J Control Theory Appl 2011 9 (3) 410–420
DOI 10.1007/s11768-011-0177-1

Stable reinforcement learning with recurrent neural networks

James Nate KNIGHT, Charles ANDERSON

Department of Computer Science, Colorado State University, Fort Collins, CO 80523, U.S.A.

Abstract: In this paper, we present a technique for ensuring the stability of a large class of adaptively controlled systems. We combine IQC models of both the controlled system and the controller with a method of filtering control parameter updates to ensure stable behavior of the controlled system under adaptation of the controller. We present a specific application to a system that uses recurrent neural networks adapted via reinforcement learning techniques. The work presented extends earlier works on stable reinforcement learning with neural networks. Specifically, we apply an improved IQC analysis for RNNs with time-varying weights and evaluate the approach on more complex control system.

Keywords: Stability analysis; Integral quadratic constraint; Recurrent neural network; Reinforcement learning; Linear matrix inequality

1 Introduction

Modern robust control theory is based on explicit mathematical models of uncertainty [1]. 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. For example, some types of uncertainty can be described by integral quadratic constraints (IQCs) [2] and lead to representations of uncertainty as convex sets of operators. Linear systems are a particularly tractable type of model, and the design of feedback controllers for linear systems is a well understood problem. Thus, linear models of physical systems are particularly attractive. Most physical systems, however, exhibit some nonlinear dynamics and linear models are generally insufficient for accurately describing them. Unmodeled nonlinear dynamics can often be modeled within the same framework as uncertainty. Because robust controllers must be insensitive to inaccuracies and uncertainties in system models, performance is often sacrificed in the actual system to which the controller is applied. Additional loss in performance is introduced by restricting controllers to being linear and of low order. These properties are generally desirable because low order, linear controllers can be easily analyzed and understood. Performance can often be improved by the use of nonlinear and adaptive control techniques, but guaranteeing stability and performance is more difficult with these types of controllers.

In this paper, we present a technique for ensuring the stability of a large class of adaptively controlled systems. We combine IQC models of both the controlled system and the controller with a method of filtering control parameter updates to ensure stable behavior of the controlled system under adaptation of the controller. We present a specific application to a system that uses recurrent neural networks adapted via reinforcement learning techniques. The present work extends earlier works in [3, 4]. Specifically, we apply an improved IQC analysis for RNNs with time-varying weights developed in [5] and evaluate the approach on more complex control systems.

In the first part of the paper we present a general algorithm for ensuring the stability of nonlinear systems controlled with time-varying RNNs. We illustrate the approach on a simple supervised learning problem that adapts an RNNs behavior to known system dynamics. In the second part of the paper we illustrate how the approach can be applied to ensure the stability of a nonlinear plant controlled by an RNN adapted using reinforcement learning techniques.

2 Background

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 mapping system states to control outputs. For these types of tasks, it will be sufficient to restrict attention to continuous time RNN formulations of the form

$$\begin{cases} \dot{x} = -Cx + W\Phi(x) + u, \\ y = x. \end{cases} \tag{1}$$

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

$$\Phi(x) = [\phi(x_1)\ \ \phi(x_2)\ \cdots\ \phi(x_n)]^{\mathrm{T}}.$$

The function $\phi(x)$ is a continuous one-dimensional map, and generally a sigmoid-like function, such as $\tanh x$. More

Received 28 July 2010; revised 22 March 2011.
This work was supported by the National Natural Science Foundation (No.0245291).

general models allow the selection of certain states as outputs or an additional mapping to be applied at the output layer.

We provide here a basic outline of the stability analysis that is of interest for the present problem. More detailed information can be found in [5–8]. We shall define an m-dimensional signal, u, to be a mapping from the time interval $[0, \infty)$ to $\mathbb{R}^m$. The space of signals that is piecewise continuous and square integrable in addition to the norm

$$\|u\|_{\mathcal{L}_2^m} = \sqrt{\int_0^\infty u^{\mathrm{T}}(t)u(t)\,\mathrm{d}t}, \;\; u(t) \in \mathbb{R}^m,$$

forms a normed vector space denoted $\mathcal{L}_2^m$ or $\mathcal{L}_2^m[0, \infty)$. To deal with unstable systems properly this space of signals is expanded to include those that are integrable when restricted to a finite interval of time. The extended space, $\mathcal{L}_{\mathrm{pe}}^m$, is defined as $\mathcal{L}_{\mathrm{pe}}^m = \{u | u_\tau \in \mathcal{L}_{\mathrm{p}}^m, \forall \tau \in [0, \infty)\}$, where u_τ represents the restriction of the signal u to the time interval $[0, \tau)$. Given this definition, the operator H is taken to be a mapping from $\mathcal{L}_{\mathrm{pe}}^m$ to $\mathcal{L}_{\mathrm{pe}}^q$.

A nonlinear system such as (1) can be viewed as an operator, H, mapping an input signal, u, in some signal space to an output signal, y, into another signal space. It is the properties of this operator that determine the stability of the system. The following definition is thus of interest.

Definition 1 (Finite gain $\mathcal{L}_{\mathrm{p}}$ stability [6]) An operator $H : \mathcal{L}_{\mathrm{pe}}^m \to \mathcal{L}_{\mathrm{pe}}^q$ is finite gain $\mathcal{L}_{\mathrm{p}}$ stable if there exists a nonnegative constant γ such that $\|(H(u))_\tau\|_{\mathcal{L}_{\mathrm{p}}^q} \leqslant \gamma \|u_\tau\|_{\mathcal{L}_{\mathrm{p}}^m}$, for all u_τ in $\mathcal{L}_{\mathrm{p}}^m$ and τ in $[0, \infty)$. The constant γ is called the gain of the system.

In [2], the method of integral quadratic constraint analysis (IQC) was introduced. The IQC formalism allows complex systems to be analyzed by considering descriptions of the system's various components. Also, the IQC paradigm allows general purpose software to be constructed for piecing together IQC models for complex systems and automating the derivation of the necessary stability analysis problems [9].

An integral quadratic constraint describes the relationship between two $\mathcal{L}_2$ signals, v and w in the following way

$$\int_{-\infty}^{\infty} \begin{bmatrix} \hat{v}(\mathrm{j}\omega) \\ \hat{w}(\mathrm{j}\omega) \end{bmatrix}^* \Pi(\mathrm{j}\omega) \begin{bmatrix} \hat{v}(\mathrm{j}\omega) \\ \hat{w}(\mathrm{j}\omega) \end{bmatrix} \mathrm{d}\omega \geqslant 0, \qquad (2)$$

where $\hat{v}$ and $\hat{w}$ are the Fourier transforms of the two signals [2]. Pairs of signals satisfying the constraint are said to satisfy the IQC given by Π. These types of constraints are of interest because the existence of an IQC for the system operator H allows the formulation of stability proofs using the IQC stability theorem proved in [2]. The IQC stability theorem gives conditions under which the system H satisfies the finite gain stability definition given above. The stability proofs can be formulated as semidefinite programming problems that can be solved efficiently by numerical software.

3 Stable learning with RNNs

Adapting RNNs in an off-line setting, whether as controllers or as models, generally requires no stability analysis. In a control setting, off-line adaptation of recurrent neural networks uses interaction with model systems that may not accurately reflect reality. Online adaptation, however, allows RNN components to adapt to the actual properties of the controlled plant and track the changes of that system over time. Using RNNs in online adaptive systems, however, requires that their stability be guaranteed during adaptation. In this section an algorithm is presented that ensures the stability of an adaptive RNN. The algorithm is applicable to many systems that can be modeled and analyzed using IQCs. For instance, the algorithm is easily extended to ensure the stability of an RNN in a control loop with an uncertain nonlinear plant model. We give such an application in Section 3. Here, however, the exposition is focused on simply maintaining stability of an RNN under adaptation from some arbitrary algorithm. This removes some distractions and helps maintain the focus on the basic properties of the algorithm.

In [5], an IQC-based stability analysis is developed for RNNs with fixed and time-varying weights. Both types of analysis are necessary for enabling the algorithm proposed in this chapter. Because the stability analysis computations are expensive and generally have worse asymptotic complexity than the computation of weight updates, they dominate the run time of the proposed algorithm. Steps are taken to minimize the number of stability analysis computations thereby reducing the cost of ensuring the stability of adaptive RNNs.

Before proceeding to an example problem some notation is introduced. Define the set $\mathcal{W}_{\mathrm{ss}}^n$ as all $n \times n$ weight matrices, W, which result in stable RNN dynamics. The analysis provided in [5] produces an inner approximation of this set whose quality is dependent on the choice of IQCs. The set $\mathcal{W}_{\mathrm{ss}}^n$ is not generally known explicitly and to limit the notation somewhat, the symbol $\mathcal{W}_{\mathrm{ss}}^n$ is also used to represent approximations of it.

3.1 An example adaptive system

To illustrate the behavior of the proposed algorithm, a simple RNN learning problem with stability constraints is formulated in this section. The problem will be to train a recurrent neural network to model a given dynamic system subject to constraints on the stability of the RNN. The problem simulates the type of situation in which the proposed stable learning algorithm will be applied and helps to illustrate many of the underlying issues that must be considered.

The dynamic system that the RNN is trained to model is itself another RNN. The problem can be constructed to have an optimal solution with weights in the set of stable weights or outside of it. Also the solution can be made to lie near to or far from the boundary of $\mathcal{W}_{\mathrm{ss}}^n$. The problem has the following form:

$$\min_{W \in \mathcal{W}_{\mathrm{ss}}^2} \sum_{i=1}^{N} (\bar{x}(t_i) - x(t_i))^2, \;\; 0 < t_1 < t_2 < \ldots < t_n \in \mathbb{R},$$

$$\bar{x}(t) = \int_0^t (-C\bar{x}(\tau) + \bar{W}\phi(\bar{x}(\tau)) + u(\tau))\mathrm{d}\tau, \;\; \bar{x}(0) = 0,$$

$$x(t) = \int_0^t (-Cx(\tau) + W\phi(x(\tau)) + u(\tau))\mathrm{d}\tau,$$

$$x(0) = 0, \;\; u(t) = \begin{bmatrix} \sin t \\ 2\cos\dfrac{t}{2} \end{bmatrix}.$$

The input to the adaptive system will be the instantaneous error, $E(t) = \hat{x}(t) - x(t)$, generated by continuously running versions of the optimal network and the adapted network. The times, t_i specified in the problem definition are used only for evaluating the performance of the adaptive network. The task is to find an RNN with a 2×2 weight matrix that reproduces the behavior of the optimal RNN on a given input sequence using only the available error signal. The optimal RNN has a weight matrix $\bar{W}$ that is chosen to illustrate different properties of the stable learning algorithm. For illustrative purposes the diagonal elements of W will be fixed to the corresponding values in $\bar{W}$. The resulting problem then has a two-dimensional parameter space and allows the dynamics to be easily visualized. The matrix C in 1 is taken to be the identity and $\phi(x)$ is taken to be the $\tanh(\cdot)$ function.

Many algorithms exist for adapting the weights of recurrent neural networks. A survey of gradient-based approaches can be found in [10]. To solve the example learning problem described above, 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 to 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 introduces temporal dependencies between the parameters and the error function. Computation of the error gradient requires explicitly accounting for these dependencies, and RTRL provides a way of doing this.

Recall the RNN equations

$$\begin{aligned} \dot{x} &= F(x, u; C, W) = -Cx + W\Phi(x) + u, \\ y &= x, \end{aligned}$$

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

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

where $\gamma(t_0) = 0$. The variable γ is a rank three tensor with elements γ_{ij}^l corresponding to the sensitivity of x_l to changes in the weight W_{ij}. RTRL requires simulation of the γ 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

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

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 basic algorithm is practical for small networks and is sufficient for illustrating the properties of the proposed stable learning algorithm.

For the example problem, $\bar{W}$ is taken to be the matrix

$$\bar{W} = \begin{bmatrix} 0.0756 & 1.0663 \\ 1.2691 & -0.4792 \end{bmatrix}.$$

In Fig. 1, a sample of weight trajectories generated by RTRL are displayed along with an approximation to $\mathcal{W}_{\mathrm{ss}}^n$ computed using a brute force IQC analysis of many weight combinations. The optimal solution, given by $\bar{W}$, is near the boundary of $\mathcal{W}_{\mathrm{ss}}^n$. Two weight trajectories are shown for each of two starting points. The different trajectories were generated using different learning rates. For higher learning rates the trajectories tend to be more erratic and can leave $\mathcal{W}_{\mathrm{ss}}^n$. This is to be expected because RTRL has no knowledge of the stability constraints. Higher learning rates also allow more rapid optimization of the objective function. The purpose of the stable learning algorithm proposed in this chapter is to constrain the trajectories to the set $\mathcal{W}_{\mathrm{ss}}^n$ and to ensure that the weight matrix transitions do not lead to instability. Forcing the trajectory to remain within $\mathcal{W}_{\mathrm{ss}}^n$ can allow larger learning rates to be used safely and thus provide an improvement in the rate at which performance is improved.

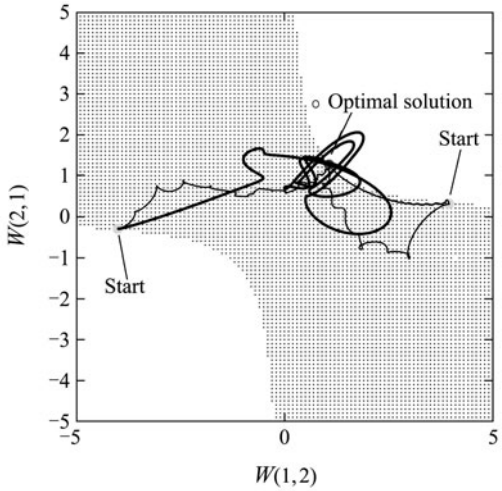

Fig. 1 Examples of weight trajectories induced by training an RNN to reproduce the output of dynamic system using RTRL. The space of RNNs with fixed weights that can be proved stable is shown in blue. Only the off diagonal weights of the 2×2 RNN weight matrix are modified. The diagonal weights are set to the optimal values. Some trajectories leave the provably stable region.

3.2 Maintaining stability of an adaptive RNN

Given a stable initial RNN it is possible to consider adapting the weights of the network to optimize an objective function. As discussed in the introduction, it is necessary to guarantee the stability of the system as it changes through time. Consider a simple approach to this problem. Compute a set of bounds on the variation of the RNN weights within which stability is guaranteed, and then filter out any weight updates that put the weights 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 cannot 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 [11] 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 cannot be expected that the optimal weight matrix for a problem, $\bar{W}$, will be reachable from W 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. The algorithm proposed in this section falls somewhere between these two extremes allowing better optimization of the objective than the first approach with less computational expense than the second.

An algorithm, called the stability constrained learning algorithm, is listed in Table 1.

Table 1 The stability constrained learning algorithm.

$j \leftarrow 0, k \leftarrow 0$
Update $\leftarrow$ **false**
Initialize $W(0)$
Compute constraint set $\mathcal{C}_0\,(W(0))$
Repeat
  $k \leftarrow k+1$
  Compute ΔW
  If $W(k-1)+\Delta W \in \mathcal{C}_j$
  $W(k) \leftarrow W(k-1)+\Delta W$
  Update $\leftarrow$ **true**
  Else
    If Update
    $j \leftarrow j+1$
    Compute constraint set $\mathcal{C}_j(W(k-1))$
    Update $\leftarrow$ **false**
    if $W(k-1)+\Delta W \in \mathcal{C}_j$
    $W(k) \leftarrow W(k-1)+\Delta W$
    update $\leftarrow$ **true**

The algorithm assumes that changes to the weight matrix are proposed by an external agent, such as RTRL, at discrete time steps indexed by the variable k. The algorithm ensures the stability of an adaptive RNN by filtering weight updates that cannot be guaranteed stable. A constraint set, $\mathcal{C}_j(W)$ is a set of bounds, $\{\underline{\Delta}, \bar{\Delta}\}$, on the variation in the RNN weight matrix centered on the fixed matrix 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 hypercube centered on W with each element of $W(k)$ constrained by

$$W_{ij} - \underline{\Delta}_{ij} \leqslant W_{ij}(k) \leqslant W_{ij} + \bar{\Delta}_{ij}.$$

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 proposed procedure makes better use of the available weight update suggestions from the adaptation algorithm. Fig. 2 illustrates the behavior of the proposed algorithm.

In Fig. 3 the result of applying the stability constrained learning algorithm to a weight trajectory generated by RTRL for the example problem described in Section 3.1 is shown.

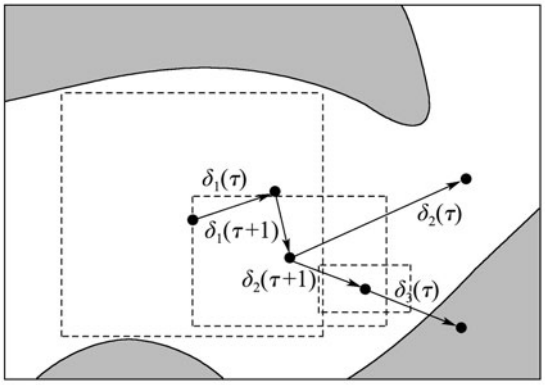

Fig. 2 Given an initial stable point, the update $\delta_1(\tau)$ is proposed and accepted since it satisfies the stability constraints. The next update is also accepted. Update $\delta_2(\tau)$ violates the stability constraints, so the constraints are updated. The proposed update violates the new constraints as well, and so is rejected. Update $\delta_2(\tau+1)$ is accepted since it satisfies the new constraints. Update $\delta_3(\tau)$ violates the constraints, and causes the constraints to be updated again. Because it still violates the new constraints the update is rejected.

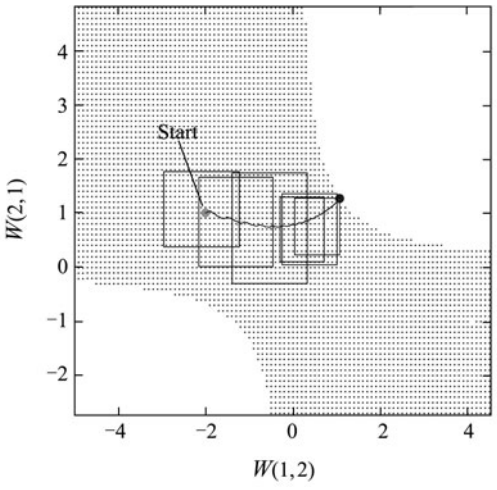

Fig. 3 Shown here are the results of applying the stability constrained learning algorithm to a sample trajectory. This is an ideal case where very few stability constraint sets are generated.

The constraint sets were generated using the method described in [5]. The example is ideal in that very few sets of stability constraints were computed and the optimal weight matrix was in $\mathcal{W}_{\rm ss}^n$.

Fig. 1 shows an example of starting from an initial point and adapting the weights using the RTRL 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 be-

havior, only that stability cannot be guaranteed along the trajectory induced by the unconstrained RTRL algorithm. Such excursions from the stability region could have damaging effects if the RNN is in a control loop with an actual physical system. Applying the proposed algorithm forces the trajectory to remain in $\mathcal{W}_{\rm ss}^n$.

Fixme, describe alternate, static stability approach The stable learning algorithm presented in the previous chapter is applied to the problem, but suffers from conservativeness in the stability analysis of the closed feedback loop. Several modifications to the basic algorithm are considered. An alternative algorithm that ensures that the feedback loop is stable for each static setting of the RNN weights visited during adaptation of the controller is developed. The algorithm gives up the guarantee of dynamic stability given by bounding the variation in the RNN weights. This compromise reduces the computational cost of the algorithm and allows the RNN to successfully adapt a stable control strategy for the system.

4 Robust adaptive neural control

In this section, we illustrate the application of the algorithms described in the previous section to the control of a nonlinear, uncertain, multiple spring-mass-damper system. The system is a simple instance of a larger class of models representing physical systems such as flexible manipulators and active suspension systems. The problem is to construct a neural controller for the system with guaranteed stability during operation and adaptation.

The next section introduces the control system under consideration and the associated models used for stability analysis and simulation. Also, an IQC analysis of the uncertain plant model is developed. Section 4.2 describes the control configuration used and develops an IQC analysis of the closed loop, control system. Also, details of the reinforcement learning algorithm used to train the adaptive controller are given. An experimental evaluation of the stable adaptive control system is reported at the end of the section. The example illustrates the ability of the stable learning algorithm to improve control performance by removing instability from the control loop.

4.1 Two degrees of freedom spring mass damper

A diagram of the simulated plant is shown in Fig. 4. 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 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.

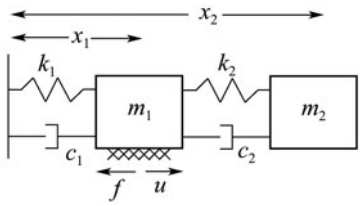

Fig. 4 A multiple spring-mass-damper system.

The plant dynamics are governed by the ordinary differential equations

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

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 $h \geqslant 0$. 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 Fig. 5 for $\dot{x}_1 \in [-5, 5]$. The friction equation and parameters are taken from [12] and model both stiction and Coulomb type friction.

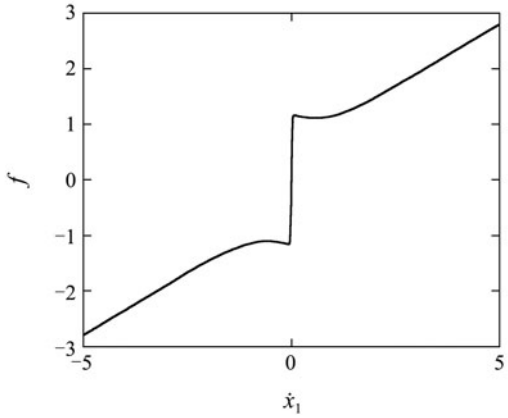

Fig. 5 A continuously differentiable friction model.

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 10 Hz, 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 $k_2 = 4.8$.

4.1.1 An uncertain linear plant model

For the purposes of controller design and stability analysis 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 $m_i \in [1.96, 2.04]$, $c_i \in [0.9, 1.1]$, 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 (3) yields the linear model

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

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 is

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

where $\Delta_{\rm lti}$ is an unknown, linear, time invariant system with $\|\Delta_{\rm lti}\|_\infty < 1$. The unknown 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 $p = (p_n + \delta_{\rm p} m_{\rm p})$ where p_n is the nominal value of the parameter, $\delta_{\rm p}$ is an unknown constant satisfying $|\delta_{\rm p}| \leqslant 1$ and $m_{\rm p}$ is a scaling factor on the uncertainty [13]. To represent the uncertain parameter, $k_1 \in [4.5, 5.5]$, for example, take $p_n = 5$ and $m_{\rm p} = 0.5$. The uncertain parameters in the model are assumed to be constant. The representation of time-varying parameters is essentially the same, but $\delta_{\rm p}$ 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: $\Delta_{\rm lti}$ and the $\delta_{\rm p}$'s for the uncertain parameters. The details of this procedure can be found in standard robust control references such as [1, 13]. 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 [13]. The IQC model developed in the next section is used to show the same result.

4.1.2 IQC analysis of the plant

Recall that the uncertain parameters in the linear model are represented by $p = (p_n + \delta_{\rm p} m_{\rm p})$ where $\delta_{\rm p}$ is unknown but satisfies $|\delta_{\rm p}| \leqslant 1$. If $w, v \in \mathcal{L}_2$, the relation $w = \delta_{\rm p} v$ satisfies IQCs, with $\Pi(s)$ of the form

$$\Pi(s) = \begin{bmatrix}x(s) & z(s)\\ z^*(s) & -x(s)\end{bmatrix}, \quad x(s) \geqslant 0,$$

where $x(s)$ and $z(s)$ are bounded, measurable functions [2, 9]. This type of IQC is known as a dynamic IQC. To work with the IQC, the functions $x(s)$ and $z(s)$ need computationally tractable representations. Generally, the functions are described as a linear combination of a finite set of simple basis functions. Increasing the number or complexity of the basis functions enlarges the set of IQCs but also increases the size of the resulting LMI constraints. For the analysis done in this section and the next, the function $x(s)$ is represented as the combination of a constant function and a simple scaled transfer function

$$x(s) = x_0 + x_1 \frac{1}{s+1}.$$

The function $z(s)$ has the representation

$$z(s) = z_1 \frac{1}{s^2-1} = -\frac{1}{2} z_1 \Big(\Big(\frac{1}{s+1} \Big)^* + \frac{1}{s+1} \Big).$$

The representation of the resulting IQCs as LMIs is described in detail in [5]. Essentially, the representation requires the extension of the plant state with states representing $\dfrac{1}{s+1} v$ and $\dfrac{1}{s+1} w$.

Unmodeled LTI dynamics, of the kind used in the uncertain plant model developed above, can also be described with IQCs. If $\Delta_{\rm lti}$ is an LTI operator with norm less than one, then the relation $w(s) = \Delta(s) v(s)$ satisfies all IQCs of the form

$$\Pi(s) = \begin{bmatrix}x(s) & 0\\ 0 & -x(s)\end{bmatrix}, \quad x(s) \geqslant 0$$

with $x(s)$ a bounded, measurable function of the form used in the previous IQC [2].

A stability analysis of the uncertain plant model can be constructed using the IQC stability theorem. A bound on the $\mathcal{L}_2$-gain of the uncertain plant model found 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.

4.2 Robust adaptive control of the multiple Spring-mass-damper

The closed-loop control system under investigation is depicted in Fig. 6. 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.

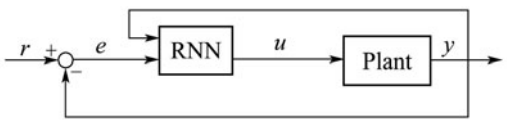

Fig. 6 The closed-loop control system.

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 (3) 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.

4.2.1 Recurrent neural network control structure

The basic RNN equations must be adapted to fit the desired control structure. The simple RNN model considered in the previous section 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

$$\dot{x_{\mathrm{r}}} = -Cx_{\mathrm{r}} + W\Phi(x_{\mathrm{r}}) + W^{\mathrm{i}}[e \;\; y]^{\mathrm{T}}, \;\; u = W^{\mathrm{o}}x_{\mathrm{r}}. \quad (4)$$

Different configurations of W^{i} and W^{o} affect the behavior of the network. A common configuration, see for instance [14], feeds each input into a different node and reads the output from another node. This leads to

$$W^{\mathrm{i}} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathrm{T}}, \;\; W^{\mathrm{o}} = [0 \;\; 0 \;\; 1 \;\; 0 \;\; \cdots \;\; 0].$$

The RNN stability analysis developed in [5] covers this modified RNN as well as the model used in the previous section.

4.2.2 IQC analysis of the closed control loop

A linear fractional representation 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 [13]. The uncertain, nonlinear 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,

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

Using IQCs for the different uncertainties and nonlinearities, 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 has a feasible solution, the control loop is proved stable for all plants in the uncertainty set and additionally, for all controllers satisfying the IQC description of the RNN.

As 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 can be bounded from above by $\gamma_{\mathrm{r}} = 0.9751$ using the IQC analysis proposed in [5]. Additionally, an IQC analysis of the full control loop 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 approach outlined in [5]. For the weight matrix given above variation bounds are computed as

$$\bar{\Delta} = \begin{bmatrix} 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \\ 0.1000 & 0.1000 & 0.1000 \end{bmatrix},$$

$$\underline{\Delta} = \begin{bmatrix} 0.3409 & 0.1210 & 0.2290 \\ 0.8891 & 0.4266 & 0.3296 \\ 1.8523 & 0.2090 & 0.4622 \end{bmatrix}.$$

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 section by allowing less variation in the weights than can be safely tolerated by the actual system.

4.2.3 Reinforcement learning for adaptive control

In the previous section, the weights of an RNN were adapted using a supervised approach that trained the RNN to reproduce a given temporal sequence. For adaptive control, an alternative approach must be taken because the desired output of the RNN is not explicitly known. In the results that follow the RNN weights are adapted using a reinforcement learning approach that we briefly described below. Reinforcement learning is an unsupervised learning approach characterized by its focus on learning through interaction. When applied to control problems, reinforcement learning can be viewed as a class of direct, adaptive, optimal control algorithms [15, 16].

A reinforcement learning formulation of the reference tracking problem considered in this chapter can be formulated following [17]. Because 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

$$\begin{cases} V^{\mu}(x(t)) = \int_t^{\infty} \mathrm{e}^{-\frac{s-t}{\tau}} c(x(s), u(s)) \mathrm{d}s, \\ u(t) = \mu(x(t)). \end{cases} \quad (5)$$

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 prob-

lem, 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 [18]. 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 [18].

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 [19]. In this approach a recurrent neural network, the critic, is used to model the value function, $V^{\mu}(x(t))$. The internal dynamics of the network are used to construct, implicitly, a model of the system satisfying the Markov property. As discussed in [4], the dynamics and stability properties of the critic do not directly effect the stability of the closed control loop because the actor is represented with a separate RNN.

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

4.3 Experimental evaluation

In this section the stable learning algorithm developed in Section 3 is applied to the control of the multiple spring-mass-damper system using the actor-critic method discussed in the previous section. The 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. This was done in [4] for example. 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 paid 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 4.1.1. The simulation was done without regard for stability because the actor and critic adaptation was performed on a model and not on the real system. Initializing the actor and critic in this way allowed them to be plugged into the real system with some a priori knowledge. All of the experiments that follow begin with these initialized actor and critic parameters.

4.3.1 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 10 Hz, the actor and critic weights were updated 10000 times. In Fig. 7, a portion of the recorded system behavior is shown. Clearly, the system exhibits instability for reference signals near zero.

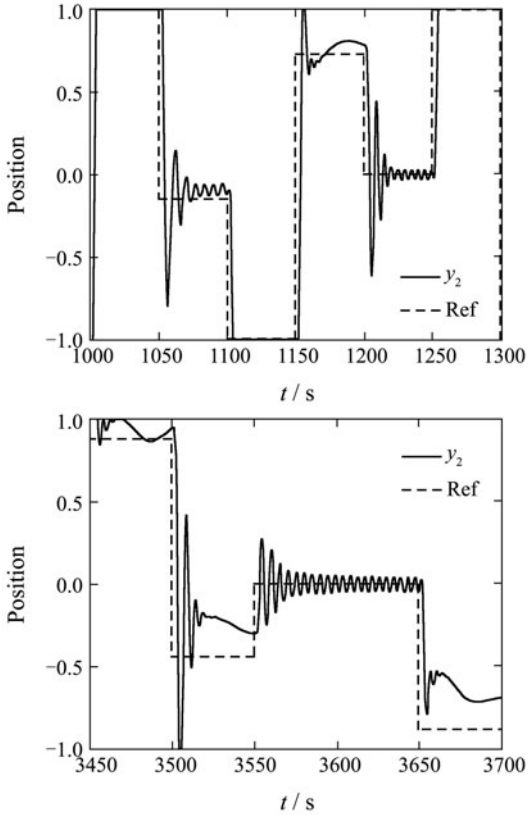

Fig. 7 An example of unstable behavior exhibited during the training of the actor-critic system.

In Fig. 8, the results of simulating the actor with the weights from the 3600 second mark and a reference input of $r = 0.2$ are shown. The actor weight matrix has the values $W = \begin{bmatrix} 1.5808 & -0.2169 & -0.5501 \\ 4.1411 & 0.1407 & 0.5892 \\ 2.9354 & 0.8355 & -0.1111 \end{bmatrix}$. The closed-loop system cannot be proved stable by the IQC analysis developed in Section 4.2.2. This type of instability must be avoided during the training of the actor and critic on the actual physical plant. The algorithm from Section 3 is applied in the next section to prevent this type of behavior.

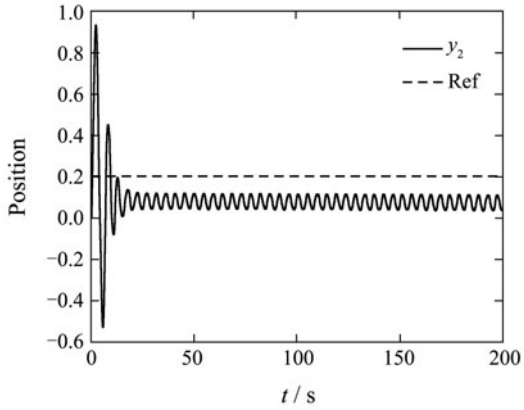

Fig. 8 An example of an unstable RNN controller visited by the actor-critic learning algorithm during training. The controller is unstable for small reference inputs.

4.3.2 Stable actor-critic learning

To avoid the instability seen in the previous example, the stable learning algorithm from Section 3 is used to filter the actor updates.

The initial actor weights, generated by training on the linear plant, cannot 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 $\gamma_{\mathrm{cl}} = 0.600$. The initial variation bounds were

$$\bar{\Delta} = \begin{bmatrix} 0.1000 & 0.1000 & 0.3313 \\ 0.1013 & 0.1000 & 0.2957 \\ 0.1000 & 0.1183 & 0.1095 \end{bmatrix},$$

$$\underline{\Delta} = \begin{bmatrix} 0.3474 & 0.3970 & 0.1141 \\ 0.1775 & 0.3067 & 0.1497 \\ 0.2674 & 0.1949 & 0.3359 \end{bmatrix}.$$

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, 2,541, roughly one in four, were rejected. The stability bounds were recomputed 6540 times. The mean variation allowed per weight over these constraint sets was 0.028. The amount of variation in network weights that can be tolerated within the stability analysis is, on average, very little. This causes the number of constraint computations to rise and increases the cost of the algorithm drastically. Unfortunately, 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. In the next section, the stability constraints are loosened somewhat and a modified version of the algorithm is applied.

4.3.3 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 exist from accepting weaker stability guarantees. Instead of than constraining the variation in the weights using the dynamic weight analysis described earlier, the weights of the actor are simply constrained to remain in $\mathcal{W}_{\mathrm{ss}}^n$ at all times using an IQC analysis of the RNN with weights fixed at the current value. Weight updates that push the weights out of $\mathcal{W}_{\mathrm{ss}}^n$ 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 [18]. The weaker stability algorithm does not protect against instability due to switching problems. Such problems might occur because of cycles in the weight settings caused by a nondecaying step size in the actor updates. This type of problem, however, was never encountered during simulation suggesting that in some cases step-wise stability is a useful property to guarantee.

Starting from the scaled actor used in the previous section, this new algorithm was simulated for 5,000 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 Fig. 9.

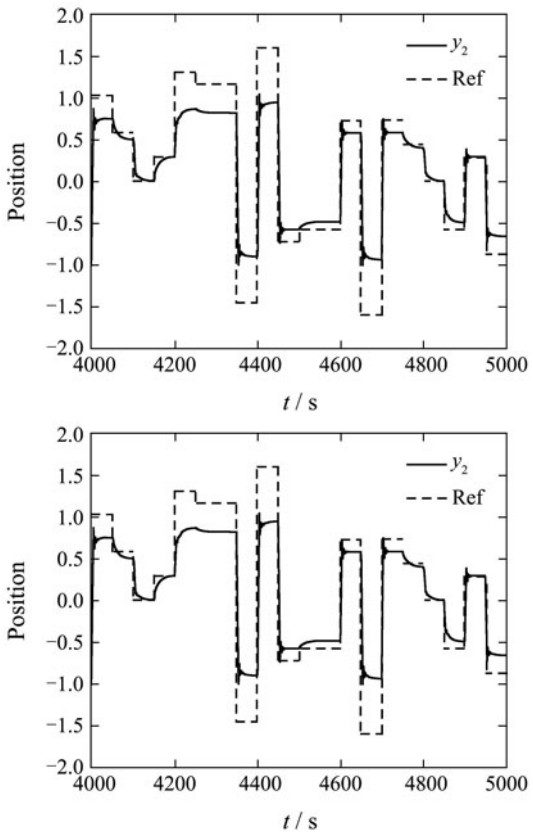

Fig. 9 Example of the behavior and weight trajectories of the step-wise stable adaptation.

No instability was seen over the entire 5000 second history. The weight trajectories over the entire adaptation period are also shown.

To complete the analysis, the stability constrained and unconstrained controllers learned after the 5000-step learning trials 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 (MSTE) and mean-squared control action (MSC) were recorded. These results are reported in Table 2 along with the gain of the controllers. Three cases were compared: initial case, no stability analysis (NS), and step-wise stability (SS). The results show that the step-wise stability approach gives better mean-squared error performance and lower overall control output than the unconstrained controller. On the other hand, the best bound on the controller gain computed using the IQC analysis is very large. This information coupled with the fact that only about one-fourth of the proposed control parameter updates were accepted suggests that the weight trajectory is stuck near the boundary of the set of stable weight matrices. When this occurs it is difficult for the learning algorithm to make progress.

Table 2 After 5000 seconds of training the learned controllers were tested on a fixed sequence of reference changes.

	MSTE	MSC	γ
Initial	0.3493	6.3065	13.623
NS	0.1084	24.9356	∞
SS	0.2478	8.9198	643.933

5 Conclusions

The stable learning algorithm described in Section 3 proved to be too restrictive and computationally expensive for application to the multiple spring mass damper system. By relaxing the stability constraints to consider a weaker step-wise stability, the computational cost was considerably reduced and better performance was achieved. Instability was observed in an actor-critic control system when no stability constraints were explicitly enforced, but application of even the relaxed step-wise stability constraint kept the actor from driving the system into instability.

The experiment also showed that both the full and step-wise constrained learning algorithms became stuck near the boundary of the set of stable RNN weight matrices. Further analysis of this problem has led to the development of a 'stability bias' approach that drastically improves the performance of the approach. This technique is described in [5] and will be detailed in a future publication.

The experimental results presented in this paper illustrate some of the practical difficulties involved in ensuring the stability of adaptive nonlinear control systems. The application of IQC analysis and the stable learning algorithm described in this paper, however, did enable stable, adaptive, nonlinear control to be achieved. The experiments also illustrated the practical need for such approaches when applying recurrent neural networks to control problems, as even in the relatively benign example presented unconstrained adaptation of the RNN weights led to instability in the controlled system. Future work in this area will be focused on improved performance in terms of reduced computational complexity and reduced conservativeness in the IQC analysis of RNNs.

Acknowledgements

The authors would like to acknowledge Michal Kocvara for allowing the use of the PENBMI optimization software in this work.

References

[1] G. E. Dullerud, F. Paganini. *A Course in Robust Control Theory*. New York: Springer-Verlag, 2000.

[2] A. Megretski, A. Rantzer. System analysis via integral quadratic constraints. *IEEE Transactions on Automatic Control*, 1997, 42(6): 819 – 830.

[3] R. M. Kretchmar. *A Synthesis of Reinforcement Learning and Robust Control Theory*. Ph.D. thesis. Fort Collins, CO: Colorado State University, 2000.

[4] C. W. Anderson, P. M. Young, M. Buehner, et al. Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. *IEEE Transactions on Neural Networks*, 2006, 18(4): 993 – 1002.

[5] J. N. Knight. *Stability Analysis of Recurrent Neural Networks with Applications*. Ph.D. thesis. Fort Collins, CO: Colorado State University, 2008.

[6] H. Khalil. *Nonlinear Systems*. New York: Prentice Hall, 2002.

[7] C. A. Desoer, M. Vidyasagar. *Feedback Systems: Input-output Properties*. New York: Academic Press, 1975.

[8] U. Jönsson. *Lecture Notes on Integral Quadratic Constraints*. Stockholm, Sweden: Department of Mathematics, Royal Institute of Technology, 2000.

[9] U. Jönsson, C. Y. Kao, A. Megretski, et al. *A Guide to IQCβ: A Matlab Toolbox for Robust Stability Analysis and Performance Analysis*. IST World, 2004.

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

[11] J. Steil. *Input-output Stability of Recurrent Neural Networks*. Ph.D. thesis. Bielefeld, Germany: Der Technischen Fakultät der Universität Bielefeld, 1999.

[12] C. Makkar, W. E. Dixon, W. G. Sawyer, et al. A new continuously differentiable friction model for control systems design. *Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, New York: IEEE, 2005: 600 – 605.

[13] U. Mackenroth. *Robust Control Systems: Theory and Case Studies*. New York: Springer-Verlag, 2004.

[14] U. D. Schiller. *Analysis and Comparison of Algorithms for Training Recurrent Nerual Networks*. Ph.D. thesis. Bielefeld, Germany: University of Bielefeld, 2003.

[15] R. S. Sutton, A. G. Barto, R. J. Williams. Reinforcement learning is direct adaptive optimal control. *IEEE Control Systems Magazine*, 1992, 12(2): 19 – 22.

[16] J. Si, A. Barto, W. Powell, et al. *Handbook of Learning and Approximate Dynamic Programming*. New York: John Wiley & Sons, 2004.

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

[18] G. G. Lendaris, J. C. Neidhoefer. Guidance in the use of adaptive critics for control. J. Si, A. G. Barto, W. B. Powell, et al., eds. *Handbook of Learning and Approximate Dynamic Programming*. New York: Wiley-IEEE Press, 2004: 97 – 124.

[19] K. Bush. *An Echo State Model of Non-Markovian Reinforcement Learning*. Ph.D. thesis. Fort Collins, CO: Colorado State University, 2008.

[20] S. Becker, S. Thrun, K. Obermayer, eds. *Advances in Neural Information Processing Systems 15*. Cambridge: MIT Press, 2003.

James Nate KNIGHT received his B.S degree in Computer Science and Mathematics from Oklahoma State University in 2001, and his M.S. degree from Colorado State University in 2003. He completed his dissertation on the stability analysis of recurrent neural networks with application to reinforcement learning and control in 2008 and received his Ph.D. from Colorado State University. His current research interested include stable adaptive control, machine learning, and optimization. E-mail: james.nate.knight@gmail.com.

Charles ANDERSON received his B.S. degree in Computer Science from the University of Nebraska-Lincoln, in 1978 and his M.S. and Ph.D. degrees in Computer Science from the University of Massachusetts, Amherst, in 1982 and 1986, respectively. From 1986 through 1990 he worked at GTE Laboratories. He has been a professor since 1991 at the Department of Computer Science, Colorado State University. His research interests are in reinforcement learning, pattern recognition, and machine learning. E-mail: anderson@cs.colostate.edu.