CHAPTER 1

STABLE ADAPTIVE NEURAL CONTROL OF PARTIALLY OBSERVABLE DYNAMIC SYSTEMS

J. Nate Knight, Ph.D. [1], Charles W. Anderson, Ph.D. [1]

[1]Department of Computer Science, Colorado State University, Ft Collins, CO

1.1 INTRODUCTION

The control of physical systems requires accounting for uncertainty. Uncertainty enters the control problem in at least three ways: unmeasured states, unknown dynamics, and uncertain parameters. Modern robust control theory is based on explicit mathematical models of uncertainty [6]. If it is possible to describe what is unknown about a system, stronger assurances can be made about its stability and performance. The automation of robust controller design relies on the tractable representation of the uncertainty in a system. Some types of uncertainty can be described by integral quadratic constraints (IQCs) and lead to representations of uncertainty as convex sets of linear operators. Linear systems are a particularly tractable type of model, and the design of feedback controllers for linear systems is a well understood problem. Most physical systems, however, exhibit some nonlinear dynamics and linear models are generally insufficient for accurately describing them.

Stable Adaptive Neural Control of Partially Observable Dynamic Systems. By J. Nate Knight and Charles W. Anderson

Unmodeled nonlinear dynamics can often be treated within the same framework as system uncertainty. Because robust controllers must be insensitive to inaccuracies and uncertainties in system models, performance is often suboptimal on the actual system to which the controller is applied. Additional loss in performance is often introduced by restricting controllers to be linear and of low order. This restriction is usually made because linear controllers can be easily analyzed and understood. Control performance can often be improved in these situations by the use of nonlinear and adaptive control techniques. Guaranteeing stability and performance, however, is more difficult in this environment. In this chapter we examine the use of adaptive, recurrent neural networks in control systems where stability must be assured.

Recurrent neural networks are, in some respects, ideal for applications in control. The nonlinearity in neural networks allows for the compensation of nonlinearties in system dynamics that is not generally possible with low order, linear controllers. The dynamics of recurrent neural networks allow internal models of unmeasured states to be produced and used for control. The difficulty in applying recurrent neural networks in control systems, however, is in the analysis and prediction of the system's behavior for the purpose of stability analysis.

The control of a nonlinear, uncertain, partially-observable, multiple spring-mass-damper system is considered in this chapter. The system is a simple instance of a larger class of models representing physical systems such as flexible manipulators and active suspension systems. We present a recurrent neural controller for the system with guaranteed stability during operation and adaptation. A reinforcement learning algorithm is used to update the recurrent neural network's weights to optimize control system tracking performance and a stability analysis based on IQC models is used to reject weight updates that can not be guaranteed to result in stable behavior [10, 11]. In this chapter, we develop a stability bias that is applied to the network's weight trajectory to improve the performance of the algorithm.

1.2 BACKGROUND

The following simple approach is one way to guarantee the stability of an adaptive control system as it changes through time. Compute a set of bounds on the variation of the control parameters within which stability is guaranteed, and then filter out any parameter updates that put the parameters outside of the computed bounds. Such an approach is at one end of a trade-off between computational cost and conservativeness. Only a single stability analysis is required, and the cost of rejecting weight updates that can not be proved stable is trivial. On the other hand, the initial weights may not be close to the optimal weights and the bounds may limit optimization of the problem objective. In [18] this approach was applied to training an RNN to model a chaotic system. Given a *good* initial weight matrix, the learning algorithm was

able to improve the model within the specified stability bounds. In general, however, it can not be expected that the optimal weight matrix, $\bar{W}$, for a problem will be reachable from an intial weight matrix, $W(0)$, while still respecting the initial stability constraints. The relative inexpensiveness of this approach has as its price a reduction in the achievable performance. At the other end of the spectrum is an algorithm that recomputes the bounds on weight variations at every update to the weights. The algorithm does not ensure that every update is accepted, but it does, in theory, result in the acceptance of many more updates than the simple approach. It also allows, again, in theory, better performance to be achieved. The computational cost of the algorithm is, however, prohibitively expensive because of the large number of stability analysis computations required.

In previous work [11, 10, 1, 12], we describe an algorithm that falls somewhere between these two extremes allowing better optimization of the objective than the first approach with less computational expense than the second. The algorithm assumes that changes to the control parameters are proposed by an external agent, such as the reinforcement learning agent described in a later section, at discrete time steps indexed by the variable k. The algorithm ensures the stability of an adaptive control system by filtering parameter updates that can not be guaranteed to result in stable behavior. A constraint set, $\mathcal{C}_j(W)$ is a set of bounds, $\{\underline{\Delta}, \overline{\Delta}\}$, on the variation in the parameters centered on the fixed parameter vector W. Variations in $W(k)$ that stay within these bounds can be assured not to result in instability. The constraint set, $\mathcal{C}_j(W)$, is a volume centered on W with each element of $W(k)$ constrained by

$$W_i - \underline{\Delta}_i \leq W_i(k) \leq W_i + \overline{\Delta}_i.$$

When an update causes $W(k)$ to lie outside of the constraint set, a new set of constraints is computed if updates to $W(k)$ have occurred since the last set of constraints was constructed. Otherwise, the update is rejected. Given this new set of constraints centered on the most recently seen stable W, the current update $W(k-1) + \Delta W$ is again checked for validity. If the update fails to satisfy the new constraints it is then rejected. Rather than rejecting the update outright, the procedure described here makes better use of the available parameter update suggestions from the adaptation algorithm.

Recurrent neural networks (RNNs) are a large class of both continuous and discrete time dynamical systems. RNN formulations range from simple ordinary differential equation (ODE) models to elaborate distributed and stochastic system models. The main focus of this work is on the application of RNNs to control problems. In this context, RNNs can be seen as input-output maps for modeling data or acting as controllers. For these types of tasks, it will be sufficient to restrict attention to continuous time RNN formulations, primarily of the form

$$\begin{aligned} \dot{x} &= -Cx + W\Phi(x) + u \\ y &= x. \end{aligned} \tag{1.1}$$

Here, x is the state of the RNN, u is a time varying input, y is the output of the network, C is a diagonal matrix of positive time constants, W is the RNN's *weight matrix* and Φ is a nonlinear function of the form

$$\Phi(x) = [\phi(x_1)\ \phi(x_2)\ \ldots\ \phi(x_n)]^T.$$

The function $\phi(x)$ is a continuous one dimensional map, and generally a sigmoid like function, such as $\tanh(x)$. Since the RNN will be applied as an input-output map, the output, denoted by y, is defined to be the state x. More general models allow the selection of certain states as outputs or an additional mapping to be applied at the output layer. These modifications do not affect the stability analysis of the RNNs dynamics, but need to be considered when the network is used in a control system.

Many algorithms exist for adapting the weights of RNNs. A survey of gradient based approaches can be found in [16]. To solve the example learning problem in this chapter, the Real Time Recurrent Learning (RTRL) algorithm is applied. RTRL is a simple stochastic gradient algorithm for minimizing an error function over the parameters of a dynamic system. It is used here because it is an online algorithm applicable in adaptive control systems. Computing the gradient of an error function with respect to the parameters of a dynamic system is difficult because the system dynamics introduce temporal dependencies between the parameters and the error function. Computation of the error gradient requires explicitly accounting for these dependencies, and RTRL provides one way of doing this.

Letting $F(x, u; C, W) = -Cx + W\Phi(x) + u$ in (1.1), the gradient of the error function $E(||x(t) - \bar{x}(t)||_2^2)$ with respect to the weight matrix, W, is given in RTRL by the equations

$$\begin{aligned}\frac{\partial E}{\partial W} &= \int_{t_0}^{t_1} s\frac{\partial E}{\partial y}dt \\ \frac{\partial s}{\partial t} &= \frac{\partial F(x, u; C, W)}{\partial W} + \frac{\partial F(x, u; C, W)}{\partial x}s,\end{aligned}$$

where $s(t_0) = 0$. The variable s is a rank three tensor with elements s_{ij}^l corresponding to the sensitivity of x_l to changes in the weight W_{ij}. RTRL requires simulation of the s variables forward in time along with the dynamics of the RNN. The gradient, however, need not necessarily be integrated. The weights can instead be updated by

$$W \leftarrow W - \eta s(t)\frac{\partial E(t)}{\partial y(t)}$$

for the update time t. The parameter η is a learning rate that determines how fast the weights change over time. The stochastic gradient algorithm requires that this parameter decrease to zero over time, but often it is simply fixed to a small value. The algorithm has an asymptotic cost of $\Theta(n^4)$ for each update

to s when all the RNN weights are adapted. For large networks the cost is impractical, but improvements have been given. For example an exact update with complexity $\Theta(n^3)$ is given in [19] and an approximate update with an $\Theta(n^2)$ cost is given in [2]. The basic algorithm is practical for small networks and is sufficient for illustrating the properties of the proposed stable learning algorithm.

Without constraints on the weight updates it is clear that the learning algorithm does not respect the stability constraints. This is not to say that the algorithm produces unstable behavior, only that stability can not be guaranteed along the trajectory induced by the unconstrained RTRL algorithm. Such excursions from the stability region could have damaging effects if the RNN was in a control loop with some actual physical system.

Applying the constraint set, $\mathcal{C}_j(W)$, forces the trajectory of $W(k)$ to remain in $\mathcal{W}_{ss}^n$, where $\mathcal{W}_{ss}^n$ is all $n \times n$ weight matrices that result in stable RNN dynamics. In some instances, weight trajectories can remain near the boundary of $\mathcal{W}_{ss}^n$. This results in the computation of a large number of constraints sets. Since this computation is expensive, its occurrence over the course of the weight adaptation should be minimized. Reducing the conservativeness of the stability analysis is not much help in this situation. If the error gradient points toward the stability boundary and the weight trajectory moves toward the boundary, allowing more variation in the interior of $\mathcal{W}_{ss}^n$ does not help. To alleviate this problem the error gradient should be modified with a term that captures stability information. In the next section, a method is introduced that explicitly biases the trajectory away from the boundary of $\mathcal{W}_{ss}^n$.

1.3 STABILITY BIAS

To bias learning trajectories away from the boundary of $\mathcal{W}_{ss}^n$ a measure of closeness to this boundary is needed. Fortunately, the $\mathcal{L}_2$-gain of the RNN with static weights grows rapidly near the boundary of $\mathcal{W}_{ss}^n$; the $\mathcal{L}_2$-gain acts as the inverse of distance to the boundary. The magnitude of the $\mathcal{L}_2$-gain is not immediately useful as a bias, however, since it contains no information about the direction of the boundary from a weight matrix. On the other hand, the gradient of this value with respect to W carries information about closeness to the boundary and of its direction from W. The computation of this derivative is now summarized.

Consider the general, nonlinear, semidefinite program (SDP) given in [7]

$$\begin{aligned} p^* = \min b^T x \quad \text{s.t.} \quad & x \in \mathbb{R}^n, \\ & \mathcal{B}(x) \preceq 0, \\ & c(x) \leq 0, \\ & d(x) = 0. \end{aligned} \tag{1.2}$$

The Lagrangian of this problem $\mathcal{L} : \mathbb{R}^n \times \mathbb{S}^m \times \mathbb{R}^p \times \mathbb{R}^q \rightarrow \mathbb{R}$, is defined by [7]

$$\mathcal{L}(x, Y, u, v) = b^T x + \mathcal{B}(x) \bullet Y + u^T c(x) + v^T d(x), \tag{1.3}$$

where $Y \in \mathbb{S}^m$, $u \in \mathbb{R}^p$, and $v \in \mathbb{R}^q$ are the Lagrange multiplier variables. The Lagrangian dual function, defined as, [3]

$$g(Y, u, v) = \inf_x \mathcal{L}(x, Y, u, v), \tag{1.4}$$

is a lower bound on the optimal value of (1.2) for all values of the multiplier variables. When the best lower bound given by (1.4), that is,

$$d^* = \max_{Y,u,v} g(Y, u, v) \quad \text{s.t. } Y \succeq 0, u \succeq 0, \tag{1.5}$$

is equal to, p^*, the optimal value of (1.2), the problem is said to satisfy a *strong duality* condition. For convex optimization problems a sufficient condition, known as Slater's condition, for strong duality is the existence of a strictly feasible point. So, if $\mathcal{B}(x)$, $c(x)$, and $d(x)$ are convex functions of x, and there exists an x satisfying, $\mathcal{B}(x) \preceq 0$ and $c(x) < 0$ then $d^* = p^*$.

Often, rather than considering a single SDP a set of related SDPs parameterized by some data θ is of interest. For example, the linear matrix inequality (LMI) stability condition for RNNs with time invariant weights forms a set of SDPs parameterized by W. For parameterized SDPs, the Lagrangian, p^*, and d^* are functions of the problem data and are written $\mathcal{L}(x, Y, u, v; \theta)$, $p^*(\theta)$, and $d^*(\theta)$. Of specific interest is the set of θ for which the SDP satisfies the strong duality condition. Over this set, the affect of perturbing the data, θ, on the optimal solution, $p^*(\theta)$, can be estimated with a Taylor series expansion using the gradient defined by

$$\nabla_\theta p^*(\theta) = \left[\frac{\partial}{\partial \theta_i} p^*(\theta)\right] = \left[\frac{\partial}{\partial \theta_i} d^*(\theta)\right] = \left[\frac{\partial}{\partial \theta_i} \mathcal{L}(\bar{x}, \bar{Y}, \bar{u}, \bar{v}; \theta)\right].$$

This gradient is well defined when the Lagrangian is differentiable with respect to the data which is always the case when it is linear in the parameters of interest. The gradient is a first order approximation to the function $p^*(\theta)$ and gives the direction in the parameter space in which $p^*(\theta)$ increases the most in this approximation.

To specialize this gradient for the stability of an RNN, consider the following optimization problem associated with proving the stability of a time invariant RNN [10]:

$$\bar{\gamma} = \inf_{\gamma,T,P} \gamma \quad \text{s.t.} \tag{1.6}$$

$$\begin{bmatrix} -CP - PC + I & P & PW + T \\ P & -\gamma I & 0 \\ W^T P + T & 0 & -2T \end{bmatrix} < 0, \quad P = P^T, \quad T \in \mathcal{D}_+. \tag{1.7}$$

Here, the decision variables are $x = (\gamma, T, P)$. An upper bound on the gain of the RNN with weight matrices C and W is given by $\sqrt{\bar{\gamma}}$. The LMI constraint in the problem has an associated Lagrange multiplier denoted Y. Take the Lagrangian to be a function of the weight matrix, W, and write $\mathcal{L}(x, Y; W)$. The problem is convex and by the definition of $\mathcal{W}_{ss}^n$ satisfies Slater's condition for all W in $\mathcal{W}_{ss}^n$. The Lagrangian takes the value $\bar{\gamma}$ at the solution to (1.6). Since the Lagrangian is linear in W, and thus differentiable, the gradient of $\bar{\gamma}$ with respect to W can be computed by the formula

$$\nabla_W \bar{\gamma} = \left[\frac{\partial}{\partial W_{ij}} \mathcal{L}\left(\bar{x}, \bar{Y}; W\right) \right]. \tag{1.8}$$

For conciseness $\nabla_W \bar{\gamma}$ will be denoted by ∇_s throughout the remainder of the chapter. The Lagrangian in (1.3) specializes to

$$\mathcal{L}(x, Y; W) = \gamma + \mathcal{B}(x) \bullet Y$$

in this case. The function $\mathcal{B}(x)$ corresponds to the left hand side of the LMI constraint in (1.7). It is a function of the problem data W given by

$$\mathcal{B}(x; W) = \sum_i x_i B^{(i)}(W)$$

where the $B^{(i)}$s are linear functions taking W into $\mathbb{S}^m$. Since this is the only point at which the problem data enters the Lagrangian, its gradient is easily computed as

$$\left[\frac{\partial}{\partial W_{ij}} \mathcal{L}\left(\bar{x}, \bar{Y}; W\right) \right] = \left(\sum_{i=1}^{n} \bar{x}_i \frac{\partial}{\partial W_{ij}} B^{(i)}(W) \right) \bullet \bar{Y}. \tag{1.9}$$

Since $\mathcal{B}(x; W)$ is linear in W, the partial derivatives $\frac{\partial}{\partial W_{ij}} B^{(i)}(W)$ are constant and can be computed without the knowledge of a specific W. Evaluation of the gradient is a simple operation, but requires the optimal values $\bar{x}$ and $\bar{Y}$ associated with a given W. This in turn requires that the optimization problem (1.6) be solved for each evaluation of the stability bias. Solving the optimization problem (1.6) dominates the computational cost of computing ∇_s. This computation has a run time on the order of $\Theta(n^3)$ to $\Theta(n^6)$ depending on the IQCs used and the choice of SDP solver [10].

Ideally, the stability bias would affect only the path the parameters take to an optimal solution and not the final point to which they converge. To achieve this ideal behavior the effect of the stability bias on the weight trajectories should be small relative to the effect of the updates given by the learning algorithm and should diminish over time. For example, the update rule might scale the magnitude of ∇_s with respect to the magnitude of the learning update, $\eta\nabla_l$, where, for example, $\nabla_l = s(t)\frac{\partial E(t)}{\partial y(t)}$ in the case of RTRL. Such an update would look like

$$W \leftarrow W - \left(\eta\nabla_l + \eta_2 \frac{\|\eta\nabla_l\|_2}{\|\nabla_s\|_2} \nabla_s \right) \tag{1.10}$$

with $\eta_2 < 1$ and decreasing with time. A potential problem with this approach is that the effect of the stability bias is small regardless of closeness to the boundary of $\mathcal{W}_{ss}^n$. An alternative updating scheme varies the scale of the stability bias with the magnitude of the $\mathcal{L}_2$-gain bound, γ. The update is given by

$$\begin{aligned} \eta_3 &\leftarrow \eta_2 \frac{3}{4}(\tanh(\gamma - \hat{\gamma}) + 1) \\ W &\leftarrow W - \left(\eta \nabla_l + \eta_3 \frac{\|\eta \nabla_l\|_2}{\|\nabla_s\|_2} \nabla_s\right) \end{aligned} \tag{1.11}$$

where $\eta_2 \leq 1$ and decreases with time as before. The scaling, η_3 is a monotonically increasing function of γ with a transition from low to high centered at $\hat{\gamma}$. This approach has the benefit of not affecting the weight trajectory when it is far from the boundary of $\mathcal{W}_{ss}^n$, but requires the transition point $\hat{\gamma}$ to be chosen. The scaling η_2 must decrease to zero with time to ensure that asymptotically the stability bias has no effect. If a certain maximum gain value is desired for the system, allowing η_3 to remain positive and setting $\hat{\gamma}$ to the intended gain bound helps to bias the system toward satisfying this condition. Methods for choosing a decay schedule for the parameter η_2 are rather ad-hoc, but the learning rate η for the RTRL updates has the same requirements and associated problems.

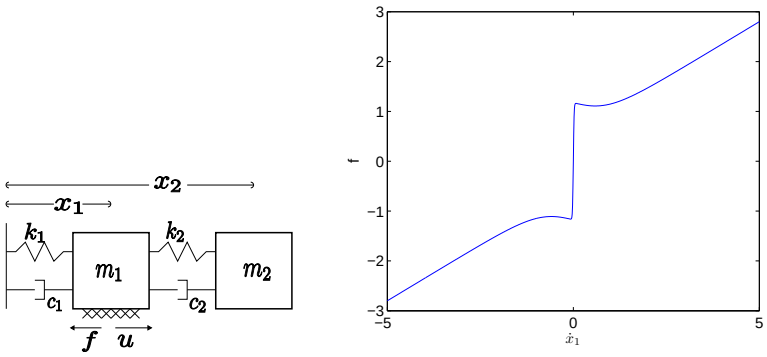
1.4 EXAMPLE APPLICATION

The example applicationed described here illustrates the ability of the simplified stability algorithm to improve control performance by removing instability from the control loop. The stability bias just proposed is integral to the success of the approach.

1.4.1 The Simulated System

The two degree of freedom spring-mass damper system is adapted from the work in [9]. Certain features of the simulated model are considered unknowns and not used explicitly in the adaptation of controllers or stability analysis. The nonlinear model and simulation details are provided in this section. Additionally, an uncertain model of the plant is developed, and an IQC analysis of the closed loop control system is described.

A diagram of the simulated plant is shown in Figure 1.1a. Two masses are connected by nonlinear springs and linear dampers. The first mass is attached via a spring and damper to a stationary point. A control force is be applied to the first mass which is also acted upon by a nonlinear, static, friction force. A position sensor is attached to the second mass. The goal of the control problem is for the second mass to track a time-varying reference signal given by an external agent.



a. b.

Figure 1.1 a.) A multiple spring-mass-damper. b.) A continuously differentiable friction model.

The plant dynamics are governed by the ordinary differential equations

$$\begin{aligned} m_1\ddot{x}_1 + c_1\dot{x}_1 + c_2(\dot{x}_1 - \dot{x}_2) + k_1x_1 + k_2(x_1 - x_2) + h_1x_1^3 + h_2(x_1 - x_2)^3 &= u - f(\dot{x}_1) \\ m_2\ddot{x}_2 + c_2(\dot{x}_2 - \dot{x}_1) + k_2(x_2 - x_1) + h_2(x_2 - x_1)^3 &= 0 \end{aligned} \tag{1.12}$$

where u is the control force applied to the first mass. Actuator dynamics are ignored for the purpose of these experiments, and the control enters the system linearly. The parameters c_1 and c_2 govern the damping force of the two dampers. The spring dynamics are governed by the spring constants k_1 and k_2 and the spring hardening constants $h_1 = h^2k_1$ and $h_2 = h^2k_2$ with $h \geq 0$. The spring hardening constant was set to $h = 0.1$ for all simulations of the system. The friction force is modeled by the equation

$$f(\dot{x}_1) = g_7(g_1(\tanh(g_2\dot{x}_1) - \tanh(g_3\dot{x}_1)) + g_4\tanh(g_5\dot{x}_1) + g_6\dot{x}_1)$$

with

$$(g_1, g_2, g_3, g_4, g_5, g_6, g_7) = (12.5, 50, 1, 11, 50, 9, 0.05).$$

The magnitude of the friction force is shown in Figure 1.1b for $\dot{x}_1 \in [-5, 5]$. The friction equation and parameters are taken from [15] and model both stiction and Coulomb type friction.

All simulations of the system were performed using a variable step size, Dormand-Prince algorithm with an absolute error tolerance of 10^{-5}. Changes in the control signal occur at a rate of 10Hz, and observations are sampled at the same rate. For the purposes of simulation, the parameters were set to $m_1 = 2$, $m_2 = 2.01$, $c_1 = 1.05$, $c_2 = 0.97$, $k_1 = 5.3$, and $k_2 = 4.8$.

1.4.2 An Uncertain Linear Plant Model

For the purposes of controller design and stability analysis an uncertain linear plant model is developed as follows. The parameters of the system are measured as $m_1 = m_2 = 2$, $c_1 = c_2 = 1$, $k_1 = k_2 = 5$ with an uncertainty of 2%, 10%, and 10%, respectively. The parameters are thus assumed to lie in the ranges $m_i \in [1.96, 2.04]$, $c_i \in [.91.1]$, and $k_i \in [4.5, 5.5]$. The simulated plant's parameters are within the assumed measurement errors, and uncertainty models based on these error estimates will be valid.

A linear model of the plant is easily constructed for use in control design and analysis. Ignoring the hardened spring and friction effects in (1.12) yields the linear model

$$\begin{bmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \dot{y}_3 \\ \dot{y}_4 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & m_1 & \\ & & & m_2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -(k_1+k_2) & k_2 & -(c_1+c_2) & c_2 \\ k_2 & -k_2 & c_2 & -c_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}.$$

The set of linear models derived by taking the parameters in the given uncertainty ranges does not completely capture all of the possible dynamics of the system that is to be controlled. The uncertainty model should also account for the nonlinearity in the spring response and the friction force. It is also possible that the system exhibits other unmodeled dynamics. To account for the effect of the unmodeled dynamics, a multiplicative input uncertainty is added to the model. If $G(s)$ is the transfer function of the nominal linear plant model, then the uncertainty model looks like

$$G_{\text{unc}}(s) = G(s)(1 + 0.2\Delta_{\text{lti}})$$

where Δ_{lti} is an unknown, linear, time invariant system with $||\Delta_{\text{lti}}||_\infty < 1$. The unknown linear time-invariant (LTI) system can model up to a 20% deviation in the plant input.

An uncertain real scalar, such as any of the parameters of the linear plant model, can be modeled with the representation $p = (p_n + \delta_p m_p)$ where p_n is the nominal value of the parameter, δ_p is an unknown constant satisfying $|\delta_p| \leq 1$ and m_p is a scaling factor on the uncertainty [14]. To represent the uncertain parameter, $k_1 \in [4.5, 5.5]$, for example, take $p_n = 5$ and $m_p = 0.5$. The uncertain parameters in the model are assumed to be constant. The representation of time-varying parameters is essentially the same, but δ_p is allowed to vary with time. Time varying parameters can have a more varied impact on a system's behavior than constant uncertain parameters. The stability analysis presented below takes advantage of the fact that the parameters are constant to reduce the conservativeness of the analysis.

To perform robustness analysis and control design for the uncertain plant, the uncertainties must be factored out of the plant to form a feedback loop between the known LTI plant and the uncertainties: Δ_{lti} and the δ_p's for the

uncertain parameters. The details of this procedure can be found in standard robust control references such as [14, 6]. Standard robustness analysis shows that the linear plant model is stable for all possible values of the uncertain parameters and all possible input perturbations allowed by the model [14]. An IQC model can be developed and used to show the same result [10]. A bound on the $\mathcal{L}_2$-gain from $u \to y$, computed using the IQC stability theorem, has a value of $\gamma = 1.4755$. The finite gain is a proof of stability for all plants covered by the uncertainty model.

1.4.3 The Closed Loop Control System

The closed loop control system under investigation is depicted in Figure 1.2. A reference signal enters the system from an external source and specifies the desired value of the plant output. In this case, the position of the second mass in the spring-mass-damper system is to be controlled. For the experiments that follow, the reference signal takes values in the range $[-2, 2]$ and can change every 50 seconds. The reference signal might represent, for example, the desired position of a read head in a hard drive or the desired location of the end point of a flexible manipulator.

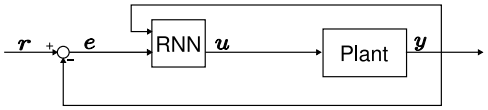


Figure 1.2 The closed loop control system.

Standard robust control designs for reference tracking problems generally use the error signal, $e = r - y$, as the input to the controller. This particular representation is invariant with respect to the position of the observed mass in the multiple spring-mass-damper system. It does not allow the controller to adequately compensate for nonlinearity in the spring, since the nonlinearity in (1.12) is a function of x_1 and $x_1 - x_2$. Even though the position of the first mass is unobserved, the trajectory of x_2 contains information about the true state of the system and thus x_1. It is possible for a recurrent neural network to use this inferred information to improve control performance. For this reason y is also included as an input to the RNN controller. The output of the RNN is the control action, u. The control signal is fed directly into the plant since actuator dynamics are being ignored.

The basic RNN equations must be adapted to fit the desired control structure. The simple RNN model considered in earlier chapters had the same number of inputs, outputs, and states. In the desired control configuration the network has two inputs, a single output, and a number of states set to determine the learning complexity. Input and output weights are added to

the RNN in the following way

$$\begin{aligned} \dot{x_r} &= -Cx_r + W\Phi(x_r) + W^{\mathrm{i}} \begin{bmatrix} e & y \end{bmatrix}^T \\ u &= W^{\mathrm{o}} x_r. \end{aligned} \tag{1.13}$$

Different configurations of W^{i} and W^{o} affect the behavior of the network. A common configuration, see for instance [17], feeds each input into a different node and reads the output from another node. This leads to

$$W^{\mathrm{i}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ \vdots & \\ 0 & 0 \end{bmatrix} \text{ and } W^{\mathrm{o}} = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}.$$

RNN stability analysis can easily be adapted to include the input and output weights [10]. These weights do not affect analysis of the internal stability of the RNN, but they do affect the gain measured from the input to output signals. For instance, if the output weights are all zero then the gain from input to output is zero regardless, even, of the internal stability of the network. It is important, then, that these weights be included in the analysis since the measured gain affects the stability analysis of the closed loop control system.

The analysis of robust control systems focuses on two properties of an uncertain system model: robust stability and robust performance. A robustly stable system is stable for all systems in the given uncertainty set. A system with robust performance, on the other hand, is guaranteed to meet certain performance requirements for all systems in the uncertainty set. Robust stability is necessary for robust performance, but it is not sufficient. Robust stability of a set of uncertain systems implies stability of an actual physical system, *if* the physical system is adequately described by the uncertain model. Increasing the size of the uncertainty set—that is, increasing the range of system dynamics it covers—can allow better assurances to be made about the stability of an actual system. At the same time, increasing the size of the uncertainty set can make it more difficult to prove robust performance.

Performance of a robust control system is measured by the $\mathcal{H}_\infty$ norm of the system. In other words, performance is measured in terms of the $\mathcal{L}_2$-gain from the system's input to its output. Performance objectives in robust control systems are specified by augmenting a given control system with weighted outputs that specify the desired behavior. For example, the weighted outputs might be designed to penalize large actions or low frequency deviations of the plant output from the reference signal. Robust performance of a system is given if the gain from the input to the weighted outputs is less than one. Robust controller synthesis methods, such as the D-K iteration [6], are computational approaches to designing linear controllers that have robust performance for a given uncertain system model.

The synthesis of neural controllers with robust performance guarantees is a difficult problem. The main difficulty arises from treating the RNN nonlinearity as uncertainty in the analysis. The uncertainty descriptions of recurrent neural networksin terms of IQCs have the serious drawback that they do not explicitly model the *boundedness* of the nonlinearities. The resulting uncertainty model and analysis can not distinguish between say, $\phi(x) = x$ and $\phi(x) = \tanh(x)$. Obviously, the resulting dynamics of the two RNNs will be very different. Recent work in [8] on generalized sector conditions may provide a way to incorporate the boundedness of $\phi(x)$ into the analysis, but at present this work is in its infancy. Until such improvements can be made, it is necessary to restrict attention to the analysis of robust stability in neural control systems. The inaccuracy in the analysis is slightly less troublesome here because the question being addressed—stability of the uncertain, closed loop system—is less specific. The analysis requires only that the gain from reference input to plant output is finite and not that it meet a prescribed bound.

While a robust synthesis method for neural controllers would be useful in generating the initial neural control design for a system, adaptation is necessary to specialize the controller to a given plant. In the context of adaptive control, robust performance is forgone in the name of performance on a specific plant. This is convenient in the case of neural control because of the problems discussed above with analyzing the robust performance of recurrent neural network controllers. Robust stability, however, is still desirable. Given that the plant is not completely known, and that even models adapted to the plant online can not be completely accurate, assuring stability with respect to a set of plant models makes sense. The robust stability of the closed control loop can be addressed using the IQC descriptions of the plant and recurrent neural network.

A linear fractional representation (LFR) of the closed loop control system is needed to apply the IQC analysis results. Representations of the control loop containing an RNN with time-invariant weights and an RNN with time-varying weights are both needed. The details of constructing LFRs can be found in [14]. The uncertain, non-linear operator in the resulting model has the structure

$$\Delta(s) = \mathrm{diag}\{\Delta_{\mathrm{lti}}(s), \delta_{m_1}, \delta_{m_2}, \delta_{k_1}, \delta_{k_2}, \delta_{c_1}, \delta_{c_2}, \Phi(\cdot)\},$$

when the RNN weights are static. The operator is augmented with the time varying coefficients for the full, time-varying control loop model,

$$\Delta(s) = \mathrm{diag}\{\Delta_{\mathrm{lti}}(s), \delta_{m_1}, \delta_{m_2}, \delta_{k_1}, \delta_{k_2}, \delta_{c_1}, \delta_{c_2}, \Phi(\cdot), \overline{\delta}_{ij}, \underline{\delta}_{ij}\}.$$

Using IQCs for the different uncertainties and nonlinearities that have been previously described, an LMI problem can be constructed to assess the stability of the closed loop system and compute a bound on its gain. If the resulting LMI is feasible, the control loop is proved stable for all plants in the uncer-

tainty set and additionally, for all controllers satisfying the IQC description of the RNN.

An an example, consider the RNN with $n = 3$, $C = I$, and

$$W = \begin{bmatrix} -1.4992 & 0.5848 & 0.5417 \\ 0.3425 & 0.4623 & 0.4551 \\ 0.7165 & 0.0323 & -0.2045 \end{bmatrix}.$$

The gain across the RNN is bounded from above by $\gamma_{\mathrm{r}} = 0.9751$. The bound was computed using the Popov IQC and taking $T \in \mathcal{M}_{dd}$, the doubly dominant nonlinearity IQC. Since the gain of the uncertain plant model is bounded by $\gamma_{\mathrm{p}} = 1.4755$ and $\gamma_{\mathrm{r}} > 1/\gamma_{\mathrm{p}} \approx 0.6777$, the small gain theorem fails to prove stability. The full IQC analysis, on the other hand provides a gain bound of $\gamma_{\mathrm{cl}} = 1.7938$ proving the stability of the closed loop for all systems in the uncertainty set. A sense of the effect the plant uncertainty has on the analysis of the closed loop can be gained by measuring the gain of the RNN in a loop with just the nominal plant. The gain of this system is bounded above by $\gamma_{\mathrm{n}} = 1.4830$. In this particular case the plant uncertainty has only a mild effect on the estimated loop gain, but this is not always the case.

Bounds on the allowable variation in the RNN parameters can be computed using the full closed loop system model in the analysis. For the weight matrix given above variation bounds are computed as

$$\overline{\Delta} = \begin{bmatrix} 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \end{bmatrix} \quad \text{and} \quad \underline{\Delta} = \begin{bmatrix} 0.3409 & 0.1210 & 0.2290 \\ 0.8891 & 0.4266 & 0.3296 \\ 1.8523 & 0.2090 & 0.4622 \end{bmatrix}$$

using the LMI approximation approach and restricting the bounds to be greater than or equal to 0.1 [10]. The uncertainty in the plant has a large effect on this particular computation. Computing the variation bounds using just the nominal plant model leads to an increase in the sum of the bounds by 1.8220 and for some weights doubles the amount of variation that can be tolerated. Inaccuracies in the uncertain plant model can thus negatively impact the performance of the stable learning algorithm presented in the previous chapter by allowing less variation in the weights than can be safely tolerated by the actual system.

1.4.4 Determining RNN Weight Updates by Reinforcement Learning

The RNN weights are adapted using a reinforcement learning approach of the reference tracking problem following [5]. Since the algorithms are generally applied in discrete time to sampled data, the following presentation uses a discrete time representation of certain parts of the problem. Given a deterministic, dynamical system

$$\dot{x} = f(x, u)$$

where $x \in \mathbb{R}^n$ is the system state and $u \in \mathbb{R}^m$ is the control input, find a control law, or *policy*, $\mu(x) : \mathbb{R}^n \to \mathbb{R}^m$, mapping x to u that minimizes

$$Q^\mu(x(t), u(t)) = \int_t^\infty e^{-\frac{s-t}{\tau}} c(x(s), u(s))\, ds, \quad u(s) = \begin{cases} \mu(x(s)) & s > t, \\ u(t) & s = t. \end{cases} \tag{1.14}$$

The function $c(x, u) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ describes the cost of being in a particular system state and taking a particular control action. For example, in the reference tracking problem, a simple cost function is $c(x(t), u(t)) = \|r(t) - x(t)\|$ where $r(t)$ is the current reference signal. The design of the cost function is one of the most important parts of the problem specification since it determines the behavior of the optimal solution [13]. The parameter, τ, is a discount factor that determines the time horizon over which the algorithm attempts to minimize the cost. For small values of τ the optimal policy will be rather myopic; only the near term effects of actions are considered. As τ increases the optimal control policy considers more of the long term impact of actions. Selecting an appropriate value for τ is another important design decision in the construction of the reinforcement learning problem, but it is not always straightforward [13].

To use a reinforcement learning approach within the context of the proposed stable learning algorithm, an explicit representation of the control policy is needed. Thus, we use an actor-critic approach to generate an explicit control policy that is modeled with an RNN. Two of the basic assumptions of the reinforcement learning approach are that the state of the environment is fully observable and that the dynamics satisfy the Markov property. The Markov property requires that the observed effect of a control action at time t is a function of only the state at the current time and does not have any dependence on the past state trajectory. The two assumptions of state observability and Markov dynamics are closely related. In the control problem under consideration, the state of the plant is not fully observable. Because of this, the observed dynamics do not satisfy the Markov property. One approach to solving this type of problem requires modeling the control problem as a type of partially observable Markov decision process or POMDP. Another approach requires that a representation of the system that satisfies the Markov property be developed. Recurrent neural networks are useful for this purpose because of their ability to model temporal sequences and model hidden dynamics [4]. To simplify the computations of this example application and because the structure of the critic does not affect the stability analysis [1], a standard feedforward neural network with complete state input and 50 hidden units is trained as the critic to model $Q^\mu(x(t), u(t))$ in the following experiments.

The RNN used for the actor has three states represented by three units, thus $W \in \mathbb{R}^{3\times 3}$. Updates to the actor weights are made using stochastic gradient descent on the gradient of the Q-function with respect to the control

inputs,

$$W \leftarrow W - \eta_a \frac{\partial Q(x,u)}{\partial u} \frac{\partial u}{\partial W}$$

The update can be computed by using $-\frac{\partial Q(x,u)}{\partial u}$ as the error function in the RTRL algorithm. The learning rate was varied for some experiments, but as a default $\eta_a = 0.001$. The updates to the actor weights proposed by this algorithm are exactly the updates that must be monitored by the stable learning algorithm to ensure stability.

1.4.5 Results

The following experiments are designed to illustrate the properties of the stable learning algorithm more than to simulate the actual application of these techniques in practice. For instance, in practice, a robust controller would be designed for the plant and the actor-critic model would simply modify the output of the robust controller in some way [1]. The combination of robust controller and actor-critic system allows a certain amount of performance to be guaranteed at the deployment of the system. In the experiments that follow less attention is payed to the quality of the learned control law than to the stability properties of the adaptation.

The actor and critic models were initialized by simulating them for 5000 seconds on the nominal, linear, plant model described in Section 1.4.2. The simulation was done without regard for stability since the actor and critic adaptation was performed on a model and not the real system. Initializing the actor and critic in this way allowed them to be hooked into the real system with some *a priori* knowledge. All of the experiments that follow begin with these initialized actor and critic parameters. Three sets of results are reported for actor-critic learning without stability analysis, with step-wise stability analysis, and with step-wise stability plus stability bias analysis. Before summarizing the results, details of how the stability analysis is applied in this example are presented.

During the training of the actor and critic the stable learning algorithm is used to filter the actor updates. The stability bias is used with two slight modifications. First, to make better use of the available information, updates that are rejected by the original algorithm are instead rescaled such that they satisfy the current constraint set. This allows progress to be made at every step. Second, rather than compute the stability bias at every step, the bias computation is turned on and off based on whether or not the previous steps were accepted without modification. Initially, the stability bias is not used, but after a sequence of three updates in a row have been rescaled due to violation of the constraints the stability bias computation is turned on. The bias computation remains on until a sequence of five steps in a row are accepted without scaling.

The initial actor weights, generated by training on the linear plant, can not be proved stable. To initialize the stable learning algorithm the weights are

scaled down to satisfy the stability constraints. The resulting initial network weights produced a gain of $\gamma_{cl} = 0.600$.

The system was simulated for 1100 seconds and the following results were observed. Unlike the previous example, no instability was observed in the controlled system during the adaptation. Out of the 11,000 updates generated by the actor-critic algorithm, 2541, roughly one in four, required rescaling to satisfy the stability constraints. The stability bias was computed 4200 times, or on almost half of the steps. The stability bounds were recomputed 6540 times. The mean variation allowed per weight over these constraint sets was 0.028. Because of repeated failures to find bounds that satisfied the minimum constraint, the lower bound was decreased repeatedly. The amount of variation allowed in the weights is, on average, very little. This causes the number of constraint computations to rise and increases the cost of the algorithm drastically. The gains computed in the stability bias computations remained relatively small, $\gamma < 5$, during the simulation. This suggests that the small amounts of allowable variation are not due to closeness of the controller to the boundary of the stable weight set, $\mathcal{W}_{ss}^n$. It appears that conservativeness in the analysis of the time-varying RNN is hindering the application of the stable learning algorithm. The algorithm succeeds in keeping the control system stable, but has an excessive cost.

1.4.5.1 Actor-Critic Learning without Stability Analysis Beginning with the actor and critic trained on the nominal, linear plant model, the control system was simulated for 1000 seconds on the actual plant without any constraints on the stability of the system. At a sampling rate of 10Hz, the actor and critic weights were updated 10000 times. In Figure 1.3 a portion of the recorded system behavior is shown. Clearly, the system exhibits instability for reference signals near zero.

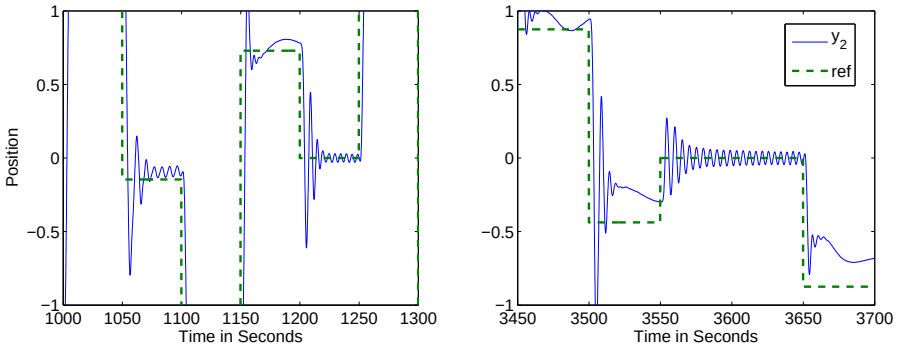


Figure 1.3 An example of unstable behavior exhibited during the training of the actor-critic system.

The closed loop system can not be proved stable by the IQC analysis.

1.4.5.2 Step-wise Stable Actor-Critic Learning Because the conservativeness in the analysis of the time-varying RNN limits the amount of variation that can be tolerated under the stability constraints, it seems worthwhile to consider what benefit might be had from accepting weaker stability guarantees. Rather than constraining the variation in the weights using the full static and dynamic stability analysis, the weights of the actor are simply constrained to remain in $\mathcal{W}_{ss}^n$ at all times; weight updates that push the weights out of $\mathcal{W}_{ss}^n$ are rejected. This guarantees that stopping the adaptation at any point will always result in a stable controller. This type of stability guarantee has been called step-wise stability [13]. The weaker stability algorithm does not protect against the type of dynamic instability due to switching problems. Such problems might occur due to cycles in the weight settings cause by a non-decaying step size in the actor updates. This type of problem was never encountered during simulation, however.

Starting from the initialized actor used in the previous section, this new algorithm was simulated for 5000 seconds on the actual plant. Of the 50,000 updates generated, only 10,364 were accepted. The updates were rejected when the weights approached the boundary between provably stable and possibly unstable weight matrices. The behavior during the last 1000 seconds is shown in Figure 1.4. No instability was seen over the entire 5000 second history.

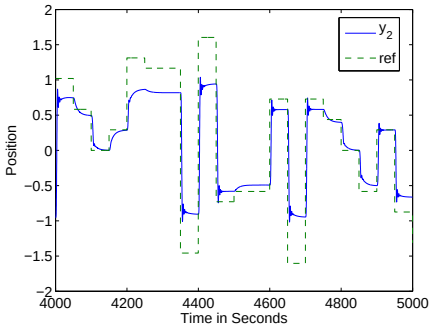


Figure 1.4 Example of the behavior of the step-wise stable adaptation.

1.4.5.3 Actor-Critic Learning with a Stability Bias Only one in four updates to the actor was accepted using the simple step-wise stability algorithm. To test whether the addition of the stability bias term to the weight updates increases the number of accepted upates, the previous experiment is repeated with the stability bias term added to the actor RNN's weight updates. The weight update with stability bias in (1.11) was applied over two 5000 second adaptation trials. The weight update is parameterized by $\hat{\gamma}$ which determines

at what $\mathcal{L}_2$-gain the bias begins to have a major effect on the updates. The first trial used $\hat{\gamma} = 5$ and the second used $\hat{\gamma} = 50$. The weighting parameters, η_2 followed the decay schedule $\eta_2 = \frac{8000-t}{8000}$. Figure 1.5a shows the last 1000 seconds of the episode and the weight trajectories for $\hat{\gamma} = 5$. Figure 1.5b shows the same data, but from the trial with $\hat{\gamma} = 50$. Several observations can be made about the data. Compared to the previous experiment where no stability bias was used, both trials of this experiment perform better at tracking the reference signal. Also, the weights grow larger in these two trials. The lack of stability bias in the previous experiment caused the actor to suffer from a lack of progress due to discarded updates.

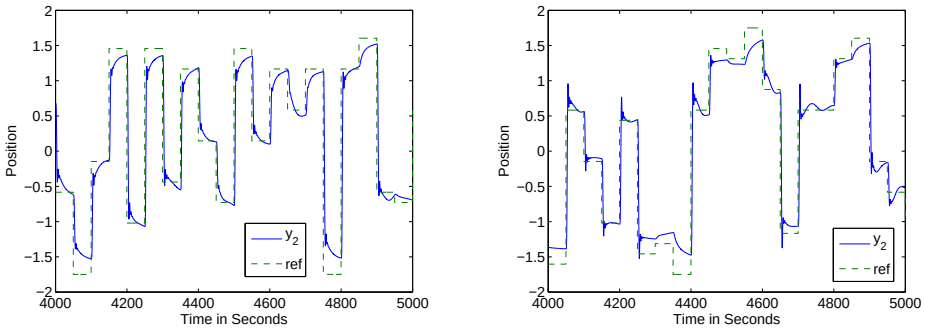


Figure 1.5 Examples of the behavior of the step-wise stable adaptation and the stability bias. a.) $\hat{\gamma} = 5$. b.) $\hat{\gamma} = 50$.

In the case of $\hat{\gamma} = 5$ all 50000 updates were accepted. When $\hat{\gamma} = 50$ nearly 82% of the updates were accepted. The larger value of $\hat{\gamma}$ in this trial allowed the network weights to approach the stability boundary more closely. This increases the likelihood that weight updates will be rejected. The dynamics of the plant output in the low $\hat{\gamma}$ experiment are smoother than those of the high $\hat{\gamma}$ trial. This is due, again, to the soft bound on the network gain that results from the update functions dependence on γ and $\hat{\gamma}$.

The performance of the two actor-critic systems was compared by running the systems for an additional 5000 seconds and comparing the mean squared tracking error and mean squared control output over the 5000 second window. The trial with $\hat{\gamma} = 5$ had a mean squared tracking error of 0.1858 and a mean squared control output of 23.1557. For the $\hat{\gamma} = 50$ case these values were 0.2050 and 26.1874 respectively. Neither of the controllers perform extremely well, but the system with the lower $\hat{\gamma}$ value performs slightly better. This may be due to the fact that the smaller gain constraint results in smoother dynamics which may in turn help to regularize the learning dynamics.

To complete the analysis, the controllers learned after the 5000 step trials of this and the previous section were compared on a fixed set of reference changes

over 1000 steps. No adaptation was performed during these test trials. The mean-squared tracking error and mean-squared control action were recorded. These results are reported in Table 1.1 along with the gain of the controllers. The results reported for the stability bias trial are for $\hat{\gamma} = 5.0$. The results show that the step-wise stability approach with the stability bias has the best overall mean-squared tracking error and a lower mean-squared control output than the unconstrained case. The gain of the controller in the step-wise stability constrained trial where no stability bias was used is very large. This is further evidence of the controller parameters getting stuck near the stability boundary.

	Mean Squared Tracking Error	Mean Squared Control Output	γ
initial actor-critic	0.35	6.31	13.62
after learning:			
without stability analysis	0.11	24.94	∞
with step-wise analysis	0.25	8.92	643.93
with step-wise and gain analysis	0.10	17.13	4.00

Table 1.1 After 5000 seconds of training the learned controllers were tested on a fixed sequence of reference changes. Learning with step-wise stability analysis and stability bias results in the lowest tracking error and the lowest controller gain, or the controller most robust to uncertainties.

1.4.6 Conclusions

The application of a stable learning framework to an uncertain, partially observable, spring-mass-damper system is was illustrated. A reinforcement learning algorithm governs the learning of a control policy in an RNN. Updates to the RNN weights are constrained to guarantee stability of the closed loop system represented with uncertain parameters and bounded nonlinearities. The introduction of a stability bias to direct RNN weight updates away from regions of instability improved the ability of the actor-critic system to learn under the stability constraints. Instability was observed in an actor-critic control system when no stability constraints were explicitly enforced. Application of even the relaxed step-wise stability constraint kept the actor from instability.

REFERENCES

1. C.W. Anderson, P.M. Young, M. Buehner, J.N. Knight, K.A. Bush, and D.C. Hittle. Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. *IEEE Transactions on Neural Networks*, 18(4):993–1002, 2006.

2. A.F. Atiya and A.G. Parlos. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks*, 11(3):697–709, 2000.

3. S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

4. K. Bush. *An Echo State Model of Non-Markovian Reinforcement Learning*. PhD thesis, Colorado State University, 2008.

5. K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12(1):219–245, January 2000.

6. G.E. Dullerud and F. Paganini. *A Course in Robust Control Theory*. Springer, 2000.

7. R.W. Freund and F. Jarre. A sensitivity analysis and a convergence result for a sequential semidefinite programming method. Technical Report Numerical Analysis Manuscript 03-4-08, Bell Laboratories, 2003.

8. T. Hu, B. Huang, and Z. Lin. Absolute stability with a generalized sector condition. *IEEE Transactions on Automatic Control*, 49(4):535–548, April 2004.

9. J.-J. Kim and T. Singh. Desensitized control of vibratory systems with friction: Linear programming approach. *Optimal Control Applications and Methods*, 25:165–180, 2004.

10. James N. Knight. *Stability Analysis of Recurrent Neural Networks with Applications*. PhD thesis, Department of Computer Science, Colorado State University, 2008.

11. James N. Knight and Charles W. Anderson. Stable reinforcement learning with recurrent neural networks. *Journal of Control Theory and Applications*, 9(3):410–420, 2011.

12. R.M. Kretchmar. *A Synthesis of Reinforcement Learning and Robust Control Theory*. PhD thesis, Computer Science Department, Colorado State University, 2000.

13. G.G. Lendaris and J.C. Neidhoefer. Guidance in the use of adaptive critics for control. In J. Si, A.G. Barto, W.B. Powell, and D. Wunsch, editors, *Handbook of Learning and Approximate Dynamic Programming*, pages 97–124. Wiley-IEEE Press, 2004.

14. U. Mackenroth. *Robust Control Systems: Theory and Case Studies*. Springer-Verlag, 2004.

15. C. Makkar, W.E. Dixon, W.G. Sawyer, and G. Hu. A new continuously differentiable friction model for control systems design. *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 600–605, 24-28 July 2005.

16. B.A. Pealmutter. Gradient calculations for dynamic recurrent neural networks: A survey. *IEEE Transactions on Neural Networks*, 6(5):1212–1228, 1995.

17. U.D. Schiller. *Analysis and Comparison of Algorithms for Training Recurrent Nerual Networks*. PhD thesis, University of Bielefeld, 2003.

18. J. Steil. *Input-Output Stability of Recurrent Neural Networks*. PhD thesis, Der Technischen Fakultät der Universität Bielefeld, 1999.

19. G.-Z. Sun, H.-H. Chen, and Y.C. Lee. Green's function method for fast online learning algorithm of recurrent neural networks. In J.E. Moody, S.J. Hanson, and R.P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pages 333–340. MIT Press, 1991.