Identification of a PUMA-560 Two-Link Robot Using A Stable Neural Network

Chris M. Jubien, Nikitas J. Dimopoulos

Department of Electrical and Computer Engineering,
University of Victoria,
PO Box 3055, Victoria, BC, V8W 3P6 CANADA

Abstract --A training procedure for a class of neural networks that are asymptotically stable is presented. The training procedure is a gradient method which adapts the interconnection weights as well as the relaxation constants and the slopes of the activation functions used so as the error between the expected and obtained responses is minimized. A method for assuring that stability is maintained throughout the training procedure is also given. Such a network was used to identify a simulated nonlinear system and a PUMA-560 two-link robot.

I. Introduction

This paper is a summary of some recent work done in the area of identification of nonlinear systems using neural networks. The main purpose of this work is to provide a way of establishing models of complex nonlinear systems that can be used in controllers. Neural networks are selected as a potentially effective way of modeling these systems, since a trained neural network is fast and easy to implement, properties that are desirable in real controllers.

One problem with using a system as complex as a nonlinear neural network in a control or identification setting is that they are often too complex to analyze fully; in particular, their stability can not be assured. When dealing with real systems, stability is the single most important property of a controller or model. Fortunately, a class of neural networks exists which is known to be asymptotically stable. This class of neural networks is used here, and the work done on identification pertains to this class of dynamic neural networks.

II. Background

It has been shown [1] that asymptotic stability is ensured for neural networks which are described by the differential equation

$$\dot{O} = -TO + Wf(O) + b \tag{1}$$

In (1), there are N neurons divided into k classes, and

$$\begin{aligned} O &= \begin{bmatrix} O_1 & O_2 & \cdots & O_k \end{bmatrix} \\ &= \begin{bmatrix} o_1 & o_2 & \cdots & o_N \end{bmatrix} \end{aligned} \tag{2}$$

is the state of the neural network,

$$W = \begin{bmatrix} W_{11} & W_{12} & \cdots & W_{1k} \\ \vdots & & & \\ W_{k1} & W_{k2} & \cdots & W_{kk} \end{bmatrix} \tag{3}$$

is the network connectivity matrix, $T = \text{diag}\ (\tau_i)$ is the diagonal matrix of neural relaxation constants, b is the input to the neural network, and $f(O)$ belongs to the class of so-called neuromime functions, which are essentially positive and monotonically non-decreasing. The condition on W that guarantees asymptotic behaviour is that it must contain all of its positive entries on one side of the main diagonal [1]. This gives an easy way to check whether a neural network is stable. For instance, the neural network shown in Figure 1 is stable provided that the connection weights in submatrices W_{23} and W_{34} are non-positive (i.e., inhibitory). This result is extremely useful in the area of identification and control. The most important feature of a controller or model is that it must be stable. By starting with a model as defined by (1), stability is ensured.

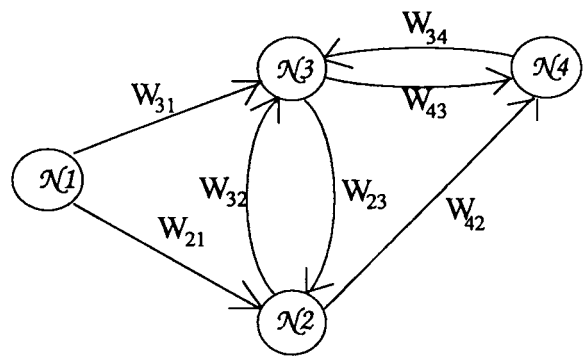

Fig. 1. Sample Neural Network

III. Parameter Adjustment In Stable Neural Networks

This section discusses a method for adjusting the weights and other parameters of neural networks that are stable in the sense described in Section 2. The general approach that is used here is to define some a criterion and then adjust the parameters in a direction that will decrease this cost. In this sense the technique is similar to linear recursive adaptive methods [4] and to classical back propagation [5]. However, since the stable neural networks described in section 2. have certain restrictions on the polarity of the connection of classes, a straightforward gradient adjustment is not possible. A solution for this is also presented here.

A. *Gradient of Cost Function*

The general equation for calculating the behaviour of the class of neural networks of interest here is

$$\dot{O} = -TO + Wf(O) + b \tag{4}$$

using the same notation introduced in section 2.. One possible criterion for measuring the performance is the quadratic cost function

$$J(e) = 1/2(O-O_d)^T A(O-O_d)$$
$$= 1/2e^T Ae \quad (5)$$

where O_d is the desired state of the neural network. Matrix A is used to eliminate from the cost any neurons whose state is not crucial. A is a diagonal matrix with ones corresponding to output neurons and zero's elsewhere. As in other recursive adaptive methods [2,4], parameters θ in the neural network are adjusted along the negative gradient of this cost, i.e.,

$$\frac{d\theta}{dt} = -\eta \frac{\partial J}{\partial \theta} \quad (6)$$

The chain rule for differentiation is used to allow for the calculation of this gradient for parameters associated with neuron j:

$$\frac{\partial J}{\partial \theta} = \frac{\partial J}{\partial o_j}\frac{\partial o_j}{\partial \theta}$$
$$= \gamma_j \frac{\partial o_j}{\partial \theta} \quad (7)$$

The notation γ_j is used to denote the derivative of the cost with respect to the activation of neuron j. If neuron j is an output neuron, this derivative is simply given by

$$\gamma_j = o_j - o_{d_j} \quad (8)$$

In a manner analogous to traditional back propagation of the error [5], this gradient may be calculated for units that are not output neurons by using the values of the gradient in all the neurons k that have neuron j as inputs:

$$\gamma_j = \sum_k \gamma_k \frac{\partial o_k}{\partial o_j}$$
$$= \sum_k \gamma_k \Delta_{kj} \quad (9)$$

Here, the notation Δ_{kj} has been introduced to represent the partial derivative $\partial o_k / \partial o_j$. To calculate Δ_{kj}, it is necessary to use the differential equation which defines the behaviour of the neural network. Rewriting (4) specifically for neuron k, and using the operator D to represent differentiation results in

$$(\tau_k + D)\, o_k = \sum_j w_{kj} f(o_j) + b_k \quad (10)$$

Differentiating (10) with respect to o_j results in

$$\dot{\Delta}_{kj} = (-\tau_k)\Delta_{kj} + w_{kj} f'(o_j). \quad (11)$$

Note that unlike classical back propagation [5], the equation governing the propagation of error from one class to the next is a differential equation.

All the derivatives required in (7) to adjust a parameter θ have now been obtained, except for the derivative $\partial o_j / \partial \theta$. The next section discusses the case when θ is a connecting weight, and the section following that discusses the case of parameters of the differential equation, such as the relaxation constant τ or parameters of the activation function $f(\,)$.

B. Input Weight Adjustment

Let θ represent a connecting weight w_{ji} which connects neuron i (input) to neuron j. Use the notation $\xi_{ji} = \partial o_j / \partial w_{ji}$. Differentiating (10) with respect to w_{ji}, the differential equation for ξ_{ji} is obtained:

$$\dot{\xi}_{ji} = -\tau_j \xi_{ji} + f(o_i) \quad (12)$$

Using this equation and the results of the previous section, equation (7) may now be written as

$$\frac{dw_{ji}}{dt} = -\eta \gamma_j \xi_{ji} \quad (13)$$

with γ_j calculated using (8) or (9) as appropriate.

C. Adjustment of Structural Parameters

The same analysis that was used to determine how to adjust the connection weights can also be used to obtain a formula for adjusting any of the other variables that parameterize the neural network. For instance, a formula for adjusting a parameter of the activation function $f(\,)$ or the neural relaxation constants τ_j (analogous to time constants for a linear system) in a way that will reduce the cost function can be obtained. Consider an activation function of the form

$$f(o) = \begin{cases} \dfrac{1}{1+e^{-\sigma_0}} & \text{if } o \geq 0 \\ 0 & \text{if } o < 0 \end{cases} \quad (14)$$

The parameter σ controls how much input causes the neuron to saturate; the larger it is, the harsher the nonlinearity. Since there is no way of knowing *a priori* what an optimal or even appropriate value for this variable is, it makes sense to adapt it while training the connection weights. Since the value γ_j will already be available during training of the weights, the only further calculation required is the derivative $v_j = \partial o_j / \partial \sigma_j$. Using (10), it is seen that

$$(\tau_j + D)\, v_j = \sum_i w_{ji} \frac{\partial f(o_i)}{\partial \sigma_j}$$
$$= \begin{cases} \sum_i w_{ji} o_i (0.25 - f(o_i)^2) & \text{if } o_i \geq 0 \\ 0 & \text{if } o_i < 0 \end{cases} \quad (15)$$

Using this formula to calculate the value for v_j, the nonlinearity parameter may be adjusted using the relation

$$\frac{d\sigma_j}{dt} = -\eta \gamma_j v_j \quad (16)$$

Using the same technique, it is possible to calculate $\beta_j = \partial o_j / \partial \tau_j$ to adjust the relaxation constants. This is useful since it is not known beforehand how fast a system the neural

network will be trying to identify. The above technique yields the differential equation for β_j :

$$\ddot{\beta}_j + 2\tau_j\dot{\beta}_j + \beta_j = -\sum_i w_{ji} f(o_i) \tag{17}$$

and an update formula for τ_j of

$$\frac{d\tau_j}{dt} = -\eta\gamma_j\beta_j \tag{18}$$

It is important here that the relaxation constant τ_j not be adjusted at too great a rate, or else (17) is not valid since τ_j is treated as a constant.

D. Weight Clamping

Section 2 describes a class of neural networks that are asymptotically stable. This condition is guaranteed provided that the connectivity matrix W has all of its positive entries on one side of the diagonal [1]. However, (13) gives a formula for adjusting the connection weights that may violate this condition. To combat this, it is necessary to check the polarity of certain crucial weights after each weight adjustment. For instance, as discussed in section 2, if the weights labeled W_{23} in Figure 1 are guaranteed to be non-positive, then the neural network will be stable. Thus after any weight in W_{23} is adjusted using (13), the weight should be checked to ensure that it is not positive. If it is, then it should be clamped at 0. This technique ensures that inhibitory weights stay inhibitory throughout the training procedure.

IV. Identification

The term identification is used in this section to refer to the process of devloping a model of an unknown system by observing its input/output behaviour.[2,4].

This section uses the results of the previous section to identify some unknown systems. A suitable neural network architecture is proposed and some motivation for this configuration is given. A simple computer simulated system is identified, and then the neural network is used to identify a two-link robot arm. The results of the neural network are then compared to the results obtained using a classical identification technique (recursive least squares) [4].

A. *Ientification Architecture*

This section discusses a motivation for selecting a neural network architecture suitable for system identification. Consider the simple nonlinear system described by the relation

$$\dot{y} = u - \frac{y}{1+4y^2} \tag{19}$$

If y remains relatively constant near some value y_{ss}, then this system can be approximated by a first order linear system that has a pole at $(1+4y_{ss}^2)^{-1}$. If y varies from this value significantly, then the 'pole' can be thought of as roving in some sense. Although this is not an exact description of the behaviour of the system, it does illustrate one of the more common types of nonlinearity which is encountered in real systems such as valve flows and airplanes cruising at various velocities.

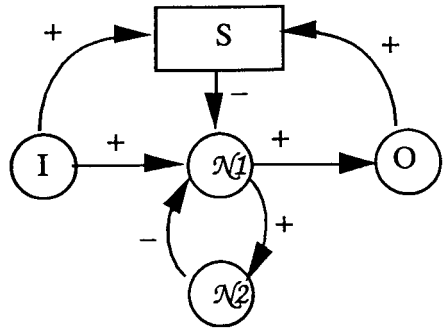

Fig. 2. An Architecture for System Identification

To take advantage of this type of nonlinearity, the architecture of Figure 2 is proposed for general system identification. In this figure, the labels I and O refer to the input and output of the system, and $\mathcal{N}1$ and $\mathcal{N}2$ refer to two classes of neural networks. The block marked S is a special connection of classes called the 'scheduler class'. The idea of this class is that it controls or schedules which neurons will be active and when, thereby emulating the movement of the 'pole' for large variations of the state variable or input. The motivation for this architecture is most clearly understood by examining the case when the unknown system and the activation function $f(\ )$ of the neurons are linear. In this case, equation (4) shows that the output O follows the weighted sum of the states in $\mathcal{N}1$. Ignoring for the moment the effects of the scheduler class, and making the time constant of O small, then the state equation of this system takes on a familiar form:

$$\begin{bmatrix} O_{N_1} \\ O_{N_2} \end{bmatrix} = \begin{bmatrix} -\tau_1 & W_{12} \\ W_{21} & -\tau_2 \end{bmatrix} \begin{bmatrix} O_{N_1} \\ O_{N_2} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} I$$

$$O = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} O_{N_1} \\ O_{N_2} \end{bmatrix} \tag{20}$$

This form can implement a general linear system of any order by the proper selection of the connection matrices and with a sufficient number of neurons. Neurons in the scheduler class have a peaked response as shown in Figure 4. (This class of neurons is not a single class as governed by (4). However, the response shown in Figure 4 was generated using a combination of 4 standard classes in a configuration shown in Figure 3. For clarity, the scheduler neurons are discussed as a single class).

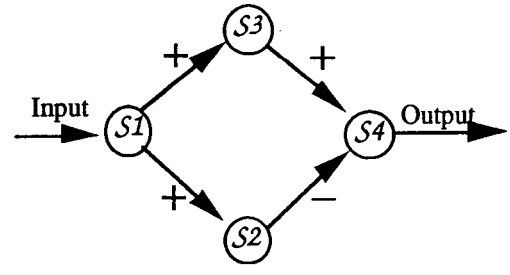

Fig. 3. Architecture for Scheduler class

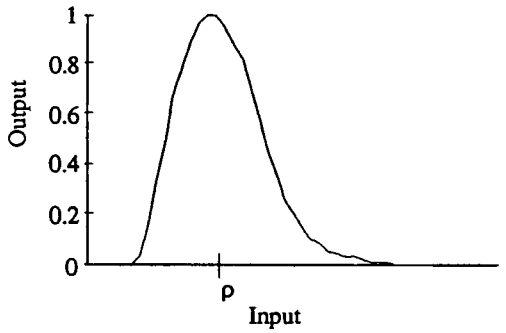

Fig. 4. Response of Scheduler Neurons

Each neuron in the class has a peak ρ that occurs at a different value. Figure 2 shows that the scheduler class receives input from I and O. Thus, depending on the value of the input and output, different neurons in $\mathcal{N}1$ and $\mathcal{N}2$ will be active. This allows the neural network to take advantage of the type of nonlinearity discussed above. The example described by (19) is well suited to this kind of architecture since neurons with different relaxation constants may be activated depending on the level of the output, thus allowing the overall system to act as desired.

B. Simulation Results

An architecture similar to that shown in Figure 2 was used to identify the model defined by (19). For this example, $\mathcal{N}2$ was not included, since it was known in this case that there was no need for second-order effects due to the nature of the differential equation. $\mathcal{N}1$ contained five neurons, and there were five neurons in the scheduler class. The training procedure consisted of applying an input that was a combination of a step of random height added to white noise. The training procedure was continued for over a million iterations. After training was complete, this simple neural network was able to follow the response to the input shown in Figure 5. The response of the neural network and the actual model are shown in Figure 6.

C. Identification of a Two-Link Robot Arm

A two-link robot arm is known to have its dynamic response governed by the differential equation

$$H(q)\ddot{q} + h(q,\dot{q})\dot{q} + F\dot{q} + g(q) = \Phi v \qquad (21)$$

where the state $q$contains the angle θ_1 that the first link makes with the vertical and the angle θ_2 that is formed between the two links; $H(q)$ is the 2x2 inertial matrix; $h(q,\dot{q})$ models the Coriolis and centripetal forces; F is the friction matrix; $g(q)$ represents the gravitational torque; and Φ is the voltage-to-torque conversion matrix [3]. All of these variables rely on many machine-specific factors, such as dimensions, weights, inertia, and joint friction. Due to the complicated manner in which these factors are combined to produce the matrices, classical identification techniques make it difficult to obtain an accurate model.

One way to mitigate these difficulties is to obtain direct measurements of as many variables as possible. This was done for a PUMA-560 robot. Lengths, masses, and inertias were obtained through direct measurement [6]. The variables which could not be easily directly measured were the matrices F and Φ. This represented four unknown scalars in total. Classical RLS parameter estimation was used to identify these four variables, and the final response to the input vector shown in Figure 7 is shown in Figure 8. Although Figure 8 shows that there was some prediction error, it was found [3] that this model was accurate enough to allow for an extremely accurate controller design when this model was used for closed loop control. The fact that the model deviates from the actual response underlies the difficulty in identifying complex systems using traditional model based methods.

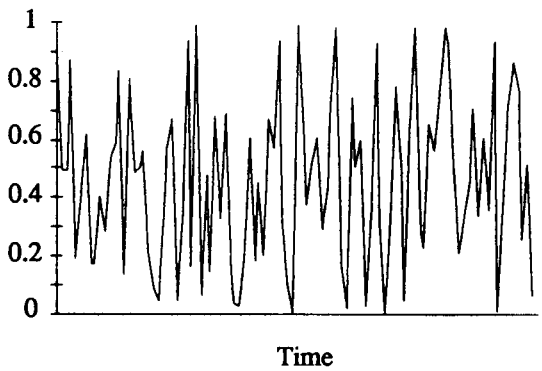

Fig. 5. Input to Model and Neural Network

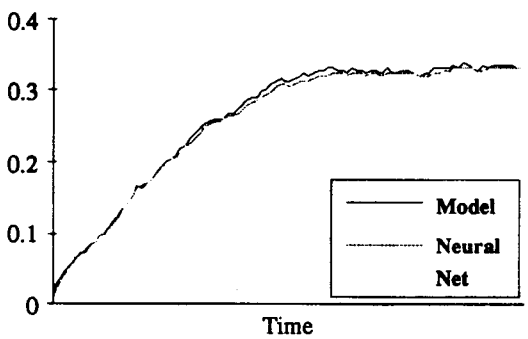

Fig. 6. Model and Neural Net Responses

The identical neural network that was used to identify the simple simulation of (19) was then trained to identify the dynamic response for θ_1, the angle that the first link makes with the vertical. Both $v_1(t)$ and $v_2(t)$ were used as inputs to the system. The neural network had an architecture similar to that shown in Figure 2, except that $\mathcal{N}2$ was not included. Class $\mathcal{N}1$ contained 5 neurons as did the scheduler class. Convergence of the response was rather slow; several million training iterations were required. However, due to the small size of the neural network, good results were obtained after two days of training on a Sparcstation. After training was completed, the neural network followed the model closely. The response is shown in Figure 9. This reveals that the neural network tracks this particular input better than the classical model obtained using least squares parameter estimation and empirical measurement. The advantage of the neural network is that due to its small size and simple calculation procedure, it is ideally suited to use in real hardware controllers driven by chips such as the

HC-11. Furthermore, the laborious process of physical parameter measurement is avoided.

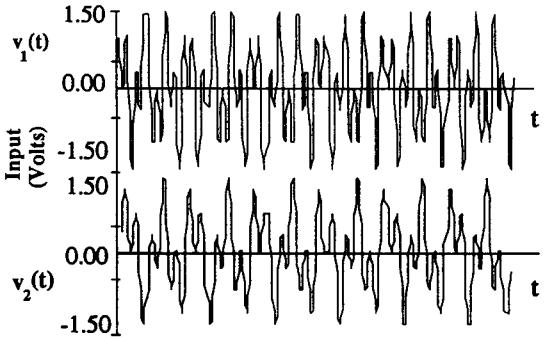

Fig. 7. Control Voltage for Link 1 and Link 2

V. Conclusions

In this paper, a class of neural networks that was known to be stable was investigated. A training procedure for the neural networks was obtained here. In addition to traditional weight adjustment training, other structural parameters of the neural network, such as relaxation constants and nonlinearity thresholds, were adapted. Formulae for adjusting these parameters along the gradient of a cost function were derived here.

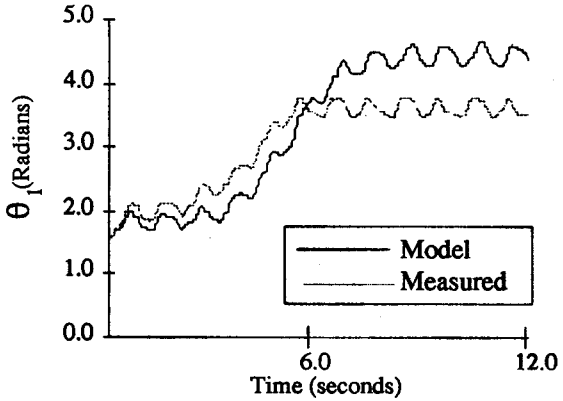

Fig. 8. Classical Model (Equation 21) and Measured Response to Input

Also, a method for dealing with stability restrictions on the connection polarity of certain classes within the neural network was discussed. This training procedure was used to train very small neural networks to identify a simulated system and an actual two-link robot. The training time for the neural network was seen to be rather long. The advantages of this type of identification model were seen to be that it allowed accurate identification and produced an easy to calculate, guaranteed stable model of very complex systems

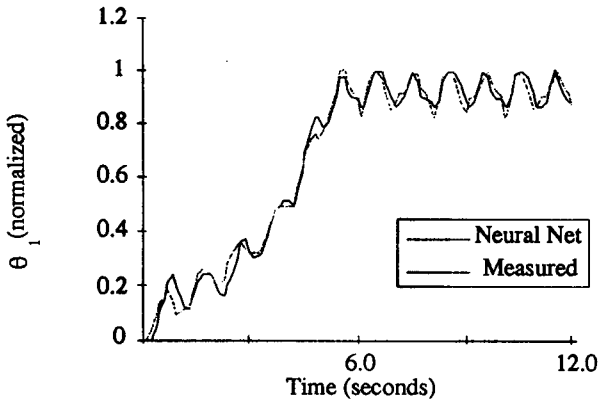

Fig. 9. Measured and Neural Net Response to Input shown in Figure 7

Acknowledgement

This work was supported by the the Institute of Robotics and Intelligent Systems, a Canadian Network of Centers of Excellence.

References

[1] N. Dimopoulos, "A Study of the Asymptotic Behaviour of Neural Networks," *IEEE Transactions on Circuits and Systems*, Vol. 36, No.5, pp. 687-694, May 1989.

[2] K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, pp.4-27, March 1990.

[3] M. Erlic, M.A.Sc. Thesis, University of Victoria, Victoria B.C., 1990.

[4] K.J. Astrom and B. Wittenmark, *Adaptive Control*, Addison-Wesley Publishing Company, 1989.

[5] B. Widrow and M. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *IEEE Proceedings, Special Issue on Neural Networks*, Vol. 78, No. 9, Sept. 1990.

[6] B. Armstrong, O.Khatib, and J. Burdick, "The Explicit Dynamic and Inertial Parameters of the PUMA-560 Arm", *IEEE Int'l Conference on Robotics and Automation*, 1986, pp. 510-518