Robotic System Sensitivity to Neural Network Learning Rate: Theory, Simulation, and Experiments

Christopher M. Clark
James K. Mills

Laboratory for Nonlinear Systems Control
Department of Mechanical and Industrial Engineering
University of Toronto
5 King's College Road
Toronto, Ontario Canada M5S 3G8

Abstract

Selection of neural network learning rates to obtain satisfactory performance from neural network controllers is a challenging problem. To assist in the selection of learning rates, this paper investigates robotic system sensitivity to neural network (NN) learning rate. The work reported here consists of experimental and simulation results. A neural network controller module, developed for the purpose of experimental evaluation of neural network controller performance of a CRS Robotics Corporation A460 robot, allows testing of NN controllers using real-time iterative learning. The A460 is equipped with a joint position proportional, integral, and derivative (PID) controller. The neural network module supplies a signal to compensate for remaining errors in the PID-controlled system. A robot simulation, which models this PID-controlled A460 robot and NN controller, was also developed to allow the calculation of sensitivity to the NN learning rate. This paper describes the implementation of three NN architectures: the error back-propagation (EBP) NN, mixture of experts (ME) NN, and manipulator operations using value encoding (MOVE) NN. The sensitivity of joint trajectory error of three NN controllers to learning rate was investigated using both simulation and experimentation. Similar results were obtained from the robot experiments and the dynamic simulation. These results of state sensitivity to NN learning rate confirm that the MOVE NN is least sensitive to learning rate, implying that selection of suitable learning rates for this NN architecture for the system considered is accomplished more readily than other NN architectures.

KEY WORDS—neural network control, robotic systems, sensitivity

1. Introduction

In the mid 1980s, interest in neural networks (NNs) grew when it was shown that nonlinear NN architectures could be trained to produce desired outputs. Rumelhart, Hinton, and Williams (1986) introduced the multilayer perceptron model using the error back propagation algorithm for training the weights of the model. Evidence was found that indicated NNs were capable of learning complex functions, which led to their use in applications including pattern recognition, function approximation, data fitting, and control of dynamic systems. Nonlinear dynamic systems, including robots executing tasks repetitively, were shown to benefit from the use of NN controllers, i.e., Chen, Mills, and Smith (1996). The results from these and other experiments reported in the literature demonstrated a reduction in robot trajectory tracking error through the use of a feed-forward error back-propagation (EBP) NN controller.

Within the literature, which addresses the application of NNs for robot control systems, considerable work has been reported that employs feed-forward EBP NNs. However, several improvements to the EBP algorithm, as well as many new NN architectures and learning algorithms have been developed specifically for robot control applications. Three such NNs, discussed in this paper, are the EBP NN, mixture of experts (ME) NN (Jacobs and Jordan 1993), and manipulator operations using value encoding (MOVE) NN (Graham and D'Eleuterio 1990).

When these NN control algorithms are implemented, selection of a suitable NN learning rate must be made. This objective is typically achieved through the use of trial and error processes. A learning rate, when selected too large, may lead to overall system instability, while a learning rate that is too low will lead to a system that does not respond quickly to parameter changes that are to be learned, for example.

Investigation of the sensitivity of trajectory tracking error of the robot system under NN control to learning rate will provide insight into the selection of learning rates for robotic systems and the effect of nonoptimal learning rates on system performance. In our work, we have investigated the sensitivity of three different NN controller architectures to learning rate and have determined the sensitivity of these architectures to variations in learning rate. This knowledge will provide users of NN controllers with a body of quantitative information, which may simplify the selection of learning rates for these NN architectures. For example, attempts to optimize the learning rate, to achieve better closed-loop performance, will be more efficient with knowledge of the sensitivity of performance to learning rate variation. Conversely, with certain NN architectures, the closed-loop system performance is more sensitive to learning rate, hence more care must be taken in selection of a suitable NN learning rate. While qualitative in nature, this work represents a first step toward a rationale for selection of NN learning rate.

In this work, we first derive the dynamic models of a robotic manipulator controlled with a proportional, integral, and derivative (PID) joint position controller, augmented with an NN controller with real-time learning. The sensitivity equations that relate state sensitivity to NN controller learning rate are derived. Using a full dynamic simulation of our experimental robotic system including real-time NN learning, the sensitivity of the system state to NN learning rate is found. Comparison of these sensitivity results is then made with experimental results obtained using an NN robot control test-bed with real-time NN learning.

By comparing the results obtained using the above procedure for three different NN architectures, i.e., EBP, ME, and MOVE, it is found that the MOVE NN exhibits markedly lower joint trajectory error sensitivity to NN learning rate. This implies that a successful choice of learning rate for the MOVE NN architecture can more readily be achieved than with either the EBP or ME NN architectures. The MOVE NN will operate successfully over a wider range of learning rates than either the EBP or ME NN controllers. Second, when selection of NN learning rate is undertaken, the fact that the MOVE NN controller is least sensitive to learning rate, when compared to either EBP or ME controllers, makes selection of a suitable learning rate a simpler task. This allows for easier implementation of the MOVE NN compared to EBP and ME NNs.

In Section 2, three NN controllers implemented are described. Section 3 describes the NN-robot control test-bed module and the experimental robot. Section 4 introduces the NN learning rate sensitivity equations and describes how they are implemented in a dynamic simulation. Experimental results are presented in Section 5, and finally conclusions are given in Section 6.

2. Neural Networks

In the following, we briefly outline relevant details of three NN controllers that were implemented experimentally on the robot system and in simulation. A more detailed description of the algorithms can be found in the corresponding references.

2.1. Error Back-Propagation

Feed-forward networks trained with EBP have been the focus of considerable work reported in the literature. EBP has exhibited slower learning times when compared with other NN learning algorithms (Graham and D'Eleuterio 1991; Jacobs and Jordan 1993). Due to its wide use in research and engineering applications, EBP was selected as the baseline for comparison with the other types of NNs being investigated.

The EBP algorithm is composed of two main steps that are repeated iteratively. The steps are (1) a forward pass to produce values for the NN outputs and (2) a backward pass to adjust the weights so as to achieve desired outputs from the network. The weight adjustment rule is given in eq. (1) below. At the n^t time step, the change in the weight w_{ig} that connects neuron I to neuron j is given by

$$\Delta w_{ij}(n) = -\lambda \frac{\partial J(n)}{\partial w_{ij}}, \tag{1}$$

where

$J(n) \equiv$ Cost Function, $J(n) \in \Re^1$;
$w_{ij} \equiv$ Connection weight, $w_{ij} \in \Re^1$;
$\Delta w_{ij} \equiv$ Change in connection weight, $\Delta w_{ij} \in \Re^1$; and
$\lambda \equiv$ Learning rate, $\lambda \in \Re^1$.

A full description and mathematical details of the algorithm can be found in Haykin (1994).

2.2. Mixture of Experts

The motivation to use ME NNs for the NN module came from the work reported by Jacobs and Jordan (1993). In this work, robot simulations illustrated ME NNs with both a faster learning time and lower joint errors than EBP feed-forward networks. The principle of "divide and conquer" is used as the basis of this NN architecture. The intention is to divide a complex problem into several smaller problems that are more easily solved (see Rueckl, Cave, and Kosslyn 1989). In a ME NN, a different NN is used for different regions of the output. This is accomplished automatically by the NN learning algorithm.

The ME NN is composed of several feed-forward NNs connected in parallel with the output from each summed to produce the output from the system. Each of the feed-forward NNs is an "expert" and will learn its portion of the task accordingly. A gating network is used to determine the proportion of each expert's output used in the system's output. The gating network itself is also a feed-forward NN whose inputs are the same as the experts.

Illustrated in Figure 1 is the network topology for the ME algorithm. Experts 1 through 3 are feed-forward NNs. Learning is similar to that of EBP in that there are forward and backward passes through the NN. However, additional weights in the gating network must also be trained. Eqs. (2a) and (2b) are the learning rules used to adjust the weights in the expert networks and the gate network, respectively. A full description of the algorithm along with mathematical details can be found in Jacobs et al. (1991).

$$\Delta w_{ijk}(n) = \lambda h_i(n) e_i(n) x_j \tag{2a}$$

$$\Delta a_{ij}(n) = \lambda \left(h_i(n) - g_i(n)\right) x_j, \tag{2b}$$

where

$\Delta w_{ijk}(n) \equiv$ Change in the ijth connection weight from expert network i, $\Delta w_{ijk}(n) \in \Re^1$;
$\Delta a_{ij}(n) \equiv$ Change in the ijth connection weight from gate network, $\Delta a_{ij}(n) \in \Re^1$;
$\lambda \equiv$ Learning rate, $\lambda \in \Re^1$;
$h_i(n) \equiv$ Probability that ith expert generates a desired response, $h_i \in \Re^1$;
$g_i(n) \equiv$ Gate function output for ith expert, $g_i \in \Re^1$;
$e_i(n) \equiv$ Error resulting from output of the ith expert, $e_i \in \Re^{n\times 1}$; and
$x_j(n) \equiv$ Input signal j, $x_j \in \Re^1$.

2.3. Manipulator Operations Using Value Encoding

The MOVE NN was chosen for investigation based on the large decrease in trajectory tracking error obtained by Graham and D'Eleuterio (1990). Originally based on the CMAC (cerebellar model articulation controller) (Albus 1975), MOVE incorporates input discretization with the learning capabilities of an NN. As the underlying basis of the network architecture, the CMAC structure acts as a preprocessor that activates output units based on the value of the input signal. The signals from the CMAC output units are used as the inputs to a single-layer EBP NN. An extension to MOVE has been made, in the work reported here, to include two networks running in parallel as in the ME network. The result is a network that benefits from the input discretization of CMAC, the learning capabilities of EBP and the modularity of ME.

Figure 2, illustrating the CMAC technique of encoding the input signals, shows how the CMAC structure is incorporated into a control system. The grid structure facilitates the CMAC input-output mapping. In this figure, the two input states to be encoded are q and $\dot{q}$. The current values of the input states q_i and $\dot{q}_j$ will activate one element in each of five separate grids, with each grid offset with respect to the others. The grids are composed of units called course cell units. Once the course cell units are activated, the CMAC portion of the forward propagation of the network is complete. As shown in Figure 3, the output from each of the coarse cell units is connected to a unit in the next layer of the network. These units are called granule cell units. To reduce the memory required, a number of course cell units are randomly connected, i.e., hashed in such a way that several course cell units are connected to one granule cell unit. The output from the granule cell units is set to 1 if any of the connecting coarse cell units are activated, or 0 otherwise (i.e., logical OR statement). The granule cell units are used as neurons in a feed-forward artificial NN. Every granule cell is connected to the output cell unit. The outputs from the granule cell units are all weighted and summed to produce the final value from the output unit.

The weights connecting the granule cells to the output unit of the network are variable. Learning of the network occurs by adjusting the value of the variable weights over time so

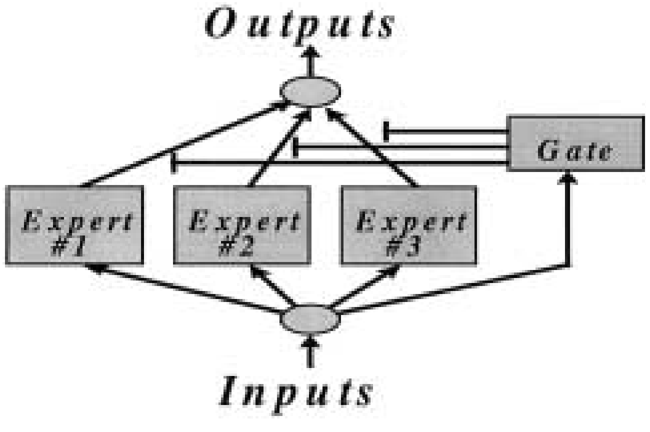


Fig. 1. The mixture of experts neural network architecture.

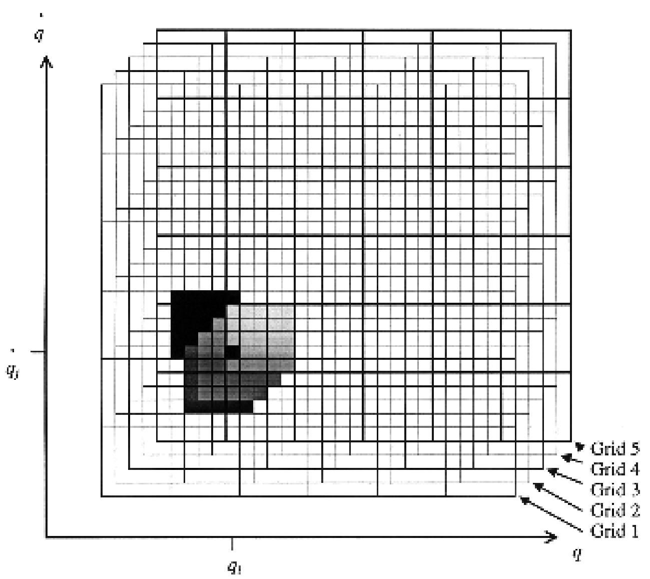
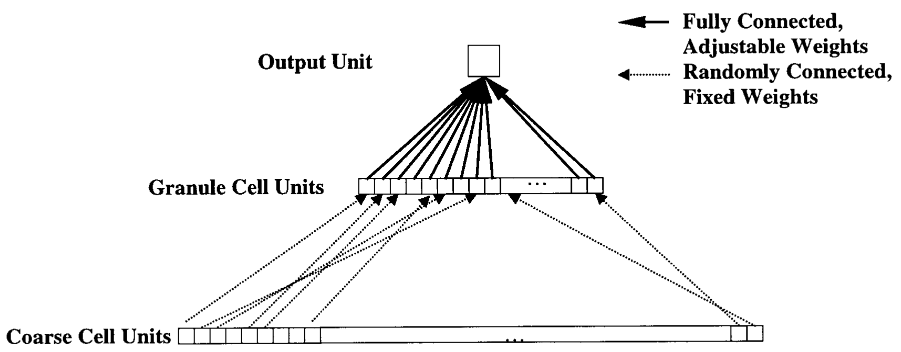


Fig. 2. Cerebellar model articulation controller (CMAC) encoding technique.


Fully Connected, Adjustable Weights
Randomly Connected, Fixed Weights
Output Unit
Granule Cell Units
Coarse Cell Units
Fig. 3. Manipulator operations using value encoding (MOVE) neural network topology.

as to obtain a desired output. The weight adjustment rule is given as

$$\Delta w_{ij}(n) = \lambda \frac{\Psi_{i,j}(n)}{\sum_j \sum_i \Psi_{i,j}(n)} J(n), \tag{3}$$

where

$\Delta w_{ij}(n) \equiv$ Change in connection weight, $\Delta w_{ij}(n) \in \Re^1$;
$\lambda \equiv$ Learning rate, $\lambda \in \Re^1$;
$\Psi_{i,j}(n) \equiv$ Activated granule cell output, $\Psi_{i,j}(n) \in \Re^1$; and
$J(n) \equiv$ Cost Function, $J(n) \in \Re^1$.

Since only the weights in the final layer of the network are adjusted iteratively, when compared with EBP and ME architectures that contain many more weights, a reduced learning time is obtained. A full description of the algorithm, including mathematical details, can be found in Graham and D'Eleuterio (1990).

3. Development of a Neural Network Control Test-Bed Module

Almost all industrial robots employ joint level PID position controllers. Given this fact, it was determined that a generic NN module controller to be used in conjunction with the PID controllers already being used in industrial robots would be developed. Details of this development are found in Chen et al. (1998). The NN module structure is illustrated in Figure 4. Due to the modular structure of the software, different NNs are readily implemented with this system.

In this development, we consider a joint-level PID-controlled robot. The open-loop dynamics of a rigid link robot with n actuated joints is expressed as

$$M(q)\ddot{q} + h(q, \dot{q}) = \tau, \tag{4}$$

where

$q \equiv$ Joint position, $q \in \Re^{n\times 1}$;
$\dot{q} \equiv$ Joint velocity, $\dot{q} \in \Re^{n\times 1}$;

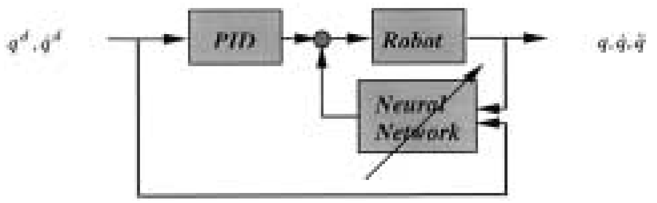


Fig. 4. Block diagram of a robot with a proportional, integral, and derivative (PID) and neural network controller.

$\ddot{q} \equiv$ Joint acceleration, $\ddot{q} \in \Re^{n\times 1}$;
$\tau \equiv$ Torque input signal, $\tau \in \Re^{n\times 1}$;
$M(\cdot) \equiv$ Inertia matrix, $M(\cdot) \in \Re^{n\times n}$; and
$h(\cdot,\cdot) \equiv$ Coriolis, centripetal and gravitational term, $h(\cdot,\cdot) \in \Re^{n\times n}$.

Under the standard assumption of fast actuator dynamics, the input torque τ signal is given by the following PID control law:

$$\tau = K_P(q_d - q) + K_I \int_0^t (q_d - q)dt + K_D(\dot{q}_d - \dot{q}) = T_{PID}, \tag{5}$$

where

$q_d \equiv$ Desired joint position, $q_d \in \Re^{n\times 1}$;
$\dot{q}_d \equiv$ Desired joint velocity, $\dot{q}_d \in \Re^{n\times 1}$;
$K_P \equiv$ Proportional gain matrix, $K_P \in \Re^{n\times n}$;
$K_I \equiv$ Integral gain matrix, $K_I \in \Re^{n\times n}$; and
$K_D \equiv$ Derivative gain matrix, $K_D \in \Re^{n\times n}$.

An NN signal ν is added to the control law to compensate for trajectory tracking errors, as given below.

$$\tau = K_P(q_d - q) + K_I \int_0^t (q_d - q)dt + K_D(\dot{q}_d - \dot{q}) + \nu. \tag{6}$$

Let the error e and control error Δv be defined below:

$$e = q_d - q \tag{7}$$

$$\Delta v = M(q)\ddot{q} + h(q, \dot{q}) - v. \tag{8}$$

With eqs. (6), (7), and (8), the closed-loop error dynamics of the system are

$$\Delta v = K_P e + K_I \int_0^t e dt + K_D \dot{e}. \tag{9}$$

If the NN can learn to compensate for the nonlinear dynamics in (8), i.e., $\Delta v \to 0$, then the closed-loop dynamics becomes

$$0 = K_P e + K_I \int_0^t e dt + K_D \dot{e}. \tag{10}$$

Selection of appropriate gains will lead to asymptotic trajectory tracking.

3.1. Robot Experiment Hardware

Experiments utilized a CRS Robotics Corporation A460 robot and a transputer-based C500 controller. The robot has 6 degrees of freedom and uses permanent magnet DC motors to activate harmonic drive gears. Table 1 lists the CRS A460 robot kinematics and dynamics parameters.

Each NN software module is programmed in C language, then compiled and downloaded to a Texas Instruments TM320C40 Digital Signal Processor (DSP). Real-time execution of NN learning, weight update, and the PID controller occurs at 500 Hz. A more detailed description of the hardware and software setup can be found in Chen et al. (1998).

Two host 486 PCs act as the user interfaces for the DSP and robot controller, respectively. These PCs initiate experiments, and control software execution, data recording, and selection of experimental parameters. Figure 5 illustrates the system hardware configuration.

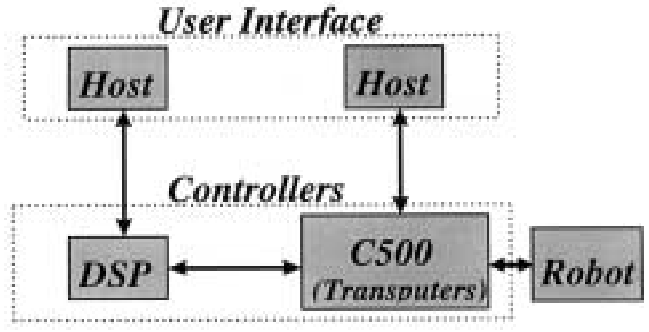


Fig. 5. Robot experimental hardware configuration.

3.2. Robot Experiment Software

Figure 6 illustrates the system software architecture. The DSP host enables a user to compile and download any type of NN to the DSP. Matlab software is used for the front-end of the programming and enables NN parameter selection as well as the generation of initial weights.

The NN module consists of procedures written in C for (a) network output generation and (b) learning algorithms, i.e., weight adjustments. The DSP communication module sends and receives data to and from the C500 communication module. The DSP communication module must (a) establish communication with the C500, (b) send NN output signals to the C500 to provide the compensating torque signal, and (c) receive the actual and desired values of joint position, velocity, and acceleration from the C500.

The DSP execution manager supervises the execution of procedures in the DSP. It ensures a proper schedule for real-time execution. The procedures include (a) NN output generation, (b) NN learning, (c) communication with the C500 controller, and (d) data recording.

The C500 host allows the user to select parameters for the PID controller and experiment execution. Important selections include robot trajectory, PID gains, maximum robot velocity and accelerations, and the number of trials in an experiment. Experiment execution and termination are also controlled via the C500 host.

The C500 communication module works with the DSP communication module to carry out data transfer. Its responsibilities include (a) establishing communication with the DSP; (b) sending actual and desired values of joint position, velocity, and acceleration; and (c) receiving the NN output signals from the DSP. The user control program contains the PID control algorithm. Modifications were made to include the addition of the NN compensating torque signal into the control scheme.

The C500 execution manager acts in a supervisory role coordinating activities of other software modules while incorporating the real-time NN communication and execution. A detailed description of the software flow chart can be found in Chen et al. (1998).

4. Sensitivity Functions and the Development of a Neural Network/Robot Control Simulation

The system sensitivity function (Frank 1976) provides a measure of how a closed-loop system will behave given a variation in a system parameter. In this section, we introduce the sensitivity equations for the PID-controlled robot system with an NN compensator. The sensitivity equations of robot joint trajectory tracking error with respect to NN learning rate are then derived. Details of the numerical solution of the sensitivity equations are given.

Table 1. CRS A460 Robot Data

Link	Twist α (deg)	Length a (in)	Offset d (in)	Angle θ (deg)
1	+90	0	0	θ_1
2	0	$a_2 = 12.125$	0	θ_2
3	−90	0	0	θ_3
4	+90	0	$d_4 = 12.125$	θ_4
5	−90	0	0	θ_5
6	0	0	0	θ_6

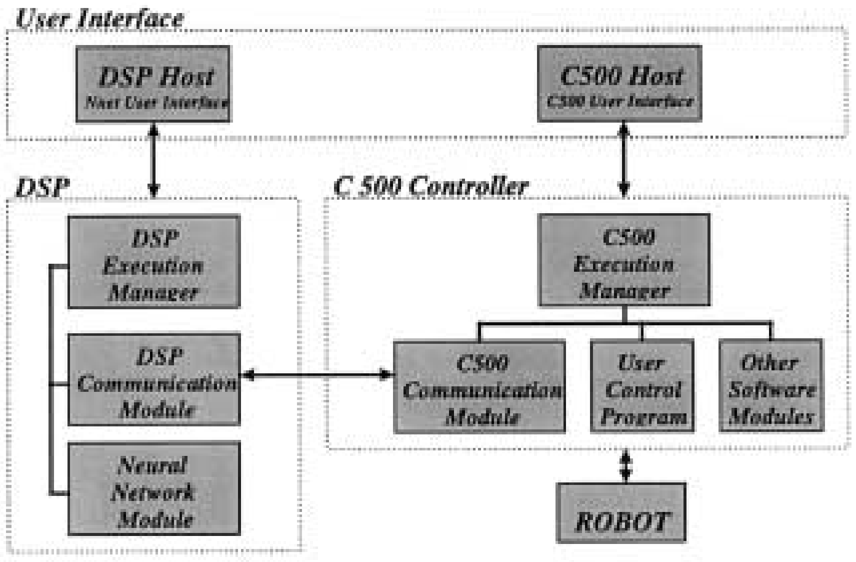


Fig. 6. Robot experimental software architecture.

4.1. Sensitivity Functions

The state sensitivity is defined as the change of a state value relative to the change of a system parameter (Frank 1976). The sensitivity function ς is defined as below.

$$\varsigma = \frac{\partial z}{\partial \alpha}, \tag{11}$$

where
$\varsigma \equiv$ Sensitivity of state z to parameter α, $\varsigma \in R^{n\times 1}$;
$z \equiv$ System state, $z \in R^{n\times 1}$; and
$\alpha \equiv$ A system parameter, $\alpha \in R^1$.
Consider a system with r parameters given by

$$\dot{z} = f(z, \alpha, t). \tag{12}$$

The sensitivity function of the system is given by partial differentiation of (12) with respect to the system parameter α as

$$\frac{\partial \dot{z}}{\partial \alpha} = \frac{\partial f}{\partial z}\frac{\partial z}{\partial \alpha} + \frac{\partial f}{\partial \alpha}, \tag{13}$$

where
$\frac{\partial \dot{z}}{\partial \alpha} \equiv$ Sensitivity derivative, $\frac{\partial \dot{z}}{\partial \alpha} \in \Re^{n\times 1}$;
$\frac{\partial f}{\partial z} \equiv$ Jacobian matrix, $\frac{\partial f}{\partial z} \in \Re^{n\times n}$;
$\frac{\partial z}{\partial \alpha} \equiv$ Sensitivity, $\frac{\partial z}{\partial \alpha} \in \Re^{n\times 1}$; and
$\frac{\partial f}{\partial \alpha} \equiv$ System parameters, $\frac{\partial f}{\partial \alpha} \in \Re^{n\times 1}$.

From eq. (11), the sensitivity function dynamics are expressed as

$$\dot{\varsigma} = \frac{\partial}{\partial t}\left(\frac{\partial z}{\partial \alpha}\right) = \frac{\partial \dot{z}}{\partial \alpha}. \tag{14}$$

Substituting eq. (14) into eq. (13) yields the sensitivity equation

$$\dot{\varsigma} = f_z \varsigma + f_\alpha, \tag{15}$$

where
$\dot{\varsigma} \equiv$ Sensitivity derivative, $\dot{\varsigma} \in \Re^{n\times 1}$;
$f_z \equiv \frac{\partial f}{\partial z} \equiv$ Jacobian matrix, $f_z \in \Re^{n\times n}$;
$\varsigma \equiv$ Sensitivity, $\varsigma \in \Re^{n\times 1}$; and
$f_\alpha \equiv \frac{\partial f}{\partial \alpha} \equiv$ System parameters, $\frac{\partial f}{\partial \alpha} = f_\alpha \in \Re^{n\times 1}$.

4.2. The System State Model

Simultaneous solution of eq. (15) and the system dynamics given by (12) will yield the sensitivity of the closed-loop system with respect to any parameter α. To solve these equations, a robot simulation developed with Simulink was used (i.e., Liu and Mills 1998). Here, we present a derivation of the model, including the robot dynamics and weight dynamics of the system.

Using a first-order approximation, the weight dynamics, given in eqs. (1), (2), and (3), are expressed as

$$\dot{w}_{ij} \approx \frac{w_{ij}(n) - w_{ij}(n-1)}{\Delta T} = \frac{1}{\Delta T}\Delta w_{ij}, \tag{16}$$

where $\Delta T \equiv$ Neural network update period, $\Delta T \in \Re^1$.

The dynamics of the ijth weight of an NN trained with EBP is

$$\dot{w}_{ij} = -\tilde{\lambda}\frac{\partial J}{\partial w_{ij}}, \tag{17}$$

where

$w_{ij} \equiv$ Weight connecting the ith and jth neuron, $\dot{w}_{ij} \in \Re^1$;

$\frac{\partial J}{\partial w_{ij}} \equiv$ Cost function derivative with respect to the weight w_{ij}, $\frac{\partial J}{\partial w_{ij}} \in \Re^1$; and

$\tilde{\lambda} \equiv$ Modified learning rate (i.e., divided by ΔT), $\tilde{\lambda} \in \Re^1$.

Using the delta function δ_j as described in Haykin (1994) and the neuron output y_j, eq. (17) can be rewritten as

$$\dot{w}_{ij} = \tilde{\lambda}\delta_j y_j, \tag{18}$$

where

$y_j \equiv$ The output of the jth neuron, $y_j \in \Re^1$ and

$\delta_j \equiv$ Delta function for the jth neuron, $\delta_j \in \Re^1$.

Let the vector $w \in \Re^{rxsxl}$ represent the weights of the NN, as follows,

$$w = \begin{bmatrix} \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{rs} \end{bmatrix}_1 \\ \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{rs} \end{bmatrix}_2 \\ \vdots \\ \begin{bmatrix} w_{11} \\ w_{12} \\ \vdots \\ w_{rs} \end{bmatrix}_l \end{bmatrix}. \tag{19}$$

Hence, the weight dynamics are represented as

$$\dot{w} = \begin{bmatrix} \begin{bmatrix} \tilde{\lambda}\delta_1 y_1 \\ \tilde{\lambda}\delta_2 y_2 \\ \vdots \\ \tilde{\lambda}\delta_s y_s \end{bmatrix}_1 \\ \begin{bmatrix} \tilde{\lambda}\delta_1 y_1 \\ \tilde{\lambda}\delta_2 y_2 \\ \vdots \\ \tilde{\lambda}\delta_s y_s \end{bmatrix}_2 \\ \vdots \\ \begin{bmatrix} \tilde{\lambda}\delta_1 y_1 \\ \tilde{\lambda}\delta_2 y_2 \\ \vdots \\ \tilde{\lambda}\delta_s y_s \end{bmatrix}_l \end{bmatrix} = f(\lambda, y). \tag{20}$$

From eqs. (4) and (6), the robot dynamics are expressed as

$$\begin{aligned} M(q)\ddot{q} + h(q,\dot{q}) = K_P(q_d - q) + K_I \int_0^T (q_d - q)dt \\ + K_D(\dot{q}_d - \dot{q}) + v. \end{aligned} \tag{21}$$

Isolating $\ddot{q}$ in eq. (21) yields

$$\begin{aligned} \ddot{q} = M^{-1}(q)[K_P(q_d - q) + K_I \int_0^T (q_d - q)dt \\ + K_D(\dot{q}_d - \dot{q}) - h(q,\dot{q}) + v]. \end{aligned} \tag{22}$$

By assigning the state variables as below,

$$\begin{aligned} x_1 &= q \in \Re^{n\times 1} \\ x_2 &= \dot{q} \in \Re^{n\times 1} \\ x_3 &= \int_0^t (q_d - q)d\sigma \in \Re^{n\times 1} \\ \dot{x}_1 &= x_2 \\ \dot{x}_2 &= M^{-1}(x_1)[K_P(x_{1d} - x_1) + K_1 x_3 + K_D(x_{2d} - x_2) \\ &\quad - h(x_1, x_2) + v(x_1, x_2, w)], \\ \dot{x}_3 &= x_{1d} - x_1 \end{aligned} \tag{23}$$

the system can now be expressed in block matrix form as

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix}=\begin{bmatrix}0 & I_n & 0\\ 0 & 0 & 0\\ -I_n & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}+\begin{bmatrix}0\\ -M^{-1}h\\ 0\end{bmatrix}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\end{bmatrix}\begin{bmatrix}0 & 0 & 0\\ -K_P & -K_D & K_I\\ 0 & 0 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\end{bmatrix}\begin{bmatrix}0\\ K_P x_{1d}+K_D x_{2d}\\ x_{1d}\end{bmatrix} \tag{24}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\end{bmatrix}\begin{bmatrix}0\\ v\\ 0\end{bmatrix}.$$

The overall system state model is defined with the state vector z,

$$z=\begin{bmatrix}z_1\\ z_2\\ z_3\\ z_4\end{bmatrix}=\begin{bmatrix}x_1\\ x_2\\ x_3\\ w\end{bmatrix}, \tag{25}$$

to give

$$\begin{bmatrix}\dot{z}_1\\ \dot{z}_2\\ \dot{z}_3\\ \dot{z}_4\end{bmatrix}=\begin{bmatrix}0 & I_n & 0 & 0\\ 0 & 0 & 0 & 0\\ -I_n & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}z_1\\ z_2\\ z_3\\ z_4\end{bmatrix}+\begin{bmatrix}0\\ -M^{-1}h\\ 0\\ f(\lambda, y)\end{bmatrix}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\\ 0\end{bmatrix}\begin{bmatrix}0 & 0 & 0 & 0\\ -K_P & -K_D & K_I & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}z_1\\ z_2\\ z_3\\ z_4\end{bmatrix}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\\ 0\end{bmatrix}\begin{bmatrix}0\\ K_P z_{1d}+K_D z_{2d}\\ z_{1d}\\ 0\end{bmatrix} \tag{26}$$
$$+\begin{bmatrix}0\\ -M^{-1}\\ 0\\ 0\end{bmatrix}\begin{bmatrix}0\\ v\\ 0\\ 0\end{bmatrix},$$

where $f(\lambda, y)$ is defined in eq. (20).

4.3. Robot Simulation Software

The simulation of eqs. (15) and (26) was carried out using Simulink and Matlab with a fourth-order Runge-Kutta algorithm.

5. Experiment and Simulation Results

In this section, we calculate the sensitivity functions of the robotic system under NN control. We briefly discuss NN implementation details, and the validity of the robot simulation is confirmed. Finally, sensitivity plots from experiments and simulations are given.

5.1. Experiment and Simulation Neural Network Implementation

A series of experiments was conducted to establish a baseline performance for each of the three NN controllers implemented. This required that a number of parameters, specific to each NN, i.e., learning rate, number of neurons in each layer, and so on, be tuned to achieve acceptable behavior.

Experiments were conducted on a CRS Robotics Corporation 6-degree-of-freedom A460 industrial robot. The NN's compensation signal was applied to only the first three joints of the robot. The rationale for this is based on the fact that the robot trajectory error is mainly due to trajectory tracking errors of the first three joints. In the following, an "experiment" is defined as the consecutive execution of a number of trials, while a "trial" is defined as the execution of a commanded trajectory sequence once, by the robot. The trajectory input consists of a set of "way points" connected by smooth trajectories generated from spline functions. These way points are listed in Table 2 with the joint trajectories plotted in Figure 7.

A valid comparison of the three NNs implemented dictated that all experiments be conducted with certain parameters and inputs held constant. These parameters include, for example, the input trajectory, PID gains, and NN parameters, and are given in Tables 2 and 3. Additionally, within each set of trials

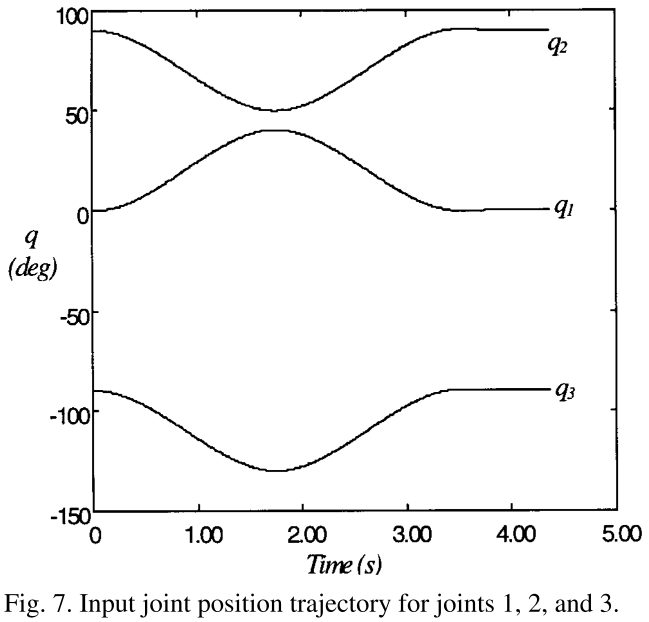


Fig. 7. Input joint position trajectory for joints 1, 2, and 3.

Table 2. Simulation and Experiment Robot Parameters

Robot Experiment Parameter	
Maximum joint velocity (rad/s)	0.6
Robot trajectory way points (degrees)	[0 0 0], [40 −40 −40], [0 0 0]
Proportional gains (for three joints)	[5500 6500 7500]
Integral gains (for three joints)	[2 2 2]
Derivative gains (for three joints)	[60 60 60]

Table 3. Experiment and Simulation Neural Network Parameters

Neural Network Experiment Parameter	EBP	ME	MOVE
Number of neurons in input layer	9	9	9
Number of neurons in hidden layer 1	7	7	NA
Number of neurons in hidden layer 2	5	5	1000
Number of neurons in output layer	3	3	3
Number of experts	NA	2	2
Number of grids in MOVE	NA	NA	5
Number of cells per grid in MOVE	NA	NA	3
Learning rate: λ_e	3E-8	2	6
Output limits	[40 40 40]	[60 75 70]	[55 65 75]

NOTE: EBP = error back-propagation; ME = mixture of experts; MOVE = manipulator operations using value encoding.

for a particular NN implementation, a number of parameters given in the following sections were also held fixed.

The input to the NNs consisted of the position and velocity of the first three joints of the robot. The output of the NN consists of three signals, ν, as given in eq. (6). These compensating signals, as seen in the following, lead to reduction in error beyond which the PID control can achieve.

To provide a measure of the trajectory tracking performance, the time averaged joint error norm for each trial is calculated according to the equation below, and it is plotted.

$$E = \frac{1}{T}\int_0^T \sqrt{e_1^2 + e_2^2 + e_3^2}dt, \tag{27}$$

where

$E \equiv$ Time averaged joint error norm, $E \in \Re^1$;
$e_i \equiv q_{id} - q_i \equiv$ Joint error, $i = 1, 2, 3$;
$q \equiv$ Joint position, $q \in \Re^{n\times 1}$;
$q_d \equiv$ Desired joint position, $q_d \in \Re^{n\times 1}$; and
$T \equiv$ Trial Period, $T \in \Re^1$.

5.2. Neural Network Performance: Simulation and Experiment

Simulation studies were conducted to permit robot system sensitivity to be calculated without noise and uncertainty of our experiment, which can mask important results. Comparison of experimental data with the simulations then allowed us to analyze the nature of our experimental results more easily. To undertake this task, experiments and simulations were conducted with identical PID and NN controller parameters. Robot simulations produced similar performance results to those obtained from the robot experimental results. Illustrated in Figure 8 are time-averaged joint error norm plots for the robot experiments and simulations using the EBP, ME, and MOVE NNs. All three plots show a similar reduction in joint error norm for the simulation and experiment.

To provide insight into the effect of learning rate on the performance of robotic systems with NN controllers, sensitivity calculations were carried out. To make a valid comparison of the sensitivity functions for the three NNs tested, the sensitivity function, given by (11), is normalized as follows:

$$\varsigma_{\text{norm}} = \frac{\partial z/\|z\|}{d\lambda/\lambda_s} = \varsigma\frac{\lambda_s}{\|z\|}, \tag{28}$$

where

$\varsigma_{\text{norm}} \equiv$ Normalized sensitivity, $\varsigma_{\text{norm}} \in \Re^{n\times 1}$;
$\varsigma \equiv$ Sensitivity, $\varsigma \in \Re^{n\times 1}$; and
$\|z\| \equiv$ State norm, $\|z\| \in \Re^1$.

Simulation results of normalized sensitivity of joint 1 trajectory tracking error to learning rate, λ, for EBP, ME, and MOVE systems, is shown in Figures 9 through 11. The joint error norm, E, given by (27), corresponding to each simulation, is also plotted. These plots are representative of the dynamic behavior seen in joints 2 and 3. From these plots, it is noted that the joint 1 trajectory tracking error sensitivity to learning rate is the highest as the joint error norm decreases at the greatest rate. This is due to the fact that as the error in the system decreases most rapidly, during higher rates of change in the NN weights, small changes in learning rate will lead to greater changes in the system behavior, resulting in greater sensitivity. Hence, during the period in which the changes in

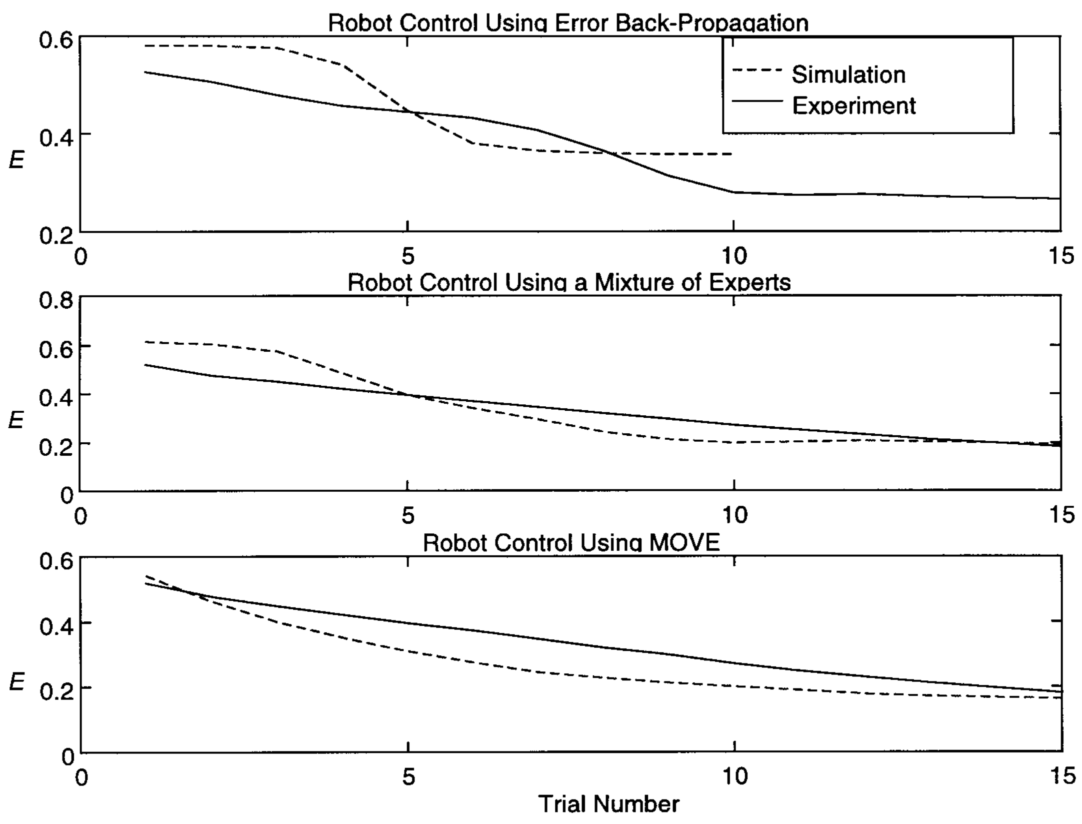


Fig. 8. Joint error norm for experimental and simulation results.
NOTE: MOVE = manipulator operations using value encoding.

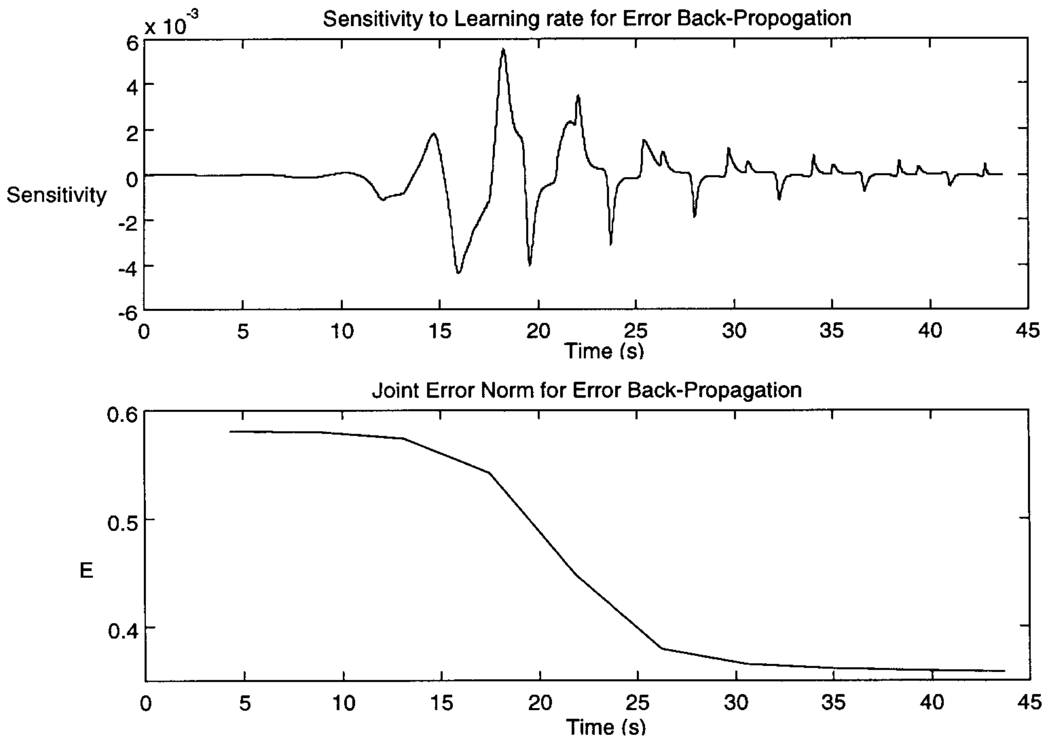


Fig. 9. Normalized learning rate sensitivity versus joint error norm for an error back-propagation neural network, simulation results.

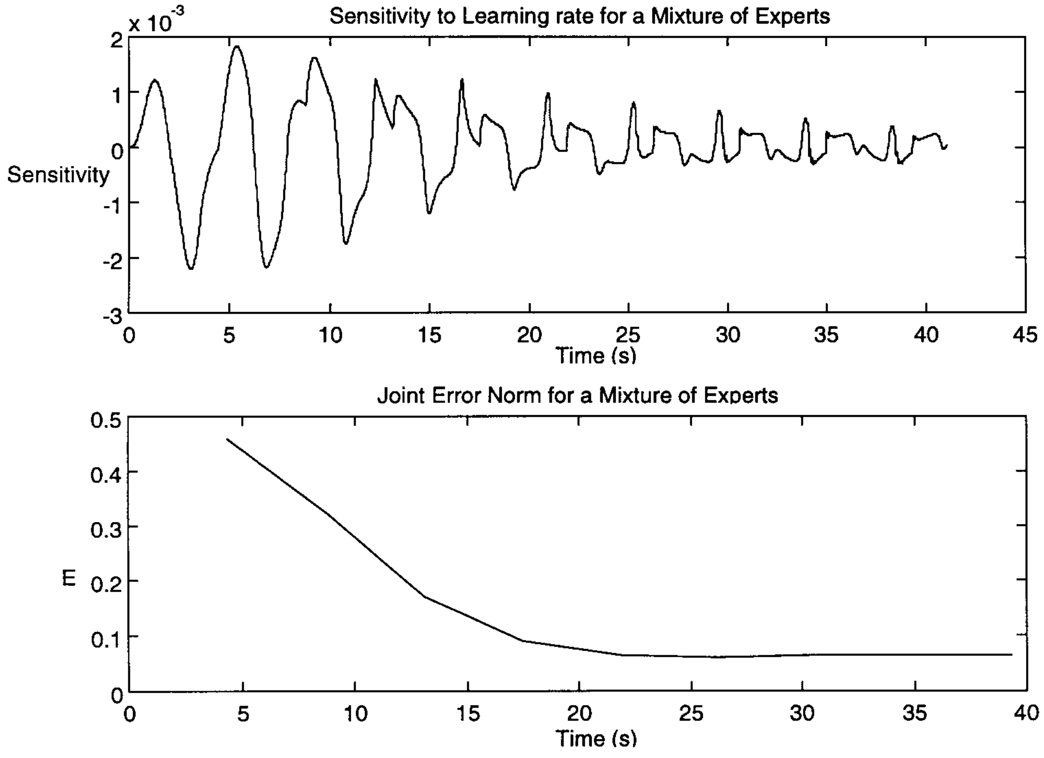


Fig. 10. Normalized learning rate sensitivity versus joint error norm for a mixture of experts neural network, simulation results.

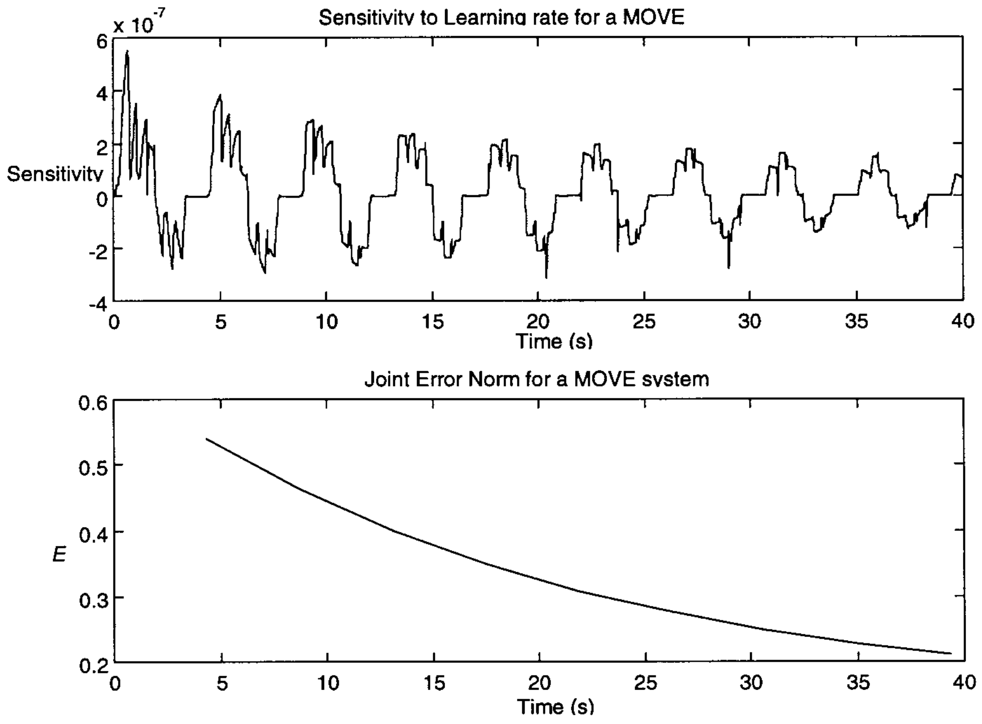


Fig. 11. Normalized learning rate sensitivity versus joint error norm for a manipulator operations using value encoding (MOVE) neural network, simulation results.

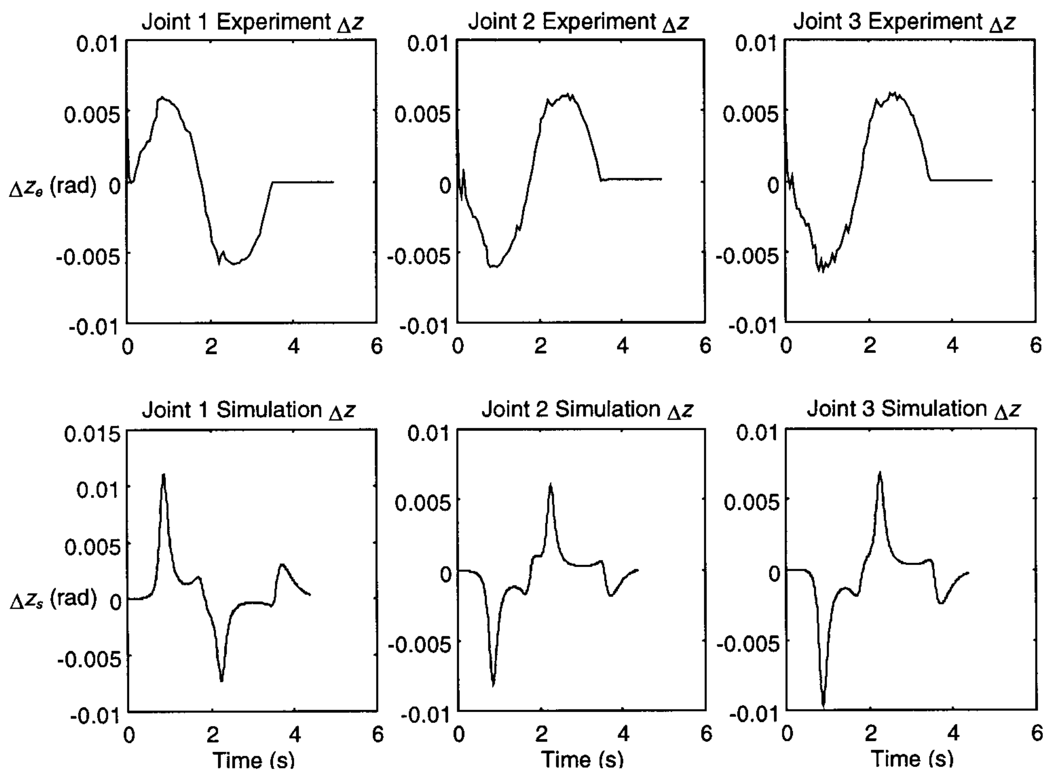


Fig. 12. Experimental and simulation results, Δz versus time for error back-propagation.

the weights are greatest, it would be anticipated that the system state is most sensitive to variations in learning rate. From these plots, it is seen that the MOVE NN controller exhibited the least sensitivity to learning rate variation.

The change in robot joint trajectory error state (Δz_e) resulting from a change in learning rate ($\Delta\lambda$) obtained from robot experiments is compared with results from the robot simulation sensitivity (Δz_S). The simulation learning rate perturbation, $\Delta\lambda_s$, corresponds to the experiment learning rate perturbation, but is normalized with respect to λ_e/λ_s, as shown below.

$$\Delta z_e = z(\lambda_e + \Delta\lambda) - z(\lambda_e) \tag{29}$$

$$\Delta z_s = \frac{\partial z}{\partial \lambda}\Delta\lambda_s, \tag{30}$$

where

$z(\cdot) \equiv$ Experiment system state as a function of λ, $z(\cdot) \in \Re^{n\times 1}$;

$\Delta z_e \equiv$ Change in experiment state z caused by perturbation in λ, $\Delta z_e \in \Re^{n\times 1}$;

$\Delta z_s \equiv$ Change in simulation state z caused by perturbation in λ, $\Delta z_s \in \Re^{n\times 1}$;

$\lambda_e \equiv$ Learning rate used in experiment, $\lambda_e \in \Re^1$;

$\lambda_s \equiv$ Learning rate used in simulation, $\lambda_s \in \Re^1$;

$\Delta\lambda_e \equiv$ Experiment learning rate perturbation, $\Delta\lambda_e \in \Re^1$; and

$\Delta\lambda_s \equiv$ Experiment learning rate perturbation with respect to λ_s, $\Delta\lambda_e \in \Re^1$.

Experimental and simulation results of the variation in trajectory tracking error, denoted by Δz_e and Δz_S, respectively, for joints 1, 2, and 3, are illustrated in Figures 12 through 14 for the EBP, ME, and MOVE NN controllers. We note that

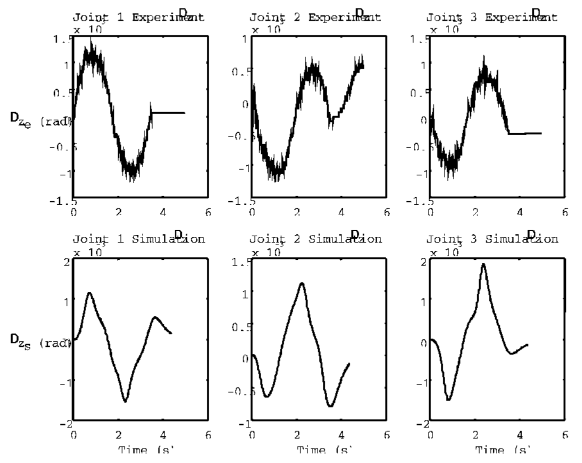


Fig. 13. Experimental and simulation results, Δz versus time for mixture of experts.

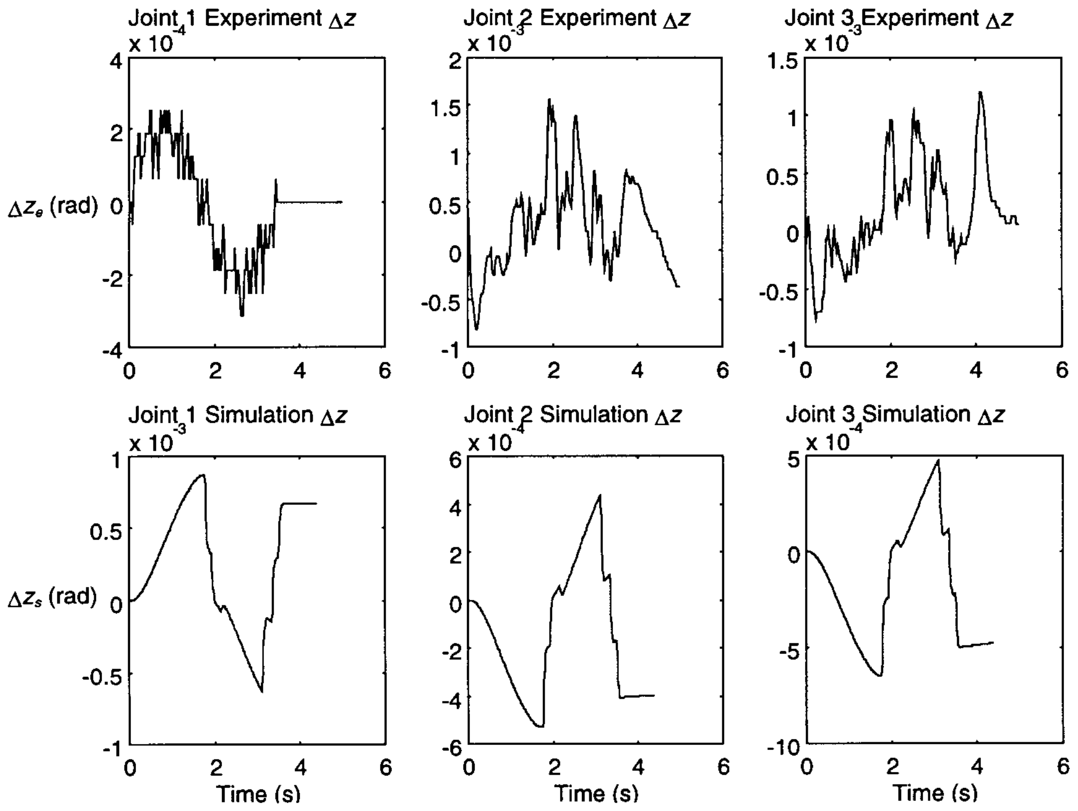


Fig. 14. Experimental and simulation results, Δz versus time for a manipulator operations using value encoding (MOVE) controller with $\lambda = 80$.

in each set of plots, the dynamic behavior obtained from the simulation is very similar to the experimental results. This similarity confirms the validity of our simulations as a diagnostic tool in our work.

Calculation of Δz_e and Δz_S allows a designer of an NN controller to predict the effect of variations in learning rate on overall system performance. Conversely, in the process of selection of a suitable learning rate for operation of an NN, low sensitivity to perturbations in the value of the learning rate indicates that the choice of learning rate is not critical to neural overall system behavior.

From Figures 12 through 14, it is seen that the lowest sensitivity of joint trajectory tracking error to learning rate was achieved by the MOVE and ME NN controllers, i.e., the MOVE and ME Δz values were the smallest when compared with those Δz values obtained using EBP.

6. Conclusion

Learning rate sensitivity simulation results were shown to correspond well with results obtained from robot experiments. The simulation of learning rate sensitivity proved to be a good predictor of actual system behavior. Simulations indicate that the MOVE NN exhibited the lowest joint trajectory error sensitivity to learning rate when compared to the EBP and ME NN controllers. Experimental results, while similar to simulations, indicated that the MOVE and ME NN controller architectures resulted in similar sensitivity to learning rate, both lower than the EBP architecture. Differences in trajectory tracking error sensitivity to learning rate, among three NN controller architectures, provide insight into both the selection of learning rates for robotic systems and the effect of nonoptimal learning rates on system performance.

Our experimental and simulation work supports two claims. First, the MOVE and ME NN will operate successfully over a wider range of learning rates than the EBP NN controller. Conversely, with the EBP NN controller, the closed-loop system performance is more sensitive to learning rate; hence, more care must be taken in selection of a suitable NN learning rate. Second, when selection of NN learning rate is undertaken, the fact that the MOVE and ME NN controller is least sensitive to learning rate, when compared to the EBP controller, makes selection of a suitable learning rate a simpler task. Hence, a wider range of appropriate learning rates can be used when using the MOVE and ME NNs as opposed to when using an EBP NN. While qualitative in nature, this work represents a first step toward a rationale for selection of NN learning rate.

The results of this paper provide a mechanism for the investigation of sensitivity of NN performance to learning rate variation, and specific results for three NN architectures. The benefits of the work presented are twofold. First, when adjusting learning rates, it is beneficial to know how sensitive the system performance is to the learning rate. For example, if it is known that the system performance is strongly dependent on learning rate, then considerable care must be made in tuning this parameter. Our work clearly demonstrates that there is a variation in sensitivity to learning rate between NN controller architectures; hence, this aspect of controller design has merit. Second, knowing that one NN architecture has a lower sensitivity to learning rate variation allows more rapid tuning of learning rates, leading to cost savings during the tuning procedure. Selection of learning rates carried out in the absence of such sensitivity information may lead to costly tuning without substantial benefit, i.e., small changes made to learning rate that have little impact on performance, or conversely, use of an NN architecture that is strongly dependent on NN learning rate, leading to difficulties in learning rate selection.

References

Albus, J. S. 1975. A new approach to manipulator control: Cerebellar model articulation controller (CMAC). *Trans. ASME J. Dynamic Systems, Measurement and Control*, September, pp. 220–223.

Chen, P.C.Y., Mills, J. K., and Smith, K. C. 1996. Performance improvement of robot continuous-path operation through iterative learning using neural networks. *Machine Learning Journal* 23:191–220.

Chen, P.C.Y., Ogilvie, A., Clark, C., Zhou, K., and Mills, J. K. 1998. Development of a neural network module for improving the performance of a commercial robot. *World Automation Conference*, Alaska, May, pp. 288–293.

Frank, W. P. 1976. *Introduction to Sensitivity Theory*. San Diego: Academic Press.

Graham, D.P.W., and D'Eleuterio, G.M.T. 1990. MOVE—A neural network paradigm for robotic control. *6th CASI Conference on Astronautics*, Ottawa, Canada, October 21, pp. 588–593.

Graham, D.P.W., and D'Eleuterio, G.M.T. 1991. Robotic control using a modular architecture of cooperative artificial neural networks. In *Artificial Neural Networks*, vol. 1, ed. T. Kohonen, 365–370. Amsterdam: North Holland.

Haykin, S. 1994. *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ: Macmillan College.

Jacobs, R., and Jordan, M. I. 1993. Learning piecewise control strategies in a modular neural network architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(2):337–345.

Jacobs, R., Jordan, M. I., Nowlan, S. J., and Hinton, G.E. 1991. Adaptive mixtures of local experts. *Neural Computation* 3(1):79–87.

Liu, H. T., and Mills, J. K. 1998. ROBOTOOL: A simulation software for robot performance analysis. *Proceedings of the Canadian Society for Mechanical Engineering Forum*, Toronto, Ontario, June, pp. 1044–1049.

Rueckl, J. G., Cave, K. R., and Kosslyn, S. M. 1989. Why and "what" and "where" processed by separate cortical visual systems? A computational investigation. *Journal of Cognitive Neuroscience* 1:171–186.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. 1986. Learning representations by back-propagating error. *Nature (London)* 323:533–536.