRT (cf. Figure 6.10). All 16 trials to create Figure 7.6 terminated with evaluations above $-.33$.

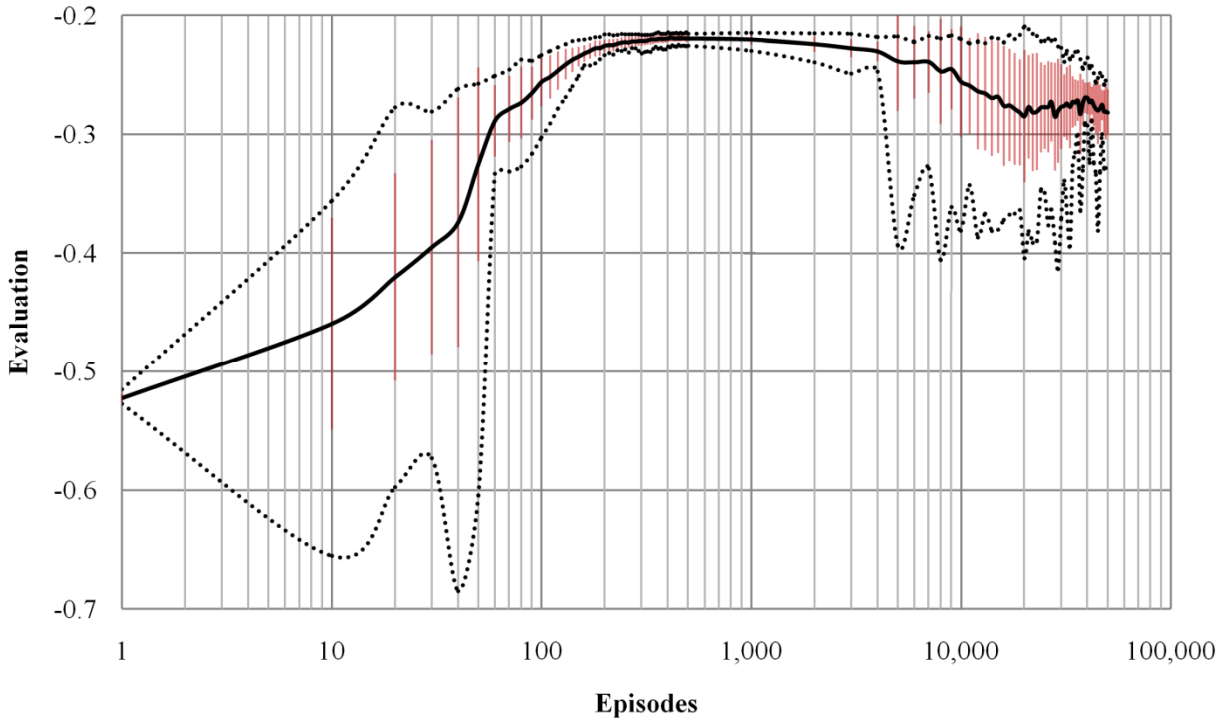


Figure 7.6: Mean performance (N=16) of the actor-critic with the ILWR-critic on the NRT. Standard deviation error bars are provided. The dotted lines represent the minimum and maximum values over all 16 trials. Notice the logarithmic scale of the horizontal axis.

7.7 Conclusion

Replacing the ANN-critic in Chapter 5 with an ILWR-critic drastically improved performance. The resulting system achieved rapid initial learning on the CT, BBT, FTT, and NRT, as well as improved long-term stability, without the need for toggling parameter sets as in the Hybrid Controller (Section 6.5). We suspect the improvement in performance is primarily due to the locality of ILWR updates. However, the system still remains unstable in the extremely long-term, beyond the timeframe considered for practical applications to FES control. This instability may be inherent to the continuous actor-critic itself, as discussed in Subsection 2.2.10.

In this chapter, the input learning rate of ILWR (see Chapter 3) was zero due to time constraints. As higher dimension control tasks are tackled, the ability to switch from SI-ILWR to DI-ILWR may be an effective means of combating the curse of dimensionality. Future research should be done to test DI-ILWR as the critic in control tasks of higher dimension. Work should also be done to determine the influence of using ILWR for the actor as well as the critic. Finally, work should be done to compare the results of using ILWR to those of kernel based methods such as RBFs. RBFs were not included in this work because preliminary tests failed to achieve acceptable performance as the critic during pre-training. This is likely due to an insufficient granularity of the search for optimal parameters.

CHAPTER 8:

CONCLUSION

This chapter is divided into two parts. Section 8.1 reviews the results and contributions of the previous chapters. Section 8.2 discusses possible future work.

8.1 Results and Contribution

This thesis, as a whole, serves as documentation of the application of the continuous actor-critic to the real-world problem of FES control of a human arm, discussing difficulties and the methods used to overcome them. The primary difficulties in FES control are that the dynamics of each subject's arm differ and the dynamics can change during trials due to muscle fatigue. The adaptive abilities of the controllers created herein were tested by requiring the controllers to adapt to changes in the arm model, which were inspired by variations in arm dynamics that were observed in actual human subjects.

We introduced the Adaptive RL FES Controller Task in Section 1.2, which requires a controller for DAS1, the arm simulator, be created that achieves rapid initial learning and long-term stability, while remaining robust to noise in sensor readings. In Section 2.1, we showed that two basic closed-loop controllers, PDs and PIDs, are insufficient for this task. We then proposed using RL methods to create an adaptive controller.

After reviewing RL in the beginning of Section 2.2, we analyzed the continuous actor-critic in Subsection 2.2.10, and showed its relation to the SRV algorithm (Subsection 2.2.9). We also discussed the lack of convergence guarantees, and provided intuition about how local the updates to the actor and critic should be. After reviewing function approximators in Section 2.3, we reproduced Doya's implementation (Doya, 2000) of the continuous actor-critic on the pendulum swing-up task in Section 2.4. We observed that the system learns even when the critic is not yet accurate.

In Chapter 3, we introduced ILWR and compared it to ANNs on several test problems, ranging from a simple function with one input and output (Sigmoid Environment, Subsection 3.1.1) to the non-linear FitzHugh-Nagumo environment (Subsections 3.1.3 and 3.1.4). We also compared ILWR and ANN's abilities to track a non-stationary function, which emulates the task of representing the critic in the continuous actor-critic. In all of the tests in Chapter 3, ILWR outperformed ANNs.

In Chapter 4, we introduced a slew of different tests to evaluate a controller's ability to adapt to clinically relevant changes in arm dynamics. These tests were dubbed the Control Test (CT, Section 4.2), Baseline Biceps Test (BBT, Section 4.3), Fatigued Triceps Test (FTT, Section 4.4), Noise Robustness Test (NRT, Section 4.5), Fatigued Biceps Test (FBT, Section 4.6), Toggling Test (TT, Section 4.7), Delayed Reward Test (DRT, Section 4.8), Discrete Reward Test (DiRT, Section 4.9), and Continuous Learning Modification (CLM, Section 4.10). The CT serves as a control, with the DAS1 arm model remaining unchanged. The BBT and FTT introduce changes to the arm dynamics, which mimic those expected in some FES subjects. The NRT tests the controller's ability to learn in the presence of sensor noise. The FBT is used in the TT as a specific test for the Hybrid Controller of Section 6.5. The DRT, DiRT, and CLM provide

additional insight into the controller's performance if humans were to provide the reward signal during actual trials.

In Chapter 5 we found that the continuous actor-critic, when using ANNs for both the actor and the critic on the tests from Chapter 4, could achieve either rapid initial learning or long-term stability, but not both. This was observed on the CT (Section 5.3), BBT (Section 5.4), and FTT (Section 5.5). Rapid initial learning was also observed to be robust to various amounts of exploration (Section 5.6), sensor noise (NRT, Section 5.7), a discretization of the reward signal (DRT, Section 5.8), and a delay in the reward signal (DiRT, Section 5.9).

We then attempted to improve long-term stability in the system devised in Chapter 5 by tweaking the cap on the TD-error (Section 6.1), by altering the muscle activation weight in the reward signal (Section 6.2), by only allowing updates to the actor when the critic is accurate (Section 6.3), and by adding a weight decay term to the ANN updates (Section 6.4). None of these improved long-term stability while preserving rapid initial learning, though the weight decay term did result in policies that generalized better to variations in arm dynamics, which was observed as improved initial performance on the BBT and FTT. Chapter 6 concludes by combining unstable rapid initial learning and slow but stable learning to create the Hybrid Controller of Section 6.5.

In Chapter 7, we attempt to achieve rapid initial learning and long-term stability without the need to toggle between various parameter settings as in the Hybrid Controller. After finding the proper parameters for training, ILWR is used to pre-train a critic. This critic is then used to replace the ANN-critic of Chapter 5. The remainder of Chapter 7 presents results on the CT (Section 7.3), BBT (Section 7.4), FTT (Section 7.5), and NRT (Section 7.6). In all cases, rapid

initial learning is preserved, while long-term stability is improved relative to the ANN critic of Chapter 5.

Though both the continuous actor-critic with an ANN actor and ILWR critic and the less elegant Hybrid Controller have achieved all the requirements of the Adaptive RL FES Controller Task, neither is completely stable. The instability, which presents beyond 10,000 arm movements, remains unexplained. It may be due to the gradient descent steps of the actor-critic being too large, or it may arise from the use of function approximators.

We have successfully completed the Adaptive RL FES Controller Task (Section 1.2) in two ways. The Hybrid Controller uses ANNs for both the actor and critic, though it requires changes to the parameters of the actor-critic, which may be either automated or manual. The second solution uses ILWR for the critic and an ANN for the actor. It achieves both rapid initial learning as well as long-term stability without the need for dynamic parameters.

Other than contributing to the FES literature by presenting an argument for the feasibility of RL for use in FES control, as discussed in Section 1.3, this thesis also contributes novel methods and theory to the RL literature. In Chapter 3, we introduced a novel function approximator, Incremental Locally Weighted Regression (ILWR), which outperforms ANNs in all tests executed in Chapters 3 and 7. In Section 6.4 we introduced a weight decay term to the continuous actor-critic, which resulted in policies that performed better when faced with minor variations to the environment. In Section 2.4, Chapter 5, and Chapter 6, we observed two different types of learning by the actor-critic: unstable rapid initial learning, and slow but stable learning.

8.2 Future Work

Because this work is among the first of its kind, applying RL to FES control, there is still significant room for further research. For example, research should be done to compare the performance of policy gradient methods to that of the continuous actor-critic, with a focus on stability and ability to scale to problems of higher dimension.

In Section 5.11, the continuous actor-critic with the pre-trained ANN actor and critic from Section 5.1, using the Fast parameters, learned on the BBT when the TD-error was replaced with a random negative signal. This remains unexplained. Further research should be performed to determine the reason for this learning, and whether it is common when using RL to adapt to a changing environment.

In Section 6.4, we introduced a weight decay term to the update equations for the continuous actor-critic (Subsection 2.2.8). For the task of FES control using the DAS1 model, this resulted in the actor-critic learning policies that performed better when the environment changed. In machine learning (e.g., classification), weight decay terms are known to improve the generalizability of results. Similarly, in RL, the weight decay term has increased the generalizability of a policy to similar environments. Further research should be done to determine whether this is a fluke of our particular system, or a trend throughout RL.

We observed, in Section 2.4, Chapter 5, and Chapter 6, that the actor-critic had two different types of learning. In the first type, the critic is accurate, and learning is slow but stable in the long-term, and Gullapalli's intuition (Subsection 2.2.10) applies. In the second, the critic is not yet accurate, so Gullapalli's intuition does not apply, and yet rapid initial learning occurs.

Future work should be done to develop an intuition for why the actor-critic learns under these conditions.

In Subsection 2.2.10, we suggested increasing the locality of updates to the critic as learning progresses in the continuous actor-critic architecture. Though our analysis suggested that this may improve performance, its application fell outside the scope of this work because the Adaptive RL FES Controller Task requires that learning parameters not be decayed.

In Chapter 3, we presented Incremental Locally Weighted Regression (ILWR), and compared it to other function approximators. In all of our tests, it outperformed Artificial Neural Networks (ANNs), especially when tracking a non-stationary function. In Chapter 7, we used it as the critic in the continuous actor-critic on a real-world problem, resulting in a significant improvement in stability over ANNs. Implementing ILWR for the actor as well as the critic could also lead to an additional improvement. Further research should be done into the performance of ILWR, with additional comparisons to ANNs and RBFs. An analysis should be performed of the differences between ILWR and RBFs with moving kernel centers. Most importantly, ILWR should be considered by researchers for use as an incremental function approximator. Additionally, an approximation or randomized algorithm for DI-ILWR updates (Chapter 3 and Appendix C) could improve runtimes for ILWR.

Throughout this thesis, the continuous actor-critic was found to be sensitive to its parameter settings, primarily the learning rates, the eligibility decay rate, exploration magnitude and time scale, reward decay rate, and the function approximators selected to represent the actor and the critic. A comparison of the sensitivity of different RL methods with respect to their

parameter settings would have been useful, providing additional information for consideration when initially selecting a learning algorithm.

The policies learned by the continuous actor-critic in Chapters 5 through 7 have unnatural muscle stimulations, as depicted in Figure 8.1. Because muscles and inertia act as a low-pass filter, movement is not sensitive to high frequencies in muscle stimulation. This, combined with the negative reward for muscle forces, ought to result in smooth muscle activations such as the PD's. Future work should be done including constraints on the derivatives of muscle stimulations requested by the controller. This may help to remove the high-frequency fluctuations that are not present in natural movements nor the PD controller's policy.

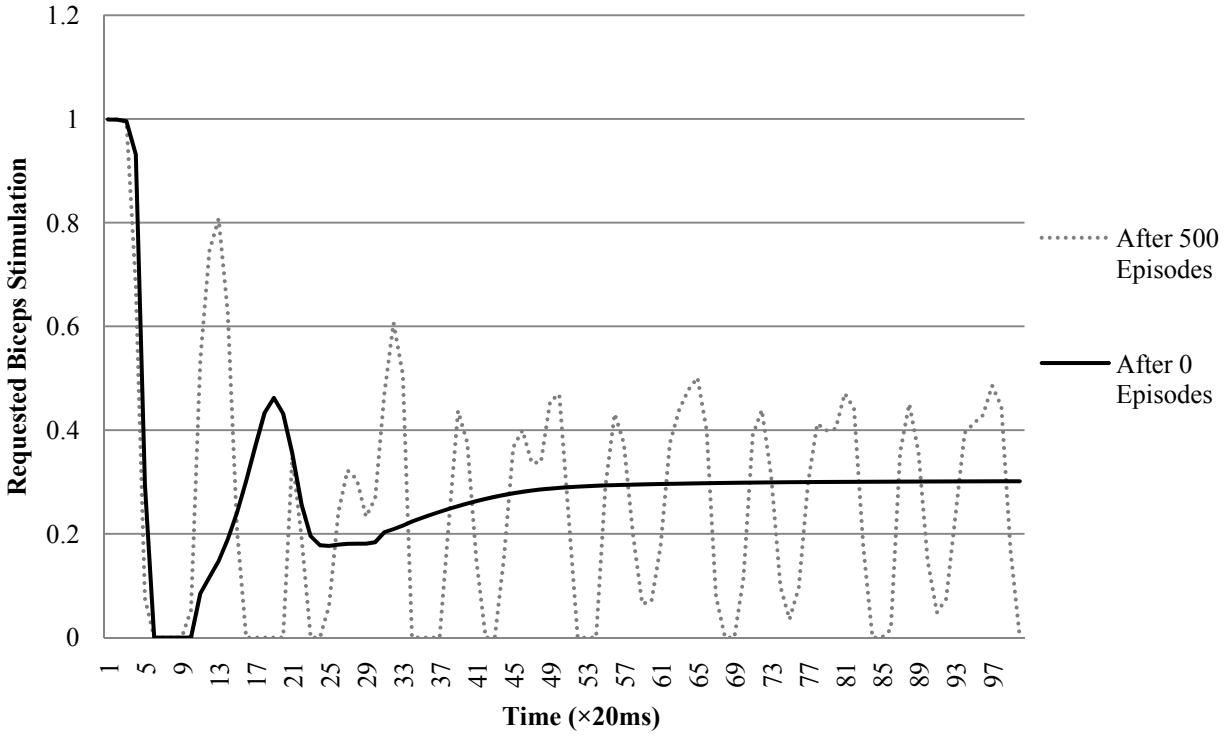


Figure 8.1: Requested biceps stimulation over an episode on the BBT before training and after 500 training episodes with the Fast parameters (Table 5.2).

Lastly, future research should be done into the application of RL controllers to FES, both in simulation and on human subjects. As this is one of the first attempts to apply RL techniques to FES, the research area is still open for significant development. The encouraging results from this thesis have inspired further work in the application of RL to FES control. At the Lerner Research Institute (LRI) of the Cleveland Clinic Foundation, researchers Kathleen Jagodnik and Dr. Antonie van den Bogert are preparing for human trials of this controller for planar arm movement, in which able-bodied subjects provide the reward signal. They will investigate how the learned policies will differ when the reward signal is provided by a human rather than generated automatically via Equation 4.1. If these tests are successful, the controller may be used for human trials using FES on a patient with spinal cord injury.

Researchers at the LRI have also created a detailed three-dimensional musculoskeletal model of a human arm (Chadwick et al., 2009). Pending successful results from the real-world application of RL for planar control, the RL controllers from Section 6.5 and Chapter 7 could be applied to the three-dimensional model, and eventually three-dimensional human trials. The primary difficulty in the switch will be the increase in the dimension of the action space, as the three-dimensional model includes over 100 muscles, though this can be overcome by clustering similar muscles into groups that are all given equal stimulation. Additionally, the ability of ILWR to cluster knowledge points around interesting areas of the domain, combined with its planar local model, may help combat the increases in state and action space dimensions.

This thesis has shown that RL is a viable approach for adaptive control tasks, specifically FES control of a human arm, and will hopefully open up a vein of further research in the area, with the long-term goal of restoring natural motor function to people with spinal cord injury.

APPENDIX A

This appendix contains a derivation of Equation 3.5, which states

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial y_{i,j}} = \begin{cases} 0, & \text{when } j \neq k, \\ \left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\right]_{d_i+1,i}, & \text{otherwise.} \end{cases} \tag{3.5}$$

Recall from (Schaal, Atkeson, and Vijayakumar, 2002) that

$$\boldsymbol{\beta} = \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\mathbf{y}. \tag{A1}$$

Therefore,

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial y_{i,j}} = \frac{\partial}{\partial y_{i,j}} \sum_{\alpha=1}^{p} \left(\left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\right]_{d_i+1,\alpha} \mathbf{y}_{\alpha,k} \right), \tag{A2}$$

by the definition of matrix multiplication. The summation is from one to p because $\left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\right]$ has p columns and $\mathbf{y}$ has p rows. When $j \neq k$, the right hand side is not a function of $y_{i,j}$, so

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial y_{i,j}} = 0 \text{ when } j \neq k, \tag{A3}$$

and when $j = k$,

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,j}}{\partial y_{i,j}} = \sum_{\alpha=1}^{p} \frac{\partial}{\partial y_{i,j}} \left(\left[\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}\right]_{d_i+1,\alpha} \mathbf{y}_{\alpha,j} \right) \tag{A4}$$

$$= \sum_{\alpha=1}^{p} \left(\left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \right]_{d_i+1,\alpha} \frac{\partial \mathbf{y}_{\alpha,j}}{\partial y_{i,j}} \right) \tag{A5}$$

$$= \left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \right]_{d_i+1,i}. \tag{A6}$$

Together, Equations A3 and A6 imply Equation 3.5.

APPENDIX B

This appendix contains derivations of Equations 3.6 and 3.7, which state

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial x_{i,j}} = \left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \left[\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y} + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{y} \right] - \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \left[\mathbf{X}^T \left(\mathbf{W} \frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{X} \right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{X} \right] \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \right]_{d_i+1,k} \tag{3.6}$$

and

$$\left[\frac{\partial \mathbf{W}}{\partial x_{i,j}} \right]_{i,i} = \mathbf{W}_{i,i} \mathbf{D}_{j,j} \left(x_{q,j} - x_{i,j} \right). \tag{3.7}$$

Recall from (Schaal, Atkeson, and Vijayakumar, 2002) that

$$\boldsymbol{\beta} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{B1}$$

Also recall the following rules from matrix calculus (Edwards and Penney, 2002):

Inverse Rule: $$\partial \mathbf{A}^{-1} = \mathbf{A}^{-1} \left(\partial \mathbf{A}\right) \mathbf{A}^{-1} \tag{B2}$$

Transpose Rule: $$\partial \left(\mathbf{A}^T \right) = \left(\partial \mathbf{A}\right)^T \tag{B3}$$

Summation Rule: $$\partial \left(\mathbf{A} + \mathbf{B} \right) = \partial \mathbf{A} + \partial \mathbf{B} \tag{B4}$$

Product Rule: $$\partial \left(\mathbf{A}\mathbf{B} \right) = \mathbf{A} \left(\partial \mathbf{B}\right) + \left(\partial \mathbf{A}\right) \mathbf{B} \tag{B5}$$

Also recall that matrix multiplication is associative, but not commutative. This appendix uses the following notations: $x_{i,j}$ is the j^{th} input of x_i, the i^{th} knowledge point; there are p knowledge

points; the inputs are of dimension d_i; and the outputs are of dimension d_o. Any other notation not specified is consistent with that of Chapter 3.

First, we expand $\boldsymbol{\beta}_{d_i+1,k}$ to obtain

$$\frac{\partial \boldsymbol{\beta}_{d_i+1,k}}{\partial x_{i,j}} = \frac{\partial \left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \right]_{d_i+1,k}}{\partial x_{i,j}}. \tag{B6}$$

By solving for

$$\frac{\partial \left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \right]}{\partial x_{i,j}}, \tag{B7}$$

we can easily extract the element at d_i+1,k for each k. When performing gradient descent on the error term, the results for $1 \le k \le d_o$ will be computed. We can expand Equation B7 using the product and inverse rules to obtain

$$\frac{\partial \left[\left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \right]}{\partial x_{i,j}} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \frac{\partial \mathbf{X}^T \mathbf{W} \mathbf{y}}{\partial x_{i,j}} + \frac{\partial \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1}}{\partial x_{i,j}} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{B8}$$

$$= \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \left[\mathbf{X}^T \frac{\partial \mathbf{W} \mathbf{y}}{\partial x_{i,j}} + \frac{\partial \mathbf{X}^T}{\partial x_{i,j}} \mathbf{W} \mathbf{y} \right] + \frac{\partial \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1}}{\partial x_{i,j}} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{B9}$$

$$= \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \left[\mathbf{X}^T \frac{\partial \mathbf{W} \mathbf{y}}{\partial x_{i,j}} + \frac{\partial \mathbf{X}^T}{\partial x_{i,j}} \mathbf{W} \mathbf{y} \right] - \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \frac{\partial \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)}{\partial x_{i,j}} \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{B10}$$

For simplicity, we will let $\boldsymbol{\theta} = \left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)^{-1}$. This value was computed during the approximation stage in the LWR algorithm, and can be stored so it need not be recomputed during the weight update stage. Substituting and applying the transpose rule, Equation B10 may be written as

$$\boldsymbol{\theta}\left[\mathbf{X}^T\frac{\partial\mathbf{W}\mathbf{y}}{\partial x_{i,j}} + \left(\frac{\partial\mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{y}\right] - \boldsymbol{\theta}\frac{\partial\left(\mathbf{X}^T\mathbf{W}\mathbf{X}\right)}{\partial x_{i,j}}\boldsymbol{\theta}\mathbf{X}^T\mathbf{W}\mathbf{y}. \tag{B11}$$

Applying the product rule twice results in

$$\boldsymbol{\theta}\left[\mathbf{X}^T\left(\mathbf{W}\frac{\partial\mathbf{y}}{\partial x_{i,j}} + \frac{\partial\mathbf{W}}{\partial x_{i,j}}\mathbf{y}\right) + \left(\frac{\partial\mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{y}\right] - \boldsymbol{\theta}\left[\mathbf{X}^T\frac{\partial(\mathbf{W}\mathbf{X})}{\partial x_{i,j}} + \frac{\partial\left(\mathbf{X}^T\right)}{\partial x_{i,j}}\mathbf{W}\mathbf{X}\right]\boldsymbol{\theta}\mathbf{X}^T\mathbf{W}\mathbf{y}. \tag{B12}$$

Applying the transpose rule, product rule, and removing the term

$$\frac{\partial\mathbf{y}}{\partial x_{i,j}} = 0, \tag{B13}$$

we obtain the simplification,

$$\boldsymbol{\theta}\left[\mathbf{X}^T\frac{\partial\mathbf{W}}{\partial x_{i,j}}\mathbf{y} + \left(\frac{\partial\mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{y}\right] - \boldsymbol{\theta}\left[\mathbf{X}^T\left(\mathbf{W}\frac{\partial\mathbf{X}}{\partial x_{i,j}} + \frac{\partial\mathbf{W}}{\partial x_{i,j}}\mathbf{X}\right) + \left(\frac{\partial\mathbf{X}}{\partial x_{i,j}}\right)^T\mathbf{W}\mathbf{X}\right]\boldsymbol{\theta}\mathbf{X}^T\mathbf{W}\mathbf{y}. \tag{B14}$$

Notice that $\partial\mathbf{X}/\partial x_{i,j}$ is a matrix with all zeros except $\mathbf{X}_{i,j} = 1$. If using an unweighted version of LWR where $\mathbf{W}$ is not a function of $x_{i,j}$, then $\partial\mathbf{W}/\partial x_{i,j} = \mathbf{0}$. If using LWR with weights, as is normal, $\partial\mathbf{W}/\partial x_{i,j} = \mathbf{0}$, except for

$$\left[\frac{\partial\mathbf{W}}{\partial x_{i,j}}\right]_{i,i} = \frac{\partial}{\partial x_{i,j}}\left(e^{-\frac{1}{2}\left(\mathbf{x}_i-\mathbf{x}_q\right)^T\mathbf{D}\left(\mathbf{x}_i-\mathbf{x}_q\right)}\right). \tag{B15}$$

Converting out of vector notation, this becomes

$$= \frac{\partial}{\partial x_{i,j}} \left(e^{-\frac{1}{2}\sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]} \right). \tag{B16}$$

Moving the derivative into the exponent, we obtain

$$= e^{-\frac{1}{2}\sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]} \frac{\partial\left(-\frac{1}{2}\sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]\right)}{\partial x_{i,j}}. \tag{B17}$$

Substituting in

$$\mathbf{W}_{i,i} = e^{-\frac{1}{2}\sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]}, \tag{B18}$$

we obtain

$$\left[\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right]_{i,i} = \mathbf{W}_{i,i} \frac{\partial\left(-\frac{1}{2}\sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]\right)}{\partial x_{i,j}}. \tag{B20}$$

Applying calculus, we obtain

$$= -\frac{\mathbf{W}_{i,i}}{2} \frac{\partial \sum_{\alpha=1}^{d_i}\left[\left(x_{i,\alpha}-x_{q,\alpha}\right)^2 \mathbf{D}_{\alpha,\alpha}\right]}{\partial x_{i,j}} \tag{B21}$$

$$= -\frac{\mathbf{W}_{i,i}}{2} \frac{\partial\left(x_{i,j}-x_{q,j}\right)^2 \mathbf{D}_{j,j}}{\partial x_{i,j}} \tag{B22}$$

$$= -\frac{\mathbf{W}_{i,i}\mathbf{D}_{j,j}}{2}\frac{\partial\left(x_{i,j}^2 - 2x_{q,j}x_{i,j} + x_{q,j}^2\right)}{\partial x_{i,j}} \tag{B23}$$

$$= -\frac{\mathbf{W}_{i,i}\mathbf{D}_{j,j}}{2}\left(2x_{i,j} - 2x_{q,j}\right) \tag{B24}$$

$$= \mathbf{W}_{i,i}\mathbf{D}_{j,j}\left(x_{q,j} - x_{i,j}\right), \tag{B25}$$

which is Equation 3.7. Note that this assumes **D** is independent of $x_{i,j}$.

APPENDIX C

This appendix contains directions for efficiently computing $\partial \hat{y}_{q,k} / \partial x_{i,j}$. Recall Equation 3.6, which may be rewritten as follows:

$$\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}} = \left\{ \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \left[\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y} + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{y} \right] - \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \left[\mathbf{X}^T \left(\mathbf{W} \frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{X} \right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{X} \right] \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W} \mathbf{y} \right\}. \tag{C1}$$

First, recall that $\boldsymbol{\theta} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1}$ was already computed during the approximation step in the incremental LWR algorithms. Substituting in $\boldsymbol{\theta}$, we obtain

$$\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}} = \left\{ \boldsymbol{\theta} \left[\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y} + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{y} \right] - \boldsymbol{\theta} \left[\mathbf{X}^T \left(\mathbf{W} \frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{X} \right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{X} \right] \boldsymbol{\theta} \mathbf{X}^T \mathbf{W} \mathbf{y} \right\}. \tag{C2}$$

Also notice that the entire matrix need only be computed once for each *i,j*, not for each distinct *k*. $\partial \mathbf{W} / \partial x_{i,j}$ and $\partial \mathbf{X} / \partial x_{i,j}$ can be computed efficiently, as both have at most one non-zero entry.

Next, we write $\partial \boldsymbol{\beta} / \partial x_{i,j}$ in terms of $\mathbf{r}_1$ and $\mathbf{r}_2$, which are defined as

$$\mathbf{r}_1 = \boldsymbol{\theta} \left[\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y} + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{y} \right] \tag{C3}$$

and

$$\mathbf{r}_2 = \boldsymbol{\theta} \left[\mathbf{X}^T \left(\mathbf{W} \frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{X} \right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}} \right)^T \mathbf{W} \mathbf{X} \right] \boldsymbol{\theta} \mathbf{X}^T \mathbf{W} \mathbf{y}. \tag{C4}$$

After computing $\mathbf{r}_1$ and $\mathbf{r}_2$, $\partial\boldsymbol{\beta} / \partial x_{i,j}$ can be computed as

$$\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}} = \mathbf{r}_1 - \mathbf{r}_2. \tag{C5}$$

Let $\mathbb{R}_{i,j}$ denote a real-valued number in the i^{th} row and j^{th} column, while $1_{i,j}$ denotes a 1 in the i^{th} row and j^{th} column. We begin by analyzing the computation of $\mathbf{r}_1$ by following the matrix operations to compute it, simplifying using the known forms of $\partial\mathbf{W} / \partial x_{i,j}$ (See Equation 3.6), $\partial\mathbf{X} / \partial x_{i,j}$, and $\mathbf{W}$:

$$\frac{\partial \mathbf{W}}{\partial x_{i,j}} = \underset{(p \times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & \mathbb{R}_{i,i} & \vdots \\ & & \\ 0 & \cdots & 0 \end{bmatrix}}, \tag{C6}$$

$$\frac{\partial \mathbf{X}}{\partial x_{i,j}} = \underset{\left(p \times d_i + 1\right)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & & \vdots \\ & 1_{i,j} & \\ 0 & \cdots & 0 \end{bmatrix}}, \tag{C7}$$

$$\mathbf{W} = \underset{(p \times p)}{\begin{bmatrix} \mathbf{W}_{1,1} & 0 & \cdots & & 0 \\ 0 & \mathbf{W}_{2,2} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \mathbf{W}_{p-1,p-1} & 0 \\ 0 & & \cdots & 0 & \mathbf{W}_{p,p} \end{bmatrix}}. \tag{C8}$$

The first step in computing $\mathbf{r}_1$ is computing $\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}}$:

$$\underset{(d_i+1\times p)}{\begin{bmatrix} \mathbb{R}_{1,1} & \cdots & \mathbb{R}_{1,p} \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{d_i+1,1} & \cdots & \mathbb{R}_{d_i+1,p} \end{bmatrix}}^{\mathbf{X}^T} \underset{(p\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ & \mathbb{R}_{i,i} & \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{bmatrix}}^{\frac{\partial \mathbf{W}}{\partial x_{i,j}}} = \underset{(d_i+1\times p)}{\begin{bmatrix} & \mathbb{R}_{1,i} & \\ & \mathbb{R}_{2,i} & \\ \mathbf{0} & \vdots & \mathbf{0} \\ & \mathbb{R}_{d_i,i} & \\ & \mathbb{R}_{d_i+1,i} & \end{bmatrix}}^{\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}}}, \quad \text{(C9)}$$

where $\mathbf{0}$ represents a submatrix with all entries equal to zero. The result can be expressed as

$$\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} = \left[\mathbf{0}, \mathbf{0}, ..., \mathbf{v}^1, ..., \mathbf{0}, \mathbf{0}\right], \quad \text{(C10)}$$

where $\mathbf{0}, \mathbf{v}^1 \in \mathbb{R}^{d_i+1}$. $\mathbf{v}^1$ can be written as

$$\mathbf{v}^1_\alpha = \mathbf{X}^T_{\alpha,i} \left(\frac{\partial \mathbf{W}}{\partial x_{i,j}} \right)_{i,i} \text{ for } 1 \le \alpha \le d_i + 1. \quad \text{(C11)}$$

The next step in computing $\mathbf{r}_1$ is computing $\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y}$:

$$\underset{}{\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}}} \qquad \underset{}{\mathbf{y}} \qquad\qquad \underset{}{\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y}}$$

$$\underset{(d_i+1\times p)}{\begin{bmatrix} \mathbf{0} & \cdots & \mathbf{v}^1 & \cdots & \mathbf{0} \end{bmatrix}} \underset{(p\times d_o)}{\begin{bmatrix} \mathbb{R}_{1,1} & & \cdots & & \mathbb{R}_{1,d_o} \\ \vdots & & \ddots & & \vdots \\ \mathbb{R}_{i,1} & \mathbb{R}_{i,2} & \cdots & \mathbb{R}_{i,d_o-1} & \mathbb{R}_{i,d_o} \\ \mathbb{R}_{p,1} & & \cdots & & \mathbb{R}_{p,d_o} \end{bmatrix}} = \underset{(d_i+1\times d_o)}{\begin{bmatrix} \mathbb{R}_{1,1} & \cdots & \mathbb{R}_{1,d_o} \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{p,1} & \cdots & \mathbb{R}_{p,d_o} \end{bmatrix}}. \quad \text{(C12)}$$

Note that the elements in $\mathbf{X}^T\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{y}$ are a function of only $\mathbf{v}^1$ and the i^{th} row of $\mathbf{y}$. As such, we can write an equation for each element in $\mathbf{X}^T\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{y}$,

$$\left[\mathbf{X}^T \frac{\partial \mathbf{W}}{\partial x_{i,j}} \mathbf{y}\right]_{t,u} = \mathbf{v}_t^1 \mathbf{y}_{i,u} = \mathbf{X}_{t,i}^T \left(\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i} \mathbf{y}_{i,u}. \quad \text{(C13)}$$

Next we must solve for the second half of $\mathbf{r}_1$, $\left(\partial \mathbf{X}/\partial x_{i,j}\right)^T \mathbf{W}\mathbf{y}$. The first step of this is to compute $\left(\partial \mathbf{X}/\partial x_{i,j}\right)^T \mathbf{W}$:

$$\underset{}{\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T} \qquad \underset{}{\mathbf{W}} \qquad\qquad \underset{}{\frac{\partial \mathbf{X}^T}{\partial x_{i,j}} \mathbf{W}}$$

$$\underset{(d_i+1\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & 1_{j,i} & \vdots \\ & & \\ 0 & \cdots & 0 \end{bmatrix}} \underset{(p\times p)}{\begin{bmatrix} \mathbf{W}_{1,1} & 0 & \cdots & & 0 \\ 0 & \mathbf{W}_{2,2} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \mathbf{W}_{p-1,p-1} & 0 \\ 0 & & \cdots & 0 & \mathbf{W}_{p,p} \end{bmatrix}} = \underset{(d_i+1\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ & \mathbb{R}_{j,i} & \\ 0 & \cdots & 0 \end{bmatrix}}. \quad \text{(C14)}$$

The real number in the j^{th} row and i^{th} column of $\frac{\partial \mathbf{X}^T}{\partial x_{i,j}} \mathbf{W}$ can be expressed as

$$\left[\frac{\partial \mathbf{X}^T}{\partial x_{i,j}}\mathbf{W}\right]_{j,i} = \mathbf{W}_{i,i}. \tag{C15}$$

The next step is to compute $\left(\partial \mathbf{X}/\partial x_{i,j}\right)^T \mathbf{W}\mathbf{y}$:

$$\underset{\left(d_i+1\times p\right)}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ & \mathbb{R}_{j,i} & \\ 0 & \cdots & 0 \end{bmatrix}} \underset{\left(p\times d_o\right)}{\begin{bmatrix} \mathbb{R}_{1,1} & \cdots & \mathbb{R}_{1,d_o} \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{p,1} & \cdots & \mathbb{R}_{p,d_o} \end{bmatrix}} = \underset{\left(d_i+1\times d_o\right)}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{j,1} & \mathbb{R}_{j,2} \cdots \mathbb{R}_{j,d_o-1} & \mathbb{R}_{j,d_o} \\ 0 & \cdots & 0 \end{bmatrix}} \tag{C16}$$

$\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}$ $\quad$ $\mathbf{y}$ $\quad$ $\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}\mathbf{y}$

The j^{th} row of $\left(\partial \mathbf{X}/\partial x_{i,j}\right)^T \mathbf{W}\mathbf{y}$ can be written as a row vector, $\mathbf{v}^2$, each element of which can be expressed as

$$\mathbf{v}^2_\alpha = \mathbf{W}_{i,i}\mathbf{y}_{i,\alpha} \text{ for } 1 \le \alpha \le d_o + 1. \tag{C17}$$

Combining this with Equation C13, we can compute $\mathbf{r}_1$, by first building $\hat{\mathbf{r}}_1$ as a $\left(d_i+1\times d_o\right)$ matrix with

$$\left[\hat{\mathbf{r}}_1\right]_{t,u} = \begin{cases} \mathbf{X}^T_{t,i}\left(\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i}\mathbf{y}_{i,u}, & t \ne j, \\ \mathbf{X}^T_{t,i}\left(\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i}\mathbf{y}_{i,u} + W_{i,i}y_{i,u}, & t = j, \end{cases} \tag{C18}$$

and then multiplying $\hat{\mathbf{r}}_1$ by $\boldsymbol{\theta}$ $\left(d_i+1\times d_i+1\right)$ on the left-hand side, giving the $d_i+1\times d_o$ result:

$$\mathbf{r}_1 = \boldsymbol{\theta}\hat{\mathbf{r}}_1. \tag{C19}$$

The time to compute $\mathbf{r}_1$ is $O\left(d_i^2 d_o + d_o d_i\right)$. The $d_i^2 d_o$ term comes from the cost of the final multiplication by $\boldsymbol{\theta}$.

Next, we compute $\mathbf{r}_2$, defined in Equation C4, using the same method. We first simplify the end by computing the $d_i + 1 \times d_o$ matrix $\boldsymbol{\varphi} = \mathbf{X}^T \mathbf{W}\mathbf{y}$. This term is independent of *i*, *j*, and *k* and need not be computed more than once for each query. Substituting in $\boldsymbol{\varphi}$, we obtain

$$\mathbf{r}_2 = \boldsymbol{\theta}\left[\mathbf{X}^T\left(\mathbf{W}\frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X}\right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}\mathbf{X}\right]\boldsymbol{\theta}\boldsymbol{\varphi}. \tag{C19}$$

Next we compute the $\mathbf{W}\left(\partial \mathbf{X}/\partial x_{i,j}\right)$ term:

$$\underset{(p\times p)}{\begin{bmatrix} \mathbf{W}_{1,1} & 0 & \cdots & & 0 \\ 0 & \mathbf{W}_{2,2} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \mathbf{W}_{p-1,p-1} & 0 \\ 0 & & \cdots & 0 & \mathbf{W}_{p,p} \end{bmatrix}} \underset{(p\times d_i+1)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & & \vdots \\ & 1_{i,j} & \\ 0 & \cdots & 0 \end{bmatrix}} = \underset{(p\times d_i+1)}{\begin{bmatrix} 0 & \cdots & 0 \\ & & \\ \vdots & \ddots & \vdots \\ & \mathbb{R}_{i,j} & \\ 0 & \cdots & 0 \end{bmatrix}}. \tag{C20}$$

$\mathbf{W}$ $\quad$ $\dfrac{\partial \mathbf{X}}{\partial x_{i,j}}$ $\quad$ $\mathbf{W}\dfrac{\partial \mathbf{X}}{\partial x_{i,j}}$

The $\mathbb{R}_{i,j}$ term is $\mathbf{W}_{i,i}$. Next we compute the $\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{X}$ term:

$$\underset{(p\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & \mathbb{R}_{i,i} & \vdots \\ & & \\ 0 & \cdots & 0 \end{bmatrix}} \underset{(p\times d_i+1)}{\begin{bmatrix} x_{1,1} & \cdots & x_{1,d_i} & 1_{1,d_i+1} \\ & & & \\ \vdots & \ddots & & \vdots \\ & & & \\ x_{p,1} & \cdots & x_{p,d_i} & 1_{p,d_i+1} \end{bmatrix}} = \underset{(p\times d_i+1)}{\begin{bmatrix} 0 & \cdots & & 0 \\ & & & \\ \vdots & & \ddots & \vdots \\ \mathbb{R}_{i,1} & \mathbb{R}_{i,2} & \cdots & \mathbb{R}_{i,d_i} & \mathbb{R}_{i,d_i+1} \\ 0 & \cdots & & 0 \end{bmatrix}}. \quad \text{(C21)}$$

$\frac{\partial \mathbf{W}}{\partial x_{i,j}}$ $\mathbf{X}$ $\frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X}$

The i^{th} row of $\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{X}$ can be expressed as a row vector, $\mathbf{v}^3 \in \mathbb{R}^{d_i+1}$, which can be described as

$$\mathbf{v}^3_\alpha = \left(\frac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i} x_{i,\alpha}. \quad \text{(C22)}$$

Using Equations C20 and C22, $\mathbf{W}\left(\partial \mathbf{X}/\partial x_{i,j}\right)+\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{X}$ can now be computed as

$$\mathbf{W}\frac{\partial \mathbf{X}}{\partial x_{i,j}}+\frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X} = \underset{(p\times d_i+1)}{\begin{bmatrix} 0 & \cdots & & 0 \\ & & & \\ \vdots & & \ddots & \vdots \\ \mathbb{R}_{i,1} & \mathbb{R}_{i,2} & \cdots & \mathbb{R}_{i,d_i} & \mathbb{R}_{i,d_i+1} \\ 0 & \cdots & & 0 \end{bmatrix}}, \quad \text{(C23)}$$

where the i^{th} row can be expressed as row vector $\mathbf{v}^3$ (Equation C22) plus $\mathbf{W}_{i,i}$ in the i^{th} row and j^{th} column. We will call this row vector $\mathbf{v}^4$, defined as

$$\mathbf{v}_{\alpha}^{4}=\begin{cases}\left(\dfrac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i} x_{i,\alpha}, & \text{when } \alpha \neq j, \\ \left(\dfrac{\partial \mathbf{W}}{\partial x_{i,j}}\right)_{i,i} x_{i,\alpha}+\mathbf{W}_{i,i}, & \text{otherwise.}\end{cases} \tag{C24}$$

We can now compute $\mathbf{X}^{T}\left[\mathbf{W}\left(\partial \mathbf{X}/\partial x_{i,j}\right)+\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{X}\right]$:

$$\begin{array}{ccc} \mathbf{X}^{T} & \mathbf{W}\dfrac{\partial \mathbf{X}}{\partial x_{i,j}}+\dfrac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X} & \mathbf{X}^{T}\left(\mathbf{W}\dfrac{\partial \mathbf{X}}{\partial x_{i,j}}+\dfrac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X}\right) \\ \begin{bmatrix} \mathbb{R}_{1,1} & \cdots & \mathbb{R}_{1,p} \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{d_i+1,1} & \cdots & \mathbb{R}_{d_i+1,p} \end{bmatrix} & \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{v}_i^4 \\ \vdots \\ \mathbf{0} \end{bmatrix} & = \begin{bmatrix} \mathbb{R}_{1,1} & \cdots & \mathbb{R}_{1,d_i+1} \\ \vdots & \ddots & \vdots \\ \mathbb{R}_{d_i+1,1} & \cdots & \mathbb{R}_{d_i+1,d_i+1} \end{bmatrix}. \\ \left(d_i+1\times p\right) & \left(p\times d_i+1\right) & \left(d_i+1\times d_i+1\right) \end{array} \tag{C25}$$

Each element in $\mathbf{X}^{T}\left[\mathbf{W}\left(\partial \mathbf{X}/\partial x_{i,j}\right)+\left(\partial \mathbf{W}/\partial x_{i,j}\right)\mathbf{X}\right]$ can be expressed as

$$\left[\mathbf{X}^{T}\left(\mathbf{W}\frac{\partial \mathbf{X}}{\partial x_{i,j}}+\frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X}\right)\right]_{t,u}=\mathbf{X}_{t,i}^{T}\mathbf{v}_{u}^{4}. \tag{C26}$$

Next, we must compute $\left(\partial \mathbf{X}/\partial x_{i,j}\right)^{T}\mathbf{W}$, as described in Equation C14, reproduced below for convenience:

$$\underset{}{\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T} \quad \underset{}{\mathbf{W}} \quad = \quad \underset{}{\frac{\partial \mathbf{X}^T}{\partial x_{i,j}}\mathbf{W}}$$

$$\underset{(d_i+1\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & & 1_{j,i} \quad \vdots \\ & & \\ 0 & \cdots & 0 \end{bmatrix}} \underset{(p\times p)}{\begin{bmatrix} \mathbf{W}_{1,1} & 0 & \cdots & & 0 \\ 0 & \mathbf{W}_{2,2} & & & \\ \vdots & & \ddots & & \vdots \\ & & & \mathbf{W}_{p-1,p-1} & 0 \\ 0 & & \cdots & 0 & \mathbf{W}_{p,p} \end{bmatrix}} = \underset{(d_i+1\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ & \mathbb{R}_{j,i} & \\ 0 & \cdots & 0 \end{bmatrix}}, \quad \text{(C14)}$$

where $\mathbb{R}_{j,i} = \mathbf{W}_{i,i}$. Next, we must multiply by **X**:

$$\underset{}{\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}} \quad \underset{}{\mathbf{X}} \quad = \quad \underset{}{\left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}\mathbf{X}}$$

$$\underset{(d_i+1\times p)}{\begin{bmatrix} 0 & \cdots & 0 \\ & \ddots & \\ \vdots & \mathbb{R}_{j,i} & \vdots \\ & & \\ 0 & \cdots & 0 \end{bmatrix}} \underset{(p\times d_i+1)}{\begin{bmatrix} x_{1,1} & \cdots & x_{1,d_i} & 1_{1,d_i+1} \\ \vdots & \ddots & & \vdots \\ x_{p,1} & \cdots & x_{p,d_i} & 1_{p,d_i+1} \end{bmatrix}} = \underset{(d_i+1\times d_i+1)}{\begin{bmatrix} 0 & & \cdots & & 0 \\ & \ddots & & & \\ \mathbb{R}_{j,1} & \mathbb{R}_{j,2} & \cdots & \mathbb{R}_{j,d_i} & \mathbb{R}_{j,d_i+1} \\ & & & & \\ 0 & & \cdots & & 0 \end{bmatrix}}.$$

(C27)

The j^{th} row of the result can be expressed as a row vector, $\mathbf{v}^5 \in \mathbb{R}^{d_i+1}$, which can be computed as

$$\mathbf{v}^5_\alpha = \mathbf{W}_{i,i} x_{i,\alpha}. \quad \text{(C28)}$$

We can now compute **M**, defined as

$$\mathbf{M} = \mathbf{X}^T \left(\mathbf{W}\frac{\partial \mathbf{X}}{\partial x_{i,j}} + \frac{\partial \mathbf{W}}{\partial x_{i,j}}\mathbf{X} \right) + \left(\frac{\partial \mathbf{X}}{\partial x_{i,j}}\right)^T \mathbf{W}\mathbf{X}. \quad \text{(C29)}$$

The left side of the summation is provided in Equation C26, and the right side is provided in Equations C27 and C28. The algorithm for computing **M** is defined in Algorithm C1.

Algorithm for Computing M = $\left(d_i+1\times d_i+1\right)$:

1. **For all** $t,u\in\mathbb{N}$ where $1\le t,u\le d_i+1$

   If $u\neq j$

   Then

   Set $\mathbf{M}_{t,u}=X_{t,i}^{T}\left(\frac{\partial\mathbf{W}}{\partial x_{i,j}}\right)_{i,i}x_{i,u}$

   Else

   Set $\mathbf{M}_{t,u}=X_{t,i}^{T}\left(\left(\frac{\partial\mathbf{W}}{\partial x_{i,j}}\right)_{i,i}x_{i,u}+\mathbf{W}_{i,i}\right)$

2. **For** $\alpha=1$ **to** d_i+1

   Do $\mathbf{M}_{j,\alpha}=\mathbf{M}_{j,\alpha}+\mathbf{W}_{i,i}x_{i,\alpha}$

Algorithm C1: Algorithm for computing **M**.

We can now compute $\mathbf{r}_2$, as

$$\mathbf{r}_2=\boldsymbol{\theta}\mathbf{M}\boldsymbol{\theta}\boldsymbol{\varphi}. \tag{C30}$$

This step takes $O\left(d_i^3+d_i^2d_o+d_id_o+d_o+d_ipd_o\right)$ time, most of which is accrued during the calculation of Equation C30. Using $\mathbf{r}_1$, provided in Equation C18 and $\mathbf{r}_2$, provided in Equation C30, we can compute the final result,

$$\frac{\partial\boldsymbol{\beta}}{\partial x_{i,j}}=\mathbf{r}_1-\mathbf{r}_2. \tag{C31}$$

Computing $\mathbf{r}_1$ took $O\left(d_i^2 d_o + d_o d_i\right)$, which is dwarfed by the $O\left(d_i^3 + d_i{}^2 d_o + d_i p^2 + d_i p d_o\right)$ time to compute $\mathbf{r}_2$, making the time to compute the final result $O\left(d_i^3 + d_i{}^2 d_o + d_i p^2 + d_i p d_o\right)$ which is polynomial with respect to the dimension of the inputs, outputs, and the number of knowledge points included in the regression.

Recall that only the bottom row of $\partial\boldsymbol{\beta}/\partial x_{i,j}$ is needed, so a slight performance increase can be achieved by only computing the bottom rows of $\mathbf{r}_1$ and $\mathbf{r}_2$. When computing $\mathbf{r}_1$, this means only computing the bottom row in the last step of multiplying by $\boldsymbol{\theta}$ on the left-hand side (step 3 in Algorithm C2). When computing $\mathbf{r}_2$, (step 5 in Algorithm C2) this means that only the bottom rows of $\boldsymbol{\theta}\mathbf{M}$ and $\left(\boldsymbol{\theta}\mathbf{M}\right)\left(\boldsymbol{\theta}\boldsymbol{\varphi}\right)$ must be computed. The complete algorithm for computing $\partial\boldsymbol{\beta}/\partial x_{i,j}$ is provided as Algorithm C2.

Algorithm C2 for computing : $\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}}$

Given: $i, j,\ \mathbf{X},\ \mathbf{W},\ d_i$ (input dimension), d_o (output dimension)

1. $\forall t \in [1, d_i + 1], \forall u \in [1, d_o], [\hat{\mathbf{r}}_1]_{t,u} = \begin{cases} \mathbf{X}_{t,i}^T \left(\frac{\partial \mathbf{W}}{\partial x_{i,j}} \right)_{i,i} \mathbf{y}_{i,u}, & t \neq j, \\ \mathbf{X}_{t,i}^T \left(\frac{\partial \mathbf{W}}{\partial x_{i,j}} \right)_{i,i} \mathbf{y}_{i,u} + W_{i,i} y_{i,u}, & t = j. \end{cases}$
2. $\boldsymbol{\theta} = \left(\mathbf{X}^T \mathbf{W} \mathbf{X} \right)^{-1}$
3. $\mathbf{r}_1 \leftarrow \boldsymbol{\theta} \hat{\mathbf{r}}_1$
4. Execute Algorithm C1, to compute $\mathbf{M}$
5. Compute $\boldsymbol{\varphi} = \mathbf{X}^T \mathbf{W} \mathbf{y}$
6. $\mathbf{r}_2 = \boldsymbol{\theta} \mathbf{M} \boldsymbol{\theta} \boldsymbol{\varphi}$
7. $\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}} = \mathbf{r}_1 - \mathbf{r}_2$

Algorithm C2: Algorithm for computing $\frac{\partial \boldsymbol{\beta}}{\partial x_{i,j}}$, which is required for DI-ILWR (Chapter 3).

APPENDIX D

This appendix contains the derivation of Equation 3.4, from Equations 3.2 and 3.3:

$$\Delta w = -\eta \nabla E(w) \tag{3.2}$$

$$E(w) \equiv \frac{1}{2}\sum_{i=1}^{d_o}\left(y_{q,i} - \hat{y}_{q,i}\right)^2 \tag{3.3}$$

$$w_i \leftarrow w_i - \eta \cdot \sum_{\alpha=1}^{d_o}\left[\left(\hat{y}_{q,\alpha} - y_{q,\alpha}\right)\frac{\partial \hat{y}_{q,\alpha}}{\partial w_i}\right]. \tag{3.4}$$

We start by writing Equation 3.2 as an update for each individual weight,

$$\Delta w_i = -\eta \cdot \nabla E(w_i). \tag{D1}$$

Next we compute the gradient of $E(w_i)$:

$$\nabla E(w_i) = \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{\alpha=1}^{d_o}\left(y_{q,\alpha} - \hat{y}_{q,\alpha}\right)^2\right) \tag{D2}$$

$$= \frac{1}{2}\sum_{\alpha=1}^{d_o}\frac{\partial\left(y_{q,\alpha} - \hat{y}_{q,\alpha}\right)^2}{\partial w_i} \tag{D3}$$

$$= \frac{1}{2}\sum_{\alpha=1}^{d_o}\left[2\left(y_{q,\alpha} - \hat{y}_{q,\alpha}\right)\frac{\partial\left(y_{q,\alpha} - \hat{y}_{q,\alpha}\right)}{\partial w_i}\right] \tag{D4}$$

$$= \sum_{\alpha=1}^{d_o}\left[\left(y_{q,\alpha} - \hat{y}_{q,\alpha}\right)(-1)\frac{\partial \hat{y}_{q,\alpha}}{\partial w_i}\right] \tag{D5}$$

$$= \sum_{\alpha=1}^{d_o} \left[\left(\hat{y}_{q,\alpha} - y_{q,\alpha} \right) \frac{\partial \hat{y}_{q,\alpha}}{\partial w_i} \right]. \tag{D6}$$

Substituting Equation D6 into Equation 3.2, we obtain Equation 3.4.

APPENDIX E

The parameters used by the DAS1 simulations are provided in the following setup files.

First, `arm.bio` reads:

```
# arm.bio
# muscles
#
# six muscles for the planar arm model
# Fmax and moment arms from Bhushan & Shadmehr, Biol Cybern 1999

verbose_level 0

joint SHOULDER
     distal_body upperarm
     limits -90 180
     end

joint ELBOW
     distal_body forearm
     limits 0 180
     end

muscle default
     a 0.25
     vmrel 10
     umax 0.04
     fecmax 1.5
     krel 0.0
     slopfac 2.0
     PEEslack 1.0
     time_constants 0.040 0.060
     end

# note: the following values of lceopt and lslack were
#   taken from
#     Garner, B.A., Pandy, M.G. "Estimation of Musculotendon
#     Properties in the Human Upper Limb." Annals of
#     Biomedical Engineering, February 2003, vol. 31,
#     no. 2, pp. 207 - 220.
#        Specifically, the values were taken from the
#        "model" column of Table 3 (p. 216).


muscle ANT_DELTOID
     fmax 800
     lceopt 0.1280 lslack 0.0538 width 1.0
```

```
     geometry pulley 0.1840 SHOULDER 0.05 end
     end

muscle POST_DELTOID
     fmax 800
     lceopt 0.1280 lslack 0.0538 width 1.0
     geometry pulley 0.1840 SHOULDER -0.05 end
     end

muscle BRACHIALIS
     fmax 700
     lceopt 0.1028 lslack 0.0175 width 1.0
     geometry pulley 0.1210 ELBOW 0.03 end
     end

muscle TRICEPS_SH
     fmax 700
     lceopt 0.0877 lslack 0.1905 width 1.0
     geometry pulley 0.2858 ELBOW -0.03 end
     end

muscle TRICEPS_LH
     fmax 1000
     lceopt 0.0877 lslack 0.1905 width 1.0
     geometry pulley 0.2858 SHOULDER -0.03 ELBOW -0.03 end
     end

muscle BICEPS
     fmax 1000
     lceopt 0.1422 lslack 0.2298 width 1.0
     geometry pulley 0.3812 SHOULDER 0.03 ELBOW 0.03 end
     end

end
```

Next, `arm.torques.bio` reads:

```
# arm.bio
# torques  // this line necessary for reinforcement learning
program, and it must be the 2nd line!
#
# six muscles for the planar arm model
# Fmax and moment arms from Bhushan & Shadmehr, Biol Cybern 1999

verbose_level 0

joint SHOULDER
     distal_body upperarm
     limits -90 180
     end
```

```
joint ELBOW
     distal_body forearm
     limits 0 180
     end

muscle default
     a 0.25
     vmrel 10
     umax 0.04
     fecmax 1.5
     krel 0.0
     slopfac 2.0
     PEEslack 1.0
     time_constants 0.040 0.060
     end

# note: the following values of lceopt and lslack were
#   taken from
#     Garner, B.A., Pandy, M.G. "Estimation of Musculotendon
#     Properties in the Human Upper Limb." Annals of
#     Biomedical Engineering, February 2003, vol. 31,
#     no. 2, pp. 207 - 220.
#        Specifically, the values were taken from the
#        "model" column of Table 3 (p. 216).


end
```

Finally, `arm_info` reads:

```
SD/FAST Information File: arm.sd
Generated 30-May-2004 12:33:04 by SD/FAST, Kane's formulation
(sdfast B.2.8 #30123) on machine ID unknown

ROADMAP (arm.sd)

Bodies        Inb
No  Name      body Joint type  Coords q
--- --------- ---- ----------- ----------------
 -1 $ground
  0 upperarm   -1  Pin           0
  1 forearm     0  Pin           1


STATE INDEX TO JOINT/AXIS MAP (arm.sd)
```

```
 Index
  q|u   Joint  Axis   Joint type    Axis type    Joint Name
 -----  -----  ----   -----------   ----------   ----------
  0|2       0     0   Pin           rotate
  1|3       1     0   Pin           rotate


SYSTEM PARAMETERS (arm.sd)

Parameter  Value  Description

nbod           2  no. bodies (also, no. of tree joints)
njnt           2  total number of joints (tree+loop)
ndof           2  no. degrees of freedom allowed by tree joints
nloop          0  no. loop joints
nldof          0  no. degrees of freedom allowed by loop joints

nq             2  no. position coordinates in state (tree
joints)
nu             2  no. rate coordinates in state (tree joints)
nlq            0  no. position coordinates describing loop
joints
nlu            0  no. rate coordinates describing loop joints

nc             0  total no. constraints defined
nlc            0  no. loop joint constraints
npresc         0  no. prescribed motion constraints
nuserc         0  no. user constraints
```

APPENDIX F

Table F1 provides a complete listing of the parameter sets utilized by the continuous actor-critic throughout this thesis. Table F2 provides comments on each parameter set.

	Fast	**Slow**	**A**	**B**	**ILWR**	**ILWR-Pretrain**	**General**	**Pendulum Swing-Up**
η_A	70	10	0.001	99.5	70	0	70	5
η_C	0	0.344	0.0001	34.4	0.1	0.1	0.344	1
σ	9,000	9,000	74.5	7991	9,000	9,000	9,000	N/A
τ_N	2,400	2,400	0.55	2,500	2,400	2,400	2,400	1
τ	0.1	0.1	1	1	0.1	0.1	0.1	1
κ	0.1	0.1	0.55	71.5	0.1	0.1	0.1	1
k_A	0	0	0	0	0	0	0.0000002	0
k_C	0	0	0	0	0	0	0.000002	0
D	N/A	N/A	N/A	N/A	$diag(.5, .5, .3, .3, .5, .5)$	$diag(.5, .5, .3, .3, .5, .5)$	N/A	N/A

Table F1: A complete listing of parameter sets for the continuous actor-critic implementations in this thesis.

Fast	Uses pre-trained ANN actor and ANN critic-10
Slow	Uses pre-trained ANN actor and ANN critic-10
A	Uses pre-trained ANN actor and ANN critic-20
B	Uses pre-trained ANN actor and ANN critic-20
ILWR	Uses pre-trained ANN actor and ILWR critic
ILWR-Pretrain	Uses pre-trained ANN actor and random initial ILWR critic
General	Uses pre-trained ANN actor and ANN critic-10
Pendulum Swing-Up	Uses random initial ANNs for the actor and critic

Table F2: Comments on the usage of each parameter set from Table F1.

REFERENCES

[1] Abbas, J.J., and Triolo, R.J. (1997). Experimental Evaluation of an Adaptive Feedforward Controller for use in Functional Neuromuscular Stimulation Systems. *IEEE Transactions on Rehabilitation Engineering*, 5(1): 12–22.

[2] Atkeson, C.G., Moore, A.W., Schaal, S. (1997). Locally Weighted Learning. *Artificial Intelligence Review*. 11–73.

[3] Barto, A.G., Sutton, R.S., and Anderson, C.W. (1988). Neuronlike Elements that can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:835-846, 1983. Reprinted in J. A. Anderson and E. Rosenfeld, *Neurocomputing: Foundations of Research*, MIT Press, Cambridge, Massachusetts.

[4] Baxt, W.G. (1990). Use of an Artificial Neural Network for Data Analysis in Clinical Decision-Making: The Diagnosis of Acute Coronary Occlusion. *Neural Computation*, 2: 480–489.

[5] Bellman, R. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics* 6.

[6] Blana, D., Kirsch, R.F., Chadwick, E.K. (2009). Combined Feedforward and Feedback Control of a Redundant, Nonlinear, Dynamic Musculoskeletal System. *Medical and Biological Engineering and Computing*. 47:533–542.

[7] Braz, G.P., Russold, M., Smith, R.M., and Davis, G.M. (2007). Electrically-Evoked Control of the Swinging Leg After Spinal Cord Injury: Open-Loop or Motion Sensor-Assisted Control? *Australasian Physical Engineering Sciences in Medicine*, 30(4): 317–323.

[8] Chadwick, E.K., Blana, D., van den Bogert, A.J., and Kirsch, R.F. (2009). A Real-Time, 3-D Musculoskeletal Model for Dynamic Simulation of Arm Movements. *IEEE Transactions on Biomedical Engineering*, 56(4): 941–948.

[9] Chang, G.C., Luh, J.J., Liao, G.D., Lai, J.S., Cheng, C.K., Kuo, B.L., and Kuo, T.S. (1997). A Neuro-Control System For the Knee Joint Position Control With Quadriceps Stimulation. *IEEE Transactions on Rehabilitation Engineering*, 5(1): 2–11.

[10] Crago, P.E., Lan, N., Veltink, P.H., Abbas, J.J., and Kantor, C. (1996). New Control Strategies for Neuroprosthetic Systems. *Journal of Rehabilitation Research and Development*, 33(2): 158–172.

[11] Crago, P.E., Nakai, R.J., and Chizeck, H.J. (1991). Feedback Regulation of Hand Grasp Opening and Contact Force During Stimulation of Paralyzed Muscle. *IEEE Transactions on Biomedical Engineering*, 38(1): 17–28.

[12] Davoodi, R., and Andrews, J.B. (1998). Computer Simulation of FES Standing Up in Paraplegia: A Self-Adaptive Fuzzy Controller With Reinforcement Learning. *IEEE Transactions on Rehabilitation Engineering* 6(2): 151–161.

[13] Doya, K. (2000). Reinforcement Learning in Continuous Time and Space. *Neural Computation*, 12(1):219–245.

[14] Edwards, C.H., Penney, D.E. (2002). *Calculus, Matrix Version*, Englewood Cliffs, NJ: Prentice Hall. Sixth Edition.

[15] Ferrarin, M., Pavan, E.E., Spadone, R., Cardini, R., and Frigo, C. (2002). Standing-Up Exerciser Based on Functional Electrical Stimulation and Body Weight Relief. *Medical and Biological Engineering and Computing*, 40(3): 282–289.

[16] Franklin, G.F., Powell, D.J. Workman, L.M. (1997). *Digital Control of Dynamic Systems*, Englewood Cliffs, NJ: Prentice Hall. Third Edition.

[17] Gullapalli, V. (1990). A Stochastic Reinforcement Learning Algorithm for Learning Real-Valued Functions. *Neural Networks*, 3:671–192.

[18] Hagen, S.T., Kröse, B. (2000). Q-learning for Systems with Continuous State and Action Spaces. *10th Belgian-Dutch Conference on Machine Learning*, Tilburg, The Netherlands, 13 December.

[19] Hutchinson, J.M. (1994). *A Radial Basis Function Approach to Financial Time Series Analysis*. Ph.D. dissertation, Massachusetts Institute of Technology.

[20] Izawa, J., Toshiyuki, K., and Koji, I. (2004). Biological Arm Motion Through Reinforcement Learning. *Biological Cybernetics*, 91(1): 10–22.

[21] Izhikevich, E.M. (2007). *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*, MIT Press, Cambridge, Massachusetts.

[22] Jaeger, R.J. (1992). Lower Extremity Applications of Functional Neuromuscular Stimulation. *Assistive Technology*, 4(1): 19–30.

[23] Jagodnik, K.M., and van den Bogert, A.J. (2007). A Proportional Derivative FES Controller for Planar Arm Movement. *12th Annual Conference International FES Society*, Philadelphia.

[24] Jagodnik, K.M., van den Bogert, K.M., Branicky, M.B., Thomas, P.S. (2008). A Proportional Derivative Controller for Planar Arm Movement. *North American Congress on Biomechanics (NACOB)*, Ann Arbor, Michigan, 5–9 August. Poster.

[25] Kaelbling, L.P., Littman, M.L., and Moore, A.W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, Volume 4.

[26] Klassen, M., Pao, Y.H., Chen, V. (1988). Characteristics of the Functional Link Net: A Higher Order Deltarule Net. *Proceedings of the IEEE Second Annual International Conference on Neural Networks*, San Diego, California, July.

[27] Kobetic, R., and Marsolais, E.B. (1994). Synthesis of Paraplegic Gait With Multi-Channel Functional Neuromuscular Stimulation. *IEEE Transactions on Rehabilitation Engineering*, 2: 66–79.

[28] Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., and Jackel, L. D. (1990). Handwritten Digit Recognition with a Back-propagation Network. *Advances in Neural Information Processing Systems*, 2: 248–257.

[29] Leung, M.T., Engeler, W.E., and Frank, P. (1990). Fingerprint Processing using Backpropagation Neural Networks. *Proceedings of the International Joint Conference on Neural Networks I*, 15–20.

[30] Lynch, L.C.; and Popovic, R.M. (2008). Functional Electrical Stimulation: Closed-Loop Control of Induced Muscle Contractions. *IEEE Control Systems Magazine*, 28(2): 40–50.

[31] McLean, S.G., Su, A., and van den Bogert, A.J. (2003). Development and Validation of a 3-D Model to Predict Knee Joint Loading During Dynamic Movement. *Journal of Biomechanical Engineering*, 125(6): 864–874.

[32] Mitchell, T. (1997). *Machine Learning*. Singapore: McGraw-Hill.

[33] Peckham, P.H., Keith, M.W., Kilgore, K.L., Grill, J.H., Wuolle, K.S., et al. (2001). Efficacy of an Implanted Neuroprosthesis for Restoring Hand Grasp in Tetraplegia: A Multicenter Study. *Archives of Physical Medicine and Rehabilitation*, 82: 1380–1388.

[34] Peckham, P.H., and Knutson, J.S. (2005). Functional Electrical Stimulation for Neuromuscular Applications. *Annual Review of Biomedical Engineering*, 7: 327–360.

[35] Pomerleau, D.A. (1993). *Neural Network Perception for Mobile Robot Guidance*. Boston: Kluwer.

[36] Ragnarsson, K.T. (2008). Functional Electrical Stimulation After Spinal Cord Injury: Current Use, Therapeutic Effects and Future Directions. *Spinal Cord*, 46(6): 255–274.

[37] Russell, S., and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Englewood Cliffs, NJ: Prentice Hall. Second Edition.

[38] Schaal, S., Atkeson, C.G., and Vijayakumar, S. (2002). Scalable Techniques From Nonparametric Statistics for Real Time Robot Learning. *Applied Intelligence*, 17:49–60.

[39] Schultz, A.B., Faulkner, J.A., and Kadhiresan, V.A. (1991). A Simple Hill Element-Nonlinear Spring Model of Muscle Contraction Biomechanics. *Journal of Applied Physiology*, 70(2): 803–812.

[40] Sejnowski, T.J., Yuhas, B.P., Goldstein, M.H., Jr., and Jenkins, R.E. (1990). Combining Visual and Acoustic Speech Signals with a Neural Network Improves Intelligibility. *Advances in Neural Information Processing Systems*, 2: 232–239.

[41] Shannon, C.E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine*, 41:256–275.

[42] Shea, P.M., and Liu, F. (1990). Operational Experience with a Neural Network in the Detection of Explosives in Checked Airline Baggage. *Proceedings of the International Joint Conference on Neural Networks*, Vol. II, 175–178.

[43] Sheffler, L.R., and Chae, J. (2007). Neuromuscular Electrical Stimulation in Neurorehabilitation. *Muscle Nerve*, 35(5): 562–590.

[44] *Spinal Cord Injury Facts and Figures at a Glance.* (2008). National Spinal Cord Injury Statistical Center (NSCISC), Birmingham, Alabama, January.

[45] Stroeve, S. (1996). Learning Combined Feedback and Feedforward Control of a Musculoskeletal System. *Biological Cybernetics*, 75(1): 73–83.

[46] Sujith, O.K. (2008). Functional Electrical Stimulation in Neurological Disorders. *European Journal of Neurology*, 15(5): 437–444.

[47] Sukharev, A.G. (1971). Optimal Strategies of the Search for an Extremum. *U.S.S.R. Computational Mathematics and mathematical Physics*, 11(4). Translated from Russian, *Zh. Vychisl. Mat. i Mat. Fiz.*, 11, 4, 910–924.

[48] Sutton, R.S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts.

[49] Thomas, P.S., Branicky, M.B., van den Bogert, A.J., and Jagodnik, K.M. (2008a). Creating a Reinforcement Learning Controller for Functional Electrical Stimulation of a Human Arm. *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*, New Haven, Connecticut, 1–6 June.

[50] Thomas, P.S., Branicky, M.B., van den Bogert, A.J., and Jagodnik, K.M. (2008b). Creating a Reinforcement Learning Controller for Functional Electrical Stimulation of a Human Arm. *Research ShowCase*. Case Western Reserve University, Cleveland, Ohio, 17 April. Poster.

[51] Thomas, P.S., Branicky, M.B., van den Bogert, A.J., and Jagodnik, K.M. (2008c). FES Control of a Human Arm using Reinforcement Learning. *Adaptive Movements in Animals and Machines (AMAM)*, Cleveland, Ohio, 1–6 June. Poster.

[52] Thomas, P.S., Branicky, M.B., van den Bogert, A.J., and Jagodnik, K.M. (2009a). Application of the Actor-Critic Architecture to Functional Electrical Stimulation Control of a Human Arm. *Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence Conference*, Pasadena, 14–16 July. To appear.

[53] Thomas, P.S., Branicky, M.B., van den Bogert, A.J., and Jagodnik, K.M. (2009b). Achieving Long-Term Stability using a Reinforcement Learning Controller for Functional Electrical Stimulation Control of a Human Arm. *Research ShowCase*. Case Western Reserve University, Cleveland, Ohio, 16 April. Poster.

[54] Vijayakumar, S., D'Souza, A., Schaal, S. (2005). Incremental Online Learning in High Dimensions. *Neural Computation*, 17(12):2602–2634.

[55] Watkins, C.J.C.H. (1989). *Learning from Delayed Rewards*. PhD thesis, Cambridge University, Cambridge, England.

[56] Wedge, N.A. (2004). *Analysis of Cellular Cardiac Bioelectricity Models Toward Computationally Efficient Whole-Heart Simulation*. MS thesis, Case Western Reserve University, Cleveland, Ohio.

[57] Witten, I. H. and Corbin, M. J. (1973). Human Operators and Automatic Adaptive Controllers: A Comparative Study on a Particular Control Task. *International Journal of Man-Machine Studies*, 5:75–104.

[58] Witten, I.H. (1977). An Adaptive Optimal Controller for Discrete-time Markov Environments. *Information and Control*, 34:286–295.

[59] Wheeler, G.D., Andrews, B., Lederer, R., Davoodi, R., Natho, K., Weiss, C., Jeon, J., Bhambhani, Y., and Steadward, R.D. (2002). Functional Electrical Stimulation-Assisted Rowing: Increasing Cardiovascular Fitness Through Functional Electrical Stimulation Rowing Training in Persons With Spinal Cord Injury. *Archives of Physical Medicine and Rehabilitation*, 83(8):1093–1099.