NEURAL NETS FOR FEEDBACK CONTROL

Dynamical systems are ubiquitous in nature and include naturally occurring systems such as the cell and more complex biological organisms, the interactions of populations, and so on, as well as man-made systems such as aircraft, satellites, and interacting global economies. A. N. Whitehead and L. von Bertalanffy were among the first to provide a modern theory of systems at the beginning of the century. Systems are characterized as having outputs that can be measured, inputs that

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering.* Copyright © 1999 John Wiley & Sons, Inc.

can be manipulated, and internal dynamics. *Feedback control* involves computing suitable control inputs, based on the difference between observed and desired behavior, for a dynamical system so that the observed behavior coincides with a desired behavior prescribed by the user. All biological systems are based on feedback for survival, with even the simplest of cells using chemical diffusion based on feedback to create a potential difference across the membrane to maintain its *homeostasis,* or required equilibrium condition for survival. Volterra was the first to show that feedback is responsible for the balance of two populations of fish in a pond, and Darwin showed that feedback over extended time periods provides the subtle pressures that cause the evolution of species.

There is a large and well-established body of design and analysis techniques for feedback control systems. This work began with the Greeks and Arabs; was put on a firm basis by Watt, Maxwell, Airy, and others; and has been responsible for successes in the industrial revolution, ship and aircraft design, and the space age. Design approaches include classical design methods for OPTIMAL CONTROL; ROBUST CONTROL; H-INFINITY CONTROL; ADAPTIVE CONTROL; and others; for more information refer to the articles by those names. Many systems that we desire to control have unknown dynamics, modeling errors, and various sorts of disturbances, uncertainties, and noise. This, coupled with the increasing complexity of today's dynamical systems, creates a need for advanced control design techniques that overcome limitations on traditional feedback control techniques.

In recent years, there has been a great deal of effort to design feedback control systems that mimic the functions of living biological systems (1); refer to INTELLIGENT CONTROL. There has been great interest recently in "universal model-free controllers" that do not need a mathematical model of the controlled plant, but mimic the functions of biological processes to learn about the systems they are controlling on-line, so that performance improves automatically. Techniques include fuzzy logic control, which mimics linguistic and reasoning functions, and artificial neural networks, which are based on biological neuronal structures of interconnected nodes. Neural networks (NN) have achieved great success in classification and pattern recognition. Rigorous analysis has shown how to select NN topologies and weights, for instance, to discriminate between specified exemplar patterns. By now, the theory and applications of NN in classification are well understood, so that NNs have become an important tool in the repertoire of the signal processor and computer scientist.

Now, rigorous results are also beginning to appear in the uses of NN for control theory applications (1–4). In control theory, the NN weights must usually be tuned dynamically in time. There are two classes of applications—open-loop identification and closed-loop control. Identification is similar to classification applications, so that the same open-loop NN weight-tuning algorithms (e.g., backpropagation tuning) often work. In complete contrast is the situation in feedback control, where the NN becomes part of the closed-loop system so that special steps must be taken to guarantee that its weights stay bounded.

Although fraught with difficulties, NN applications in closed-loop control are increasing as indicated by a steady stream of published articles. Early papers consisted for the most part of ad hoc discussions followed by some simulation examples. Theoretical proofs and repeatable design algorithms (e.g., two conscientious engineers should get similar results) were for the most part absent. The basic problem issues in NN feedback control are

- To provide repeatable design algorithms
- To provide on-line learning algorithms that do not require preliminary off-line tuning
- To show how to initialize the NN weights to guarantee stability
- To rigorously prove closed-loop trajectory following
- To show how to compute various gradients needed for weight tuning
- To show that the NN weights remain bounded despite unmodeled dynamics (because bounded weights guarantee bounded control signals)

At higher levels, an issue is to provide more brainlike capabilities, such as generic learning to cope with complex problems requiring strategic capabilities over time. Also important are techniques for combining off-line learning and prior information with learning functions performed on-line in real time.

This article shows that NNs do indeed fulfill the promise held out of providing model-free learning controllers for a class of nonlinear systems, in the sense that not even a structural or parametrized model of the system dynamics is needed. All the basic problem issues just mentioned are solved for a large class of mechanical motion systems with Lagrangian dynamics, including robotic manipulators. The control structures discussed in this article are multiloop controllers with NNs in some of the loops and an outer tracking unity-gain feedback loop. Throughout, there are repeatable design algorithms and guarantees of system performance including both small tracking errors and bounded NN weights.

It is shown that as uncertainty about the controlled system increases or as we desire to consider human user inputs at higher levels of abstraction, the NN controllers acquire more and more structure, eventually acquiring a hierarchical structure that resembles some of the elegant architectures proposed by computer science engineers using high-level design approaches based on cognitive linguistics, reinforcement learning, psychological theories, adaptive critics, or optimal dynamic programming techniques. Such high-level control architectures are discussed in NEUROCONTROLLERS.

NN controllers have advantages over standard adaptive control approaches in that no linearity-in-the-parameters assumption is needed and no regression matrix must be determined. This is primarily due to the NN universal function approximation property. Moreover, if designed correctly, the NN controller does not need persistence of excitation or certainty equivalence assumptions.

BACKGROUND IN NEURAL NETWORKS AND FEEDBACK CONTROL

Neural Network Structures and Properties

There is a rich and varied literature on neural networks (5); see NEURAL NET ARCHITECTURE. NNs can be used for two classes of applications in system theory: signal processing/classification and control. There are two classes of control applications—open-loop identification and closed-loop feedback control. Identification applications are close in spirit to signal processing/classification, so that the same open-loop algorithms (e.g., backpropagation weight tuning) may often be used. On the other hand, in closed-loop feedback applications, the NN is inside the control loop so that special steps must be taken to ensure that the NN weights remain bounded during the control run. Until the 1990s NN applications in closed-loop feedback control were for the most part ad hoc with no design algorithms or guaranteed performance.

Static Feedforward Neural Networks. A feedforward neural network is shown in Fig. 1. This NN has two layers of adjustable weights and is called here a two-layer net. The NN output y is a vector with m components that are determined in terms of the n components of the input vector x by the formula

$$y_i = \sum_{j=1}^{L}\left[w_{ij}\sigma\left(\sum_{k=1}^{n} v_{jk}x_k + \theta_{v_j}\right) + \theta_{w_i}\right];\ i = 1,\ldots,m \quad (1)$$

where $\sigma(\cdot)$ are the activation functions and L is the number of hidden-layer neurons. The first-to-second-layer interconnections weights are denoted v_{jk}, and the second-to-third-layer interconnection weights are denoted by w_{ij}. The threshold offsets are denoted by θ_{v_j}, θ_{w_i}.

Many different activation functions $\sigma(\cdot)$ are in common use. In this work, it is required that $\sigma(\cdot)$ is smooth enough so that at least its first derivative exists. Suitable choices include the sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (2)$$

the hyperbolic tangent

$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3)$$

and other logistic-curve-type functions.

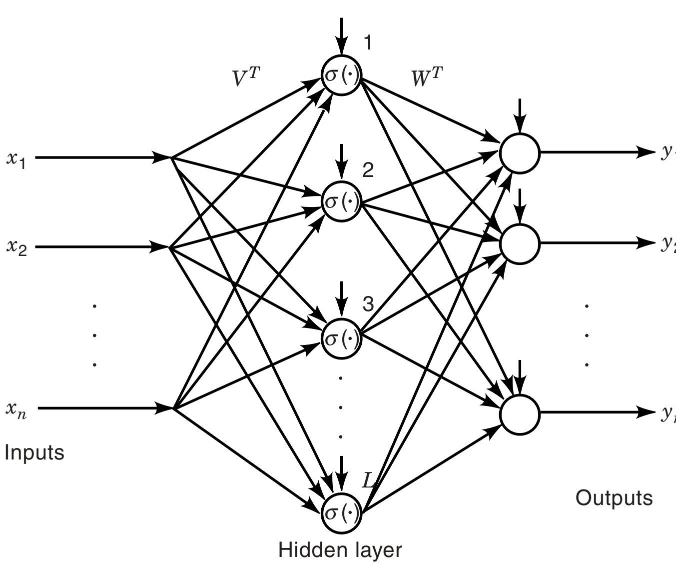

Figure 1. Two-layer feedforward neural network.

By collecting all the NN weights v_{jk}, w_{ij} into matrices of weights V^T, W^T, we can write the NN equation in terms of vectors as

$$y = W^T\sigma(V^Tx) \quad (4)$$

The thresholds are included as the first columns of the weight matrices; to accommodate this, the vectors x and $\sigma(\cdot)$ need to be augmented by placing a 1 as their first element (e.g., $x \equiv [1\,x_1\,x_2\ \cdots\ x_n]^T$). In this equation, to represent Eq. (1), we have sufficient generality if $\sigma(\cdot)$ is taken as a diagonal function from $\Re^L$ to $\Re^L$, that is $\sigma(z) = \mathrm{diag}\{\sigma(z_j)\}$ for a vector $z = [z_1\,z_2\ \cdots\ z_L]^T \in \Re^L$.

Universal Function Approximation Property. NNs satisfy many important properties. A main property of concern for feedback control purposes is the universal function approximation property (6). Let $f(x)$ be a general smooth function from $\Re^n$ to $\Re^m$. Then, it can be shown that, as long as x is restricted to a compact set $\mathcal{S}$ of $\Re^n$, there exist weights and thresholds such that we have

$$f(x) = W^T\sigma(V^Tx) + \epsilon \quad (5)$$

for some number of hidden layer neurons L. This holds for a large class of activation functions, including those just mentioned. This equation indicates that an NN can approximate any smooth function on a compact set. The value ϵ is called the NN functional approximation error, and it generally decreases as the net size L increases. In fact, for any choice of a positive number ϵ_N, we can find a feedforward NN such that $\epsilon < \epsilon_N$ for all x in $\mathcal{S}$. This means that an NN can be selected to approximate $f(x)$ to any desired accuracy ϵ_N.

The ideal NN weights in matrices W, V that are needed to best approximate a given nonlinear function $f(x)$ are difficult to determine. In fact, they may not even be unique. However, all we need to know for controls purposes is that, for a specified value of ϵ_N, some ideal approximating NN weights exist. Then, an estimate of $f(x)$ can be given by

$$\hat{f}(x) = \hat{W}^T\sigma(\hat{V}^Tx) \quad (6)$$

where $\hat{w}$ and $\hat{V}$ are estimates of the ideal NN weights that are provided by some on-line weight-tuning algorithms, which will be detailed subsequently.

The assumption that there exist ideal weights such that the approximation property holds is very much like various similar assumptions in adaptive control (7,8), including Erzberger's assumptions and linearity in the parameters. The very important difference is that in the NN case, the approximation property always holds, whereas in adaptive control such assumptions often do not hold in practice, and so they imply restrictions on the form of the systems that can be controlled.

Weight-Tuning Algorithms. So that the NN can learn and adapt to its environment, the weights should be continuously updated on-line. Many types of NN weight-tuning algorithms are used, usually based on some sort of gradient algorithm. Tuning algorithms may either be given in continuous time or in discrete time, where the weights are updated only at discrete time points (e.g., the delta rule). Discrete-time tuning is useful in digital control applications of neural networks.

A common weight-tuning algorithm is the gradient algorithm based on the backpropagated error (9), where the NN is trained to match specified exemplar pairs (x_d, y_d), with x_d the ideal NN input that yields the desired NN output y_d. The discrete-time version of the backpropagation algorithm for the two-layer NN is given by

$$\begin{aligned} \hat{W}_{k+1} &= \hat{W}_k + F\sigma(\hat{V}_k^T x_{\mathrm{d}})E_k^T \\ \hat{V}_{k+1} &= \hat{V}_k + Gx_{\mathrm{d}}(\hat{\sigma}_k'^T \hat{W}_k E_k)^T \end{aligned} \tag{7}$$

where k is the discrete time index and F, G are positive definite design parameter matrices governing the speed of convergence of the algorithm. The hidden-layer output gradient or jacobian may be explicitly computed; for the sigmoid activation functions, for instance, it is

$$\hat{\sigma}' \equiv \mathrm{diag}\{\sigma(\hat{V}^T x_{\mathrm{d}})\}\big[I - \mathrm{diag}\{\sigma(\hat{V}^T x_{\mathrm{d}})\}\big] \tag{8}$$

where diag$\{v\}$ means a diagonal matrix whose diagonal elements are the components of the vector v. The error E_k that is backpropagated is selected as the desired NN output minus the actual NN output $E_k = y_d - y_k$. Backprop tuning is accomplished off-line and requires specified training data pairs (x_d, y_d), so it is a supervised training scheme.

The continuous-time version of the backpropagation algorithm for the two-layer NN is given by

$$\begin{aligned} \dot{\hat{W}} &= F\sigma(\hat{V}^T x_{\mathrm{d}})E^T \\ \dot{\hat{V}} &= Gx_{\mathrm{d}}(\hat{\sigma}'^T \hat{W} E)^T \end{aligned} \tag{9}$$

A simplified NN weight-tuning scheme is the Hebbian algorithm, a continuous-time version of which is

$$\begin{aligned} \dot{\hat{W}} &= F[\sigma(\hat{V}^T x)]E^T \\ \dot{\hat{V}} &= Gx[\sigma(\hat{V})^T x)]^T \end{aligned} \tag{10}$$

Thus, in Hebbian tuning, no jacobian need be computed; instead, the weights in each layer are updated based on the outer product of the input and output signals of that layer.

Functional-Link Basis Neural Networks. If the first-layer weights and thresholds V in Eq. (4) are fixed and only the second-layer weights W are tuned, then the NN has only one-layer of tunable weights. Such a one-layer NN is described by

$$y = W^T \phi(x) \tag{11}$$

where $x \in \Re^n$, $y \in \Re^m$. Now, $\phi(\cdot)$ is not diagonal, but it is a general function from $\Re^n$ to $\Re^L$. This is called a functional-link neural net (FLNN) (10). In this case, the NN approximation property does not generally hold. However, a one-layer NN can still approximate functions as long as the activation functions $\phi(\cdot)$ are selected as a basis, which must satisfy the following two requirements on a compact, simply connected set $\mathscr{S}$ of $\Re^n$:

1. A constant function on $\mathscr{S}$ can be expressed as Eq. (11) for a finite number L of hidden-layer neurons.
2. The functional range of Eq. (11) is dense in the space of continuous functions from $\mathscr{S}$ to $\Re^m$ for countable L.

Some special FLNN are now discussed.

Gaussian or Radial Basis Function Networks. An NN activation function often used is the Gaussian or radial basis function (RBF) (11) given as

$$\sigma(x) = e^{-x^2/2v} \tag{12}$$

when x is a scalar, with variance v. An RBF NN can be written as Eq. (4), but it has an advantage over the usual sigmoid NN in that it is standard in probability theory, Kalman filtering, and elsewhere to consider n-dimensional Gaussian functions written as

$$\sigma(x) = e^{-\frac{1}{2}x^T P^{-1} x} \tag{13}$$

with $x \in \Re^n$. If the covariance matrix is diagonal so that $P = \mathrm{diag}\{p_i\}$, this becomes separable and may be decomposed into components as

$$\sigma(x) = e^{-\frac{1}{2}\sum_{i=1}^n x_i^2/p_i} = \prod_{i=1}^n e^{-\frac{1}{2}x_i^2/p_i} \tag{14}$$

This allows us to visualize the hidden-layer neurons as having n-dimensional activation functions, as in Fig. 2.

Having in mind the insertion of Eq. (14) into Eq. (1), or equivalently Eq. (4), we can make the following observations. The first-layer thresholds θ_{vj} of the RBF NN are n-dimensional vectors corresponding to the mean values of the Gaussian functions, which serve to shift the functions in the $\Re^n$ plane. The first-layer weights in V^T are scaling factors that serve to scale the width or variance of the Gaussians. These are both usually selected in designing the RBF NN and left fixed; only the output-layer weights W^T are generally tuned. Therefore, the RBF NN is a special sort of FLNN Eq. (11).

Figure 2 shows two-dimensional (2-D) separable Gaussians with thresholds selected on a evenly spaced grid. To form an RBF NN that approximates functions (see subsequent paragraph) over the region $\{-1 < x_1 \leq 1, -1 < x_2 \leq 1\}$, we may choose $5 \times 5 = 25$ hidden-layer neurons, corresponding to five cells along x_1 and five along x_2. Nine of these neurons will have 2-D Gaussian activation functions, whereas those along the boundary require the illustrated "one-sided" activation functions.

The importance of RBF NNs (11) is that they show how to select the activation functions and number of hidden-layer neurons for specific NN applications, including approximation, while also giving insight on the information stored in the NN.

Fuzzy Neural Networks. There are many ways to bring together NNs and fuzzy logic (FL) systems (12) (see FUZZY LOGIC SYSTEMS), including architectures having both NN and FL components (e.g., using FL systems to initialize NN weights or NN to adapt FL membership functions). However, one point of view is to consider FL systems as a special class of structured NN.

It can be shown that fuzzy logic systems using product inferencing and weighted defuzzification are equivalent to special sorts of NN with suitably chosen separable activation

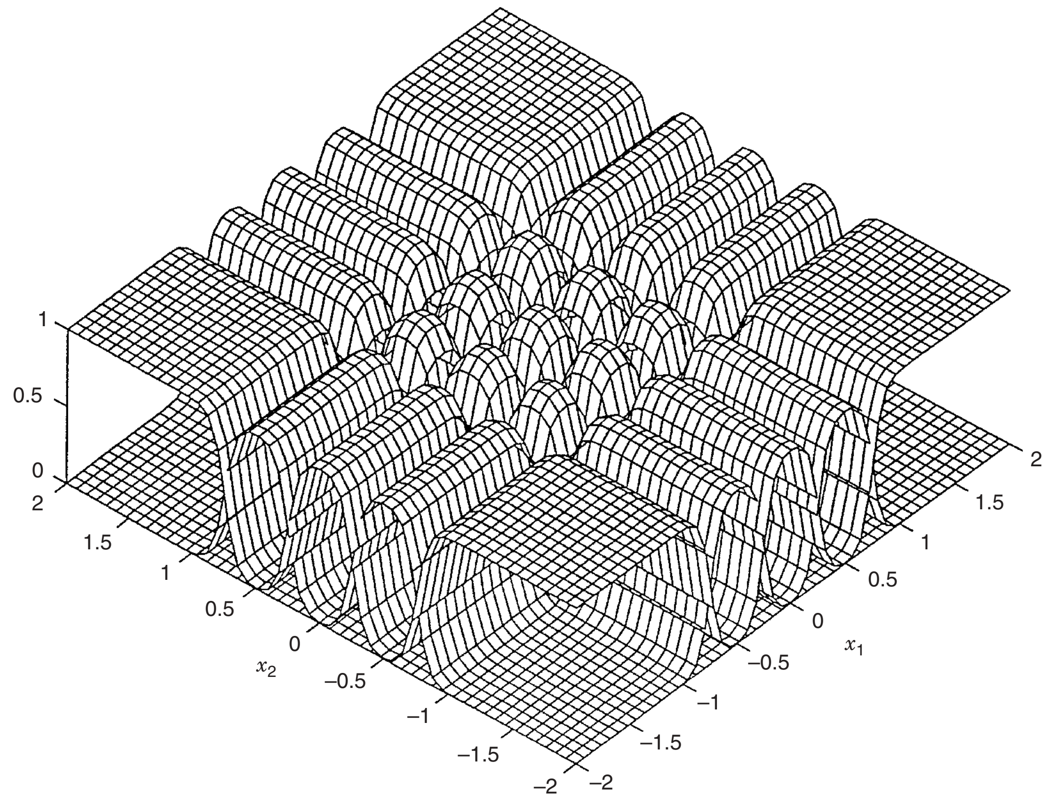

Figure 2. Two-dimensional separable Gaussian functions for an RBF NN.

functions. In fact, dividing Eq. (1) by Eq. (4) with thresholds $\theta_{wi} = 0$ is identical to the output equation for this class of FL systems. In FL systems,

$$X_j(x_k) = \sigma(v_{jk}x_k + \theta_{v_{jk}}) \tag{15}$$

are the membership functions along component x_k, shifted by $\theta_{v_{jk}}$ and scaled by v_{jk}. The n-dimensional membership functions are composed using multiplication of scalar membership functions as in Eq. (14). The output-layer weights w_{ij} are known as the control representative values in FL systems.

The RBF NN in Fig. 2 is equivalent to a fuzzy system with Gaussian membership functions along x_1 and x_2. FL systems are also very closely related to the Cerebellar Model Articulation Controller (CMAC) NN (13). A CMAC NN has separable activation functions generally composed of splines. The activation functions of a 2-D CMAC composed of first-order splines (e.g., triangle functions) are shown in Fig. 3; it is equivalent to a 2-D FL system with triangle membership functions. The activation functions of a CMAC NN are called receptive field functions in analogy with the optical receptor fields of the eye.

In adaptive FL systems, we may adapt the control representative values W and/or the membership function parameters V. If V is not adapted, then the first-layer weights and thresholds are fixed so that the membership functions are not tuned. These FL systems are therefore FLNN, and the membership functions must be chosen as a basis on some compact set. If both W and V are adapted, the FL systems possess the universal approximation property Eq. (5).

From this discussion, it is evident that all the NN control techniques to be discussed in this article also apply for fuzzy logic control (14). Note specifically that backpropagation tuning can be used to adapt the FL parameters. See also FUZZY LOGIC CONTROL.

Dynamic/Recurrent Neural Networks. If the NN has its own dynamics, it is said to be *dynamic* or *recurrent*. An important recurrent NN is the Hopfield net used in classification applications. The continuous-time Hopfield net is described by the ordinary differential equation

$$\tau_i\dot{x}_i = -x_i + \sum_{j=1}^{n} w_{ij}\sigma(x_j) + u_i \tag{16}$$

with output equation

$$y_i = \sum_{j=1}^{n} w_{ij}\sigma(x_j) \tag{17}$$

This is a dynamical system of special form that contains the weights w_{ij} as adjustable parameters and positive time constants τ_i. The offsets u_i play the role of the control input term in system theory. In traditional Hopfield NN, the term *input pattern* refers to the initial state components $x_i(0)$.

In the discrete-time case, the NN is described by the difference equation

$$x_i(k+1) = p_i x_i(k) + \sum_{j=1}^{n} w_{ij}\sigma_j[x_j(k)] + u_i(k) \tag{18}$$

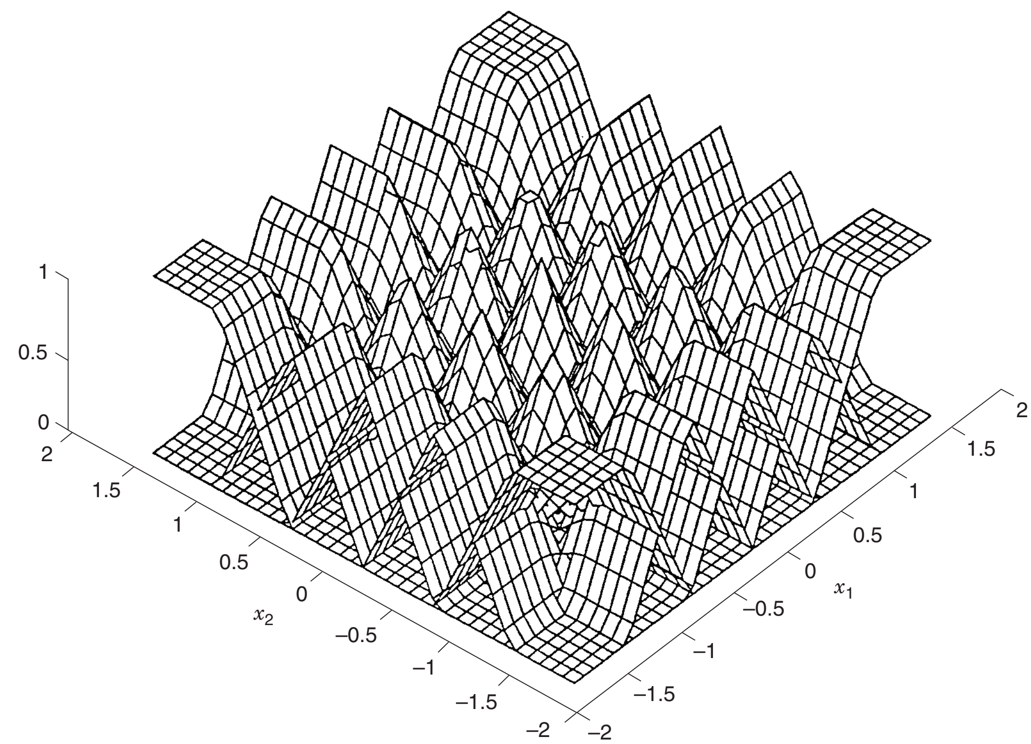

Figure 3. Receptive field functions for a two-dimensional CMAC NN with first-order splines, showing similarity to fuzzy logic system.

with $p_i < 1$. This is a discrete-time dynamical system with time index k.

Feedback Control and Early Design Using Neural Networks

Feedback control involves the measurement of output signals from a dynamical system or plant, and the use of the difference between the measured values and certain prescribed desired values to compute system inputs that cause the measured values to follow or track the desired values. In feedback control design, it is crucial to guarantee both tracking performance and internal stability or boundedness of all variables. Failure to do so can cause serious problems in the closed-loop system, including instability and unboundedness of signals that can result in system failure or destruction.

There is a large literature on NN for feedback control of unknown plants. Initially, design and analysis techniques were ad hoc, with no repeatable design algorithms or proofs of stability and guaranteed performance. Many NN design techniques mimicked adaptive control approaches, where rigorous analysis results were available (7,8). In these early techniques, there were serious unanswered questions. Because we did not know how to initialize the NN weights to provide closed-loop stability, most approaches required an off-line learning phase, where the NN weights were tuned using measurements of system inputs and outputs in a preliminary phase before the controller was allowed to provide system inputs. Such an open-loop phase has serious detrimental repercussions for industrial and mechanical systems where control is usually required immediately. Recent results show how to combine off-learning and a priori information with dynamic on-line learning in real time to improve adaptibility of the controller (see NEUROCONTROLLERS).

Most of the early approaches used standard backpropagation weight tuning because rigorous derivations of tuning algorithms suitable for feedback control purposes were not available. (In fact, it has recently been shown that backpropagation must be modified for closed-loop control purposes.) Moreover, in early applications of direct closed-loop control, the gradients (jacobians) needed for backpropagation depended on the unknown system and/or satisfied their own differential equations; this made them impossible or very difficult to compute. Thus, although rigorously applied in open-loop identification, NNs had not been fully developed for direct closed-loop control. The most serious problem was that rigorous stability proofs and guarantees of closed-loop performance were not available, so that the performance of these controllers on actual industrial or mechanical systems was open to serious question. Most research papers were supported by computer simulation results, which often indicated good performance, but only for the conditions and systems tested.

Narendra (3) and others (1,2,4) have paved the way for rigorous NN controls applications by studying the dynamical behavior of NNs in closed-loop systems, including computation of the gradients needed for backprop tuning. Several groups have done rigorous analysis of NN controllers using a variety of techniques. The Bibliography lists some work by Sanner and Slotine (11), Polycarpou and Ioannou (15,16), Rovithakis and Christodoulou (17), Sadegh (10), Chen and Khalil (18), Chen and Liu (19), and the present author with others (20,21).

Several NN feedback control topologies are illustrated in Fig. 4, some of which are derived from standard topologies in adaptive control (8). There are basically two sorts of feedback

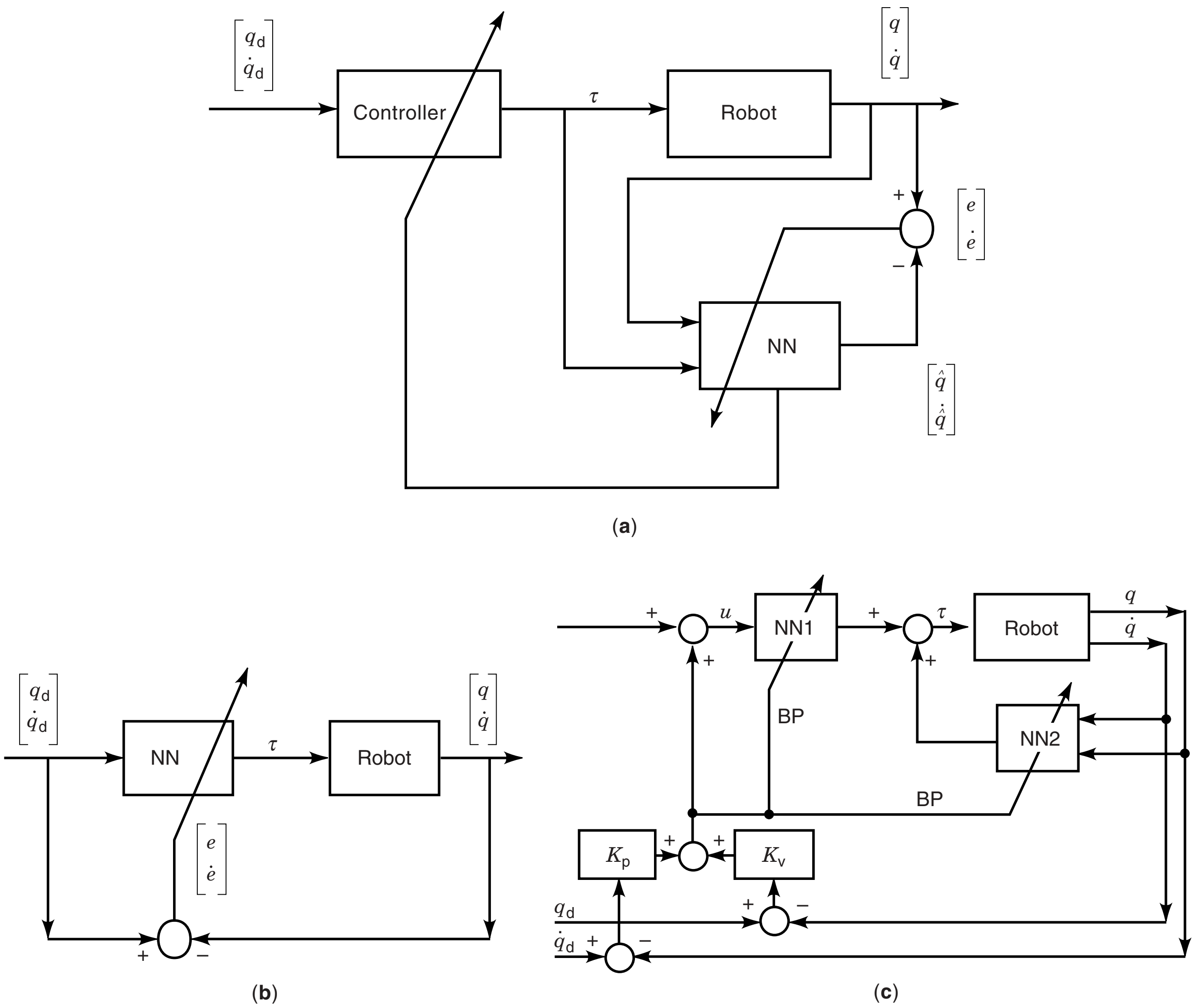

Figure 4. Neural net feedback controller topologies. (a) Indirect scheme. (b) Inverse system control. (c) Series control.

control topologies—indirect techniques and direct techniques. In indirect NN control, there are two functions; in an identifier block, the NN is tuned to learn the dynamics of the unknown plant, and the controller block then uses this information to control the plant. Direct control is more efficient and involves the NN directly tuning the parameters of an adjustable controller.

TRACKING CONTROL USING STATIC NEURAL NETWORKS

The chief common characteristic of early NN control design techniques was that rigorous design techniques and stability proofs were not offered. In keeping with the philosophy of those working in control system theory since Maxwell, Lyapunov, A. N. Whitehead, and other early researchers, to provide guarantees of closed-loop performance, it is necessary to begin with the knowledge available about the system being controlled. Many industrial mechanical systems, as well as automobiles, aircraft, and spacecraft, have dynamics in the Lagrangian form, which are exemplified by the class of rigid robot systems. Therefore, in this article the Lagrangian robot dynamics will be considered (21). The NN control techniques presented may also be applied to other unknown systems including certain important classes of nonlinear systems (22).

In this section is discussed feedback tracking control design using static feedforward NN. This amounts to the design of what is called in control system terminology the *tracking control loop* and in computer science terminology the *action-generating loop.* In subsequent sections are discussed feedback control using dynamic NNs and higher-level architectures such as reinforcement learning and adaptive critics.

Robot Arm Dynamics and Feedback Control

The dynamics of rigid Lagrangian systems, including robot arms, have some important physical and structural properties that make it very natural to use NN in their control. These properties should be taken into account in the design of any controller. In fact, they provide the background for a rigorous design algorithm for NN controllers.

Robot Dynamics and Properties. The dynamics of an n-link rigid (i.e., no flexible links or high-frequency joint/motor dynamics) robot manipulator may be expressed in the Lagrange form (23)

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + G(q) + F(\dot{q}) + \tau_d = \tau \tag{19}$$

with $q(t) \in \Re^n$ the joint variable vector, whose entries are the robot arm joint angles or link extensions. $M(q)$ is the inertia matrix, $V_m(q, \dot{q})$ is the coriolis/centripetal matrix, $G(q)$ is the gravity vector, and $F(\dot{q})$ is the friction. Bounded unknown disturbances (including, for example, unstructured unmodeled dynamics) are denoted by τ_d, and the control input torque is $\tau(t)$. The robot dynamics have the following standard properties:

Property 1. $M(q)$ is a positive definite symmetric matrix bounded by $m_1 I < M(q) < m_2 I$, with m_1, m_2 positive constants.

Property 2. The norm of the matrix $V_m(q, \dot{q})$ is bounded by $v_b(q)\|\dot{q}\|$, for some function $v_b(q)$.

Property 3. The matrix $\dot{M} - 2V_m$ is skew-symmetric. This is equivalent to the fact that the internal forces do no work.

Property 4. The unknown disturbance satisfies $\|\rho_d\| < b_d$, with b_d a positive constant.

Tracking a Desired Trajectory and the Error Dynamics. An important application in robot arm control is for the manipulator to follow a prescribed trajectory, a problem that appears in spray painting, surface finishing and grinding, and so on. Given a desired arm trajectory $q_d(t) \in \Re^n$, the tracking error is

$$e(t) = q_d(t) - q(t) \tag{20}$$

It is typical in robotics to define a so-called filtered tracking error as

$$r = \dot{e} + \Lambda e \tag{21}$$

where Λ is a symmetric positive definite design parameter matrix, usually selected diagonal. The objective in tracking controller design is to design a control system topology that keeps $r(t)$, and hence the tracking error $e(t)$, small.

Differentiating $r(t)$ and using Eq. (19), the arm dynamics may be written in terms of the filtered tracking error as

$$M\dot{r} = -V_m r - \tau + f + \tau_d \tag{22}$$

where the important nonlinear robot function is

$$f(x) = M(q)(\ddot{q}_d + \Lambda\dot{e}) + V_m(q,\dot{q})(\dot{q}_d + \Lambda e) + G(q) + F(\dot{q}) \tag{23}$$

The vector x required to compute $f(x)$ can be defined, for instance, as

$$x \equiv [e^T\ \dot{e}^T\ q_d^T\ \dot{q}_d^T\ \ddot{q}_d^T]^T \tag{24}$$

which can be measured.

Function $f(x)$ contains all the robot parameters such as payload mass, link masses and lengths, and friction coefficients. These quantities are often imperfectly known and difficult to determine. This is especially true of the payload mass, which varies in real-time applications, and the friction terms $F(\dot{q})$, which can be extremely complicated functions that vary as the joints heat up during use.

Robot Controller and the Error System. In applications, the nonlinear robot function $f(x)$ is at least partially unknown. Therefore, a suitable control input for trajectory following is given by the computed-torque-like control

$$\tau = \hat{f} + K_v r - v \tag{25}$$

with $K_v = K_v^T > 0$ a gain matrix, generally chosen diagonal, and $\hat{f}(x)$ an estimate of the robot function $f(x)$ that is provided by some means. The robustifying signal $v(t)$ is needed to compensate for unmodeled unstructured disturbances. Using this control, the closed-loop system becomes

$$M\dot{r} = -(K_v + V_m)r + \tilde{f} + \tau_d + v \tag{26}$$

This is an *error system* wherein the filtered tracking error is driven by the functional estimation error $\tilde{f} = f - \hat{f}$. The error system is of supreme importance in feedback control system design because its structure allows the study of means to make the tracking error $r(t)$ small, facilitating both the selection of good controller topologies and rigorous proofs of closed-loop performance.

In computing the control signal, the estimate $\hat{f}$ can be provided by several techniques, including adaptive control (7,8) or neural networks. The auxiliary control signal $v(t)$ can be selected by several techniques, including sliding-mode methods and others under the general aegis of robust control methods.

Neural Net Feedback Tracking Controller

Even though the general control structure is now pinned down in Eq. (25), there is no guarantee that the control τ will make the tracking error small. Thus, the control design problem is to specify a method of selecting the gains K_v, the estimate $\hat{f}$, and the robustifying signal $v(t)$ so that both the error $r(t)$ and the control signals are bounded. It is important to note that the latter conclusion hinges on showing that the estimate $\hat{f}(x)$ is bounded. Moreover, for good performance, the bounds on $r(t)$ should be, in some sense, small enough.

Neural Net Multiloop Feedback Control Topology. The control τ incorporates a proportional-plus-derivative (PD) outer loop in the term $K_v r = K_v(\dot{e} + \Lambda e)$. An NN will be used to provide the estimate $\hat{f}$ for the unknown robot function $f(x)$. The NN approximation property Eq. (6) assures us that there always exists an NN that can accomplish this within a given accuracy ϵ_N. The basic structure of this NN controller appears in Fig. 5, where $\underline{e} \equiv [e^T\ \dot{e}^T]^T$, $\underline{q} \equiv [q^T\ \dot{q}^T]^T$. The neural network that provides the estimate for $f(x)$ appears in an inner control loop, and there is an outer tracking loop provided by the PD term $K_v r$. This multiloop intelligent control structure is derived naturally from robot control notions and is not ad hoc. In control theory terminology, it is a feedback linearization controller (24). As such, it is immune to philosophical deliberations concerning suitable NN control topologies including the common discussions on feedforward vs. feedback, direct vs. indirect, and so on. It is to be noted that the static feedfor-

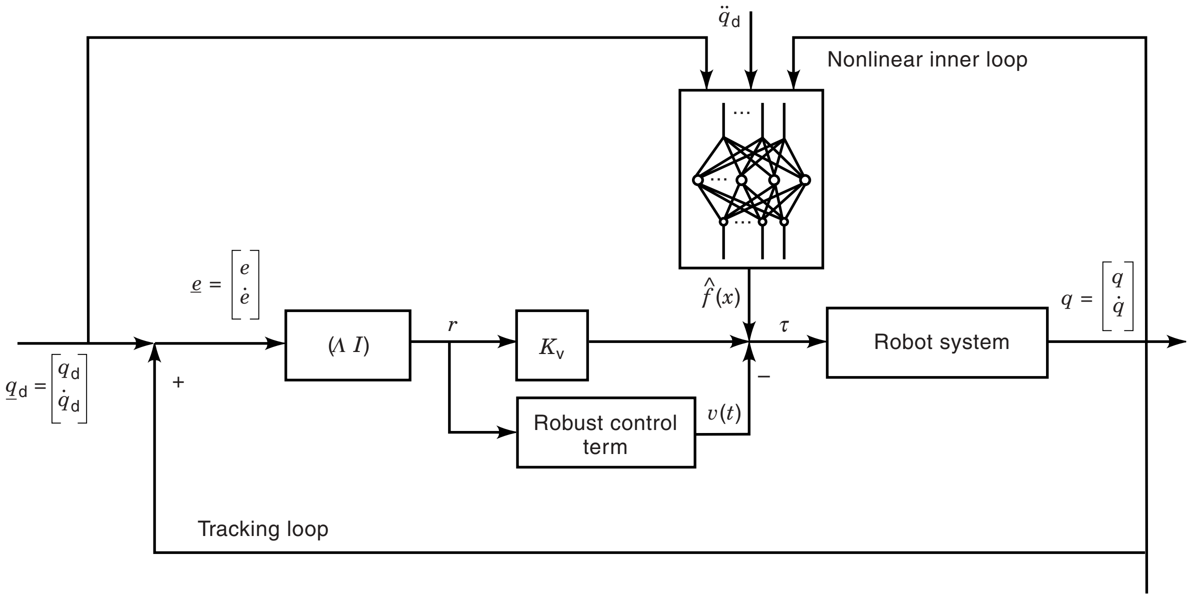

Figure 5. Neural net controller for rigid robot arms, showing inner nonlinear neural network loop and outer tracking loop.

ward NN in this diagram is turned into a dynamic NN by closing a feedback loop around it [c.f. Ref. (3)].

NN Weight Tuning for Stability and Robustness. Unfortunately, there is not yet any clue on how to tune the NN weights. The error dynamics Eq. (26) can be used to focus on selecting NN tuning algorithms, the signal $v(t)$, and the control gains K_v that guarantee the stability of the filtered tracking error $r(t)$. Then, because Eq. (21), with the input considered as $r(t)$ and the output as $e(t)$ describes a stable system, standard techniques guarantee that $e(t)$ exhibits stable behavior.

By placing the NN approximation Eq. (6) into the error system Eq. (26), we obtain the error dynamics corresponding to Fig. 5 as

$$\begin{aligned} M\dot{r} &= -(K_v + V_m)r + W^T\sigma(V^Tx) \\ &\quad - \hat{W}^T\sigma(\hat{V}^Tx) + (\epsilon + \tau_d) + v \end{aligned} \tag{27}$$

It is noted that the error dynamics are excited by both the NN reconstruction error ϵ and the robot disturbances τ_d. Unfortunately, this equation has a very contrary form for controls design because of the presence of the tunable first-to-second-layer NN weights $\hat{V}$ within the argument of the nonlinear function $\sigma(\cdot)$. In fact, selecting tuning algorithms to stabilize this system is a nonlinear adaptive control problem because the error system is nonlinear in the adjustable parameters V.

By using a certain Taylor series expansion of the hidden-layer estimation error $\sigma(V^Tx) - \sigma(\hat{V}^Tx)$, some adaptive control-like manipulations, various robust control bounding techniques, and finally an extension of nonlinear stability proof techniques, we can show that the NN controllers described in the upcoming paragraphs are guaranteed to make the system track the desired trajectory. The proofs hinge on selecting an appropriate energy function for the closed-loop system. This is much the same as energy functions selected by Hopfield and others in showing the convergence either of dynamic NN to certain local equilibria, or the convergence of certain NN weight tuning algorithms.

In closed-loop NN feedback control a suitable energy function is the Lyapunov-like function

$$\mathcal{V} = \tfrac{1}{2}r^TM(q)r + \tfrac{1}{2}\mathrm{tr}(\tilde{W}^TF^{-1}\tilde{W}) + \tfrac{1}{2}\mathrm{tr}(\tilde{V}^TG^{-1}\tilde{V}) \tag{28}$$

with tr() the trace of a matrix (e.g., sum of its diagonal elements), $r(t)$ the filtered tracking error, and the NN weight estimation errors given by $\tilde{W} = W - \hat{W}$, $\tilde{V} = V - \hat{V}$. The first term of $\mathcal{V}(t)$ is a dynamic kinetic energy term, whereas the second and third terms can be interpreted as potential energy terms. Using the error dynamics Eq. (27), we can show that, while $\mathcal{V}(t)$ is always nonnegative, its derivative $\dot{\mathcal{V}}(t)$ is always nonpositive, so that the energy in the system is bounded. Details and the proof are discovered in Ref. 21.

Modified Unsupervised Backpropagation Tuning for NN Feedback Control. Using the Lyapunov-like proof technique just outlined, it can be proven that the NN controller described completely in Table 1 yields small tracking errors and boundedness of all signals in closed-loop.

Table 1. Design Specifications for NN Rigid Robot Controller

Control Input:

$$\tau = \hat{W}^T\sigma(\hat{V}^Tx) + K_vr - v$$

NN Weight/Threshold Tuning Algorithms:

$$\dot{\hat{W}} = F\sigma(\hat{V}^Tx)r^T - F\hat{\sigma}'\hat{V}^Txr^T - \kappa F\|r\|\hat{W}$$
$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T\hat{W}r)^T - \kappa G\|r\|\hat{V}$$

Design parameters: F, G positive definite matrices and $\kappa > 0$
Robustifying signal:

$$v(t) = -K_z(\|\hat{Z}\| + Z_M)r$$

The NN controller in Table 1 is a general model-free controller for any rigid-link arm in that the detailed model of the robot dynamics is not required because it is estimated by the NN. The rationale behind the NN controller in Table 1 follows. The first terms in the weight/threshold tuning algorithms have exactly the same structure as the backpropagation-tuning algorithm Eq. (9). However, they give an on-line real-time, unsupervised version of backprop-through-time that does not need exemplar input/output pairs for tuning. In fact, the proof shows that the signal that should be backpropagated in closed-loop NN applications is exactly the filtered error $r(t)$. Moreover, the jacobian $\hat{\sigma}'$ needed in Table 1 is easily computed in terms of known quantities [i.e., $x(t)$ and the current weights $\hat{V}$].

The last terms in the weight-tuning algorithms are Narendra's e-modification, familiar in linear adaptive control (25). However, the nonlinear tuning nature of the problem, because of the appearance of tunable weights V within the argument of $\sigma(\cdot)$, has added two additional terms, namely, the middle term in the tuning algorithm for $\hat{W}$ and the robustifying signal $v(t)$.

Further properties of the NN controller are discussed in the next subsection.

Modified Hebbian Tuning for NN Feedback Control. It can be shown using similar rigorous stability proof techniques that the controller in Fig. 5 using the simplified tuning algorithms in Table 2 has the same guaranteed performance features as the backprop-related controller in Table 1. In Table 2, the first terms in the weight-tuning laws are modified versions of the Hebbian tuning algorithm Eq. (10), which does not require the computation of a jacobian. The price for this simplification is a slight increase in the magnitude of the tracking error $r(t)$.

Discrete-Time Tuning for NN Feedback Control. Because most controllers requiring the computation of nonlinear terms are implemented using digital signal processors or microprocessors, it is important to design NN controllers with discrete-time weight update algorithms, where the weights may be tuned only at the sample times. Proposed discrete-time NN tuning algorithms for feedback control abound in the literature, but until the 1990s then were ad hoc modifications of open-loop gradient-based algorithms such as the delta rule and could not guarantee any sort of stability or tracking in closed-loop feedback controls applications.

Table 2. NN Robot Controller with Hebbian Tuning

Control Input:

$$\tau = \hat{W}^T\sigma(\hat{V}^T x) + K_v r - v$$

NN Weight/Threshold Tuning Algorithms:

$$\dot{\hat{W}} = F[\sigma(\hat{V}^T x)]r^T - \kappa F\|r\|\hat{W}$$
$$\dot{\hat{V}} = Gx[\sigma(\hat{V}^T x)]^T\|r\| - \kappa G\|r\|\hat{V}$$

Design parameters: F, G positive definite matrices and $\kappa > 0$

Robustifying signal:

$$v(t) = -K_z(\|\hat{Z}\| + Z_M)r$$

Using rigorous nonlinear stability methods based on Lyapunov techniques, exactly as in deriving the continuous-time controllers in this article, it is possible though much more involved to derive digital NN controllers. A typical digital NN controller is shown in Fig. 6, where z^{-1} represents the unit delay. Exactly as in Fig. 5, it has a multiloop structure with an inner NN loop and an outer PD tracking loop. Note that the outer loop requires current and past values of the tracking error, whereas the NN requires current and past values of the system states.

Table 3 shows typical digital NN controller weight update algorithms. The discrete-time weight tuning algorithms in the table have some features in common with open-loop tuning algorithms used in the literature. Specifically, they are a form of delta rule with the first terms very similar to a discrete-time Hebbian rule with some extra terms involving the tracking error r_k. The last terms are similar to what have been called forgetting factors in computer science and are equivalent to a discrete-time version of what is known as Narendra's e-modification in adaptive control theory. These terms are required to make the NN controller robust to unknown unmodeled dynamics by ensuring that the NN weights remain bounded. To speed up learning for NN with a large number L of hidden-layer neurons, we may modify the tuning algorithms based on a projection algorithm, which is well known in adaptive control (7).

Discussion of the NN Robot Controller

Computation of the Controller. In Table 1, any NN activation functions $\sigma(\cdot)$ with a bounded first derivative can be used as long as they have the approximation property Eq. (5). The norms are the 2-vector norm and the Frobenius matrix norm, both easily computed in terms of the sums of squares of elements. In the tuning algorithms, the hidden-layer gradient or jacobian $\hat{\sigma}'$ is easily computed in terms of measurable signals—for the sigmoid activation functions it is given by

$$\hat{\sigma}' \equiv \text{diag}\{\sigma(\hat{V}^T x)\}\left[I - \text{diag}\{\sigma(\hat{V}^T x)\}\right] \tag{29}$$

which is just Eq. (8) with the constant exemplar x_d replaced by the time function $x(t)$. In the robustifying signal, $\hat{Z} \equiv \text{diag}\{\hat{W}, \hat{V}\}$ is the matrix of all the NN weights, and Z_M is an upper bound on the ideal weights in Eq. (5), which always exists and can be selected simply as a large positive number. The robustifying gain K_Z should be selected large. Note that, as in well-designed adaptive controllers, no acceleration measurements are required by the NN controller.

Bounded Tracking Errors and NN Weights. The NN controller in Table 1 guarantees that the tracking error is bounded by

$$\|r\| \leq \frac{\epsilon_N + b_d + \kappa C}{K_{v_{\min}}} \tag{30}$$

where ϵ_N is the NN functional reconstruction error bound, b_d is the robot disturbance term bound, and C represents other constant terms. The divisor $K_{v_{\min}}$ is the smallest PD gain. The form of this bound is extremely important; it shows that the tracking error increases as the disturbances or NN reconstruction errors increase, but that arbitrarily small tracking errors can be achieved by using large enough control gains K_v. The controller also guarantees boundedness of the NN weights $\hat{W}$, $\hat{V}$, which in turn ensures that the control τ is

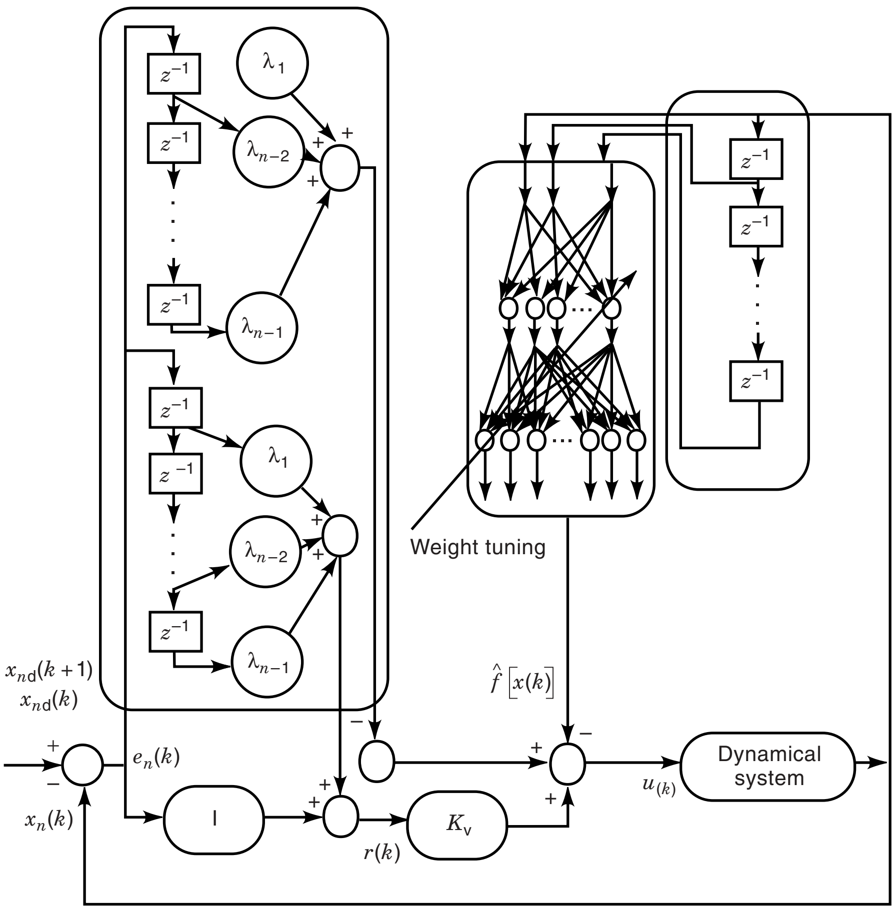

Figure 6. Digital neural net controller, showing delayed terms needed for tuning and for outer tracking loop.

bounded. Similar remarks hold for the NN controllers using Hebbian and discrete-time weight tuning.

It is important to note that removing the NN inner loop in Fig. 5 results in simply a PD controller. Although it is known that PD control can guarantee bounded tracking errors if the gains are large enough, there may be fundamental errors that cannot be removed (e.g., steady state errors and tracking errors that cannot be made arbitrarily small). Moreover, large control signals may be needed in simple PD control. On the other hand, including the NN loop allows us to derive the tighter bound Eq. (30) that can be made as small as desired.

On-Line NN Learning Feature and NN Weight Initialization. A major advantage of this NN controller is that no off-line weight tuning is needed. In fact, the NN weights are initialized at zero, then the NN learns on-line in real time. This on-line learning feature is due to the multiloop structure of the controller, for the PD outer tracking loop keeps the system stable until the NN adequately learns the function $f(x)$. That is, the controller effectively works in unsupervised mode.

Table 3. Digital NN Robot Controller Weight Updates

$$\hat{W}_{k+1} = \hat{W}_k + \alpha_1 \hat{\sigma}_k r_{k+1}^T - \kappa \|I - \alpha_1 \hat{\sigma}_k \hat{\sigma}_k^T\| \hat{W}_k$$

$$\hat{V}_{k+1} = \hat{V}_k - \alpha_2 x_k [\hat{V}_k^T x_k + K_v r_k]^T - \kappa \|I - \alpha_2 x_k x_k^T\| \hat{V}_k$$

where $\hat{\sigma}_k \equiv \sigma(\hat{V}_k^T x_k)$ and $\kappa > 0$

In most NN controllers in the literature, there is a major problem in deciding how to initialize the NN weights to give initial closed-loop stability. This leads to the need for extensive off-line training schemes to estimate the plant dynamics. Some recent results are now showing how to combine off-line learning and a priori information with on-line dynamic learning.

Advantages of NN Controllers Over Adaptive Controllers. The NN controller is no more difficult to implement on actual systems than modern adaptive control algorithms (7,8). It also embodies some notions from robust control in the signal $v(t)$. However, in addition to the advantages just discussed, NN control offers two specific advantages over adaptive control. First, to implement standard robot adaptive controllers, it is necessary to perform extensive system modeling and preliminary analysis to compute a so-called regression matrix. [This problem is avoided in Ref. (26).] The complications arising from this requirement are well known to practicing engineers. By contrast, the NN controller in Fig. 5 works for any rigid robot arm without any need to compute a regression matrix or perform any preliminary analysis whatsoever. Thus, it is a model-free controller for nonlinear rigid robot manipulators. The model-free property of NN controllers is a consequence of the NN universal approximation property.

Second, in adaptive control we require that the unknown functions [e.g., $f(x)$ in Eq. (23)] be linear in an unknown parameter vector. This is not required in the NN controller, which in fact is nonlinear in the tunable first-layer weights

V. The linear-in-the-parameters assumption does not hold for all systems and is actually a serious restriction on the types of systems that can be controller by adaptive control techniques. The NN approximation property holds for practical systems if a proper control engineering formulation is used to derive the error dynamics.

NN Complexity and Number of Hidden-Layer Neurons. The size of the NN required should be addressed. A larger net (e.g., a larger number L of hidden-layer neurons) is more difficult to implement because one integrator is needed for each NN weight. On the other hand, larger values for L will yield a smaller functional reconstruction error bound ϵ_N. According to the bound Eq. (30), this will result in smaller tracking errors. However, the form of that bound reveals that the tracking error can always be made smaller by increasing the PD gains K_v. That is, there is a design tradeoff between tracking performance and NN complexity. Use of a smaller NN can to an extent be offset by using larger PD gains, but larger NN allow smaller PD gains, presumably leading to reduced control signal magnitudes.

Partitioned Neural Networks and Preprocessing of NN Inputs. A major advantage of the NN approach is that it allows us to partition the controller in terms of partitioned NN or neural subnets. This (1) simplifies the design, (2) gives added controller structure, and (3) makes for faster weight-tuning algorithms.

Partitioned Neural Nets. The unknown nonlinear robot function Eq. (23) is

$$f(x) = M(q)\zeta_1(t) + V_m(q,\dot{q})\zeta_2(t) + G(q) + F(\dot{q}) \tag{31}$$

where $\zeta_1(t) \equiv \ddot{q}_d + \Lambda\dot{e}$, $\zeta_2(t) \equiv \dot{q}_d + \Lambda e$. Taking the four terms in $f(x)$ one at a time, use separate NN to reconstruct each term so that

$$\begin{aligned} M(q)\zeta_1(t) &= W_M^T \sigma_M(V_M^T x_M) \\ V_m(q,\dot{q})\zeta_2(t) &= W_V^T \sigma_V(V_V^T x_V) \\ G(q) &= W_G^T \sigma_G(V_G^T x_G) \\ F(\dot{q}) &= W_F^T \sigma_F(V_F^T x_F) \end{aligned} \tag{32}$$

This procedure results in four neural subnets, one for estimating the inertia terms, one for the coriolis/centripetal terms, one for gravity, and one for friction. This is called a structured or partitioned NN, as shown in Fig. 7. It is direct to show that the individual partitioned NNs can be separately tuned, making for a faster weight update procedure. That is, each of the neural subnets can be tuned individually using the rules in Table 1.

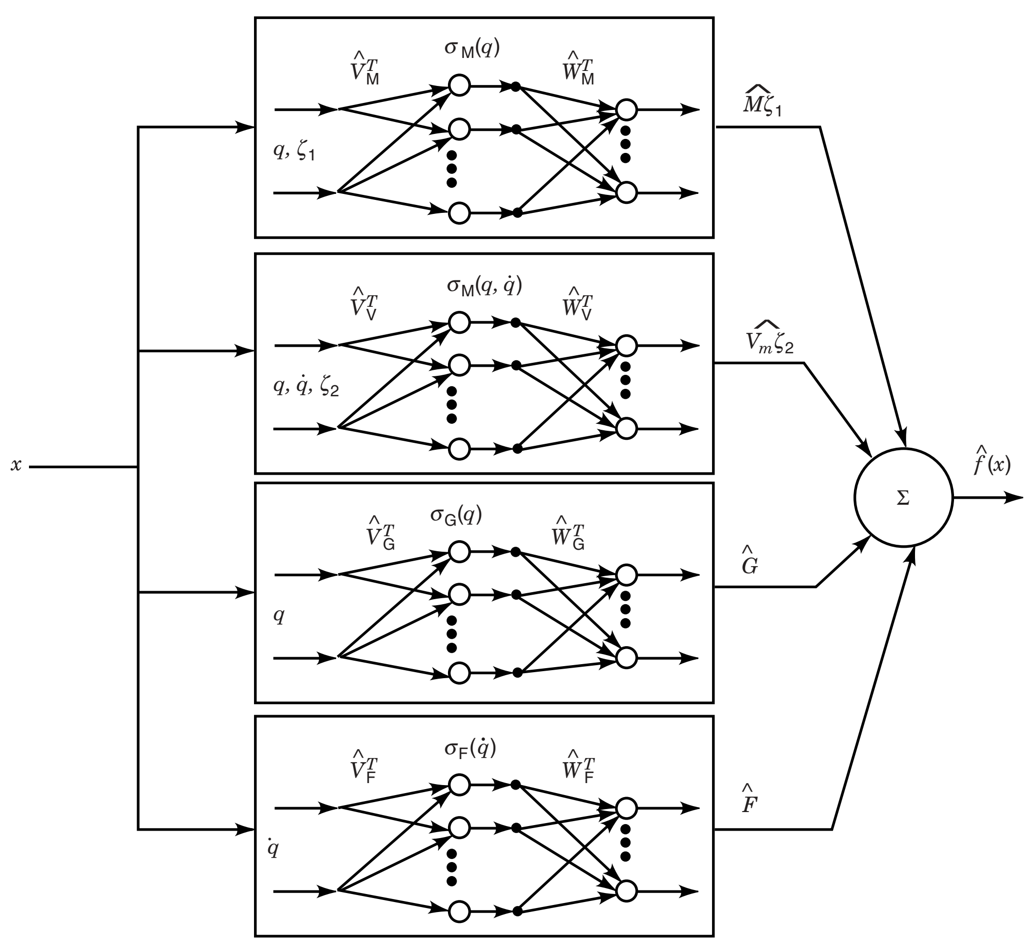

Figure 7. Partitioned neural net, which has more structure and is faster to tune than unpartitioned neural network.

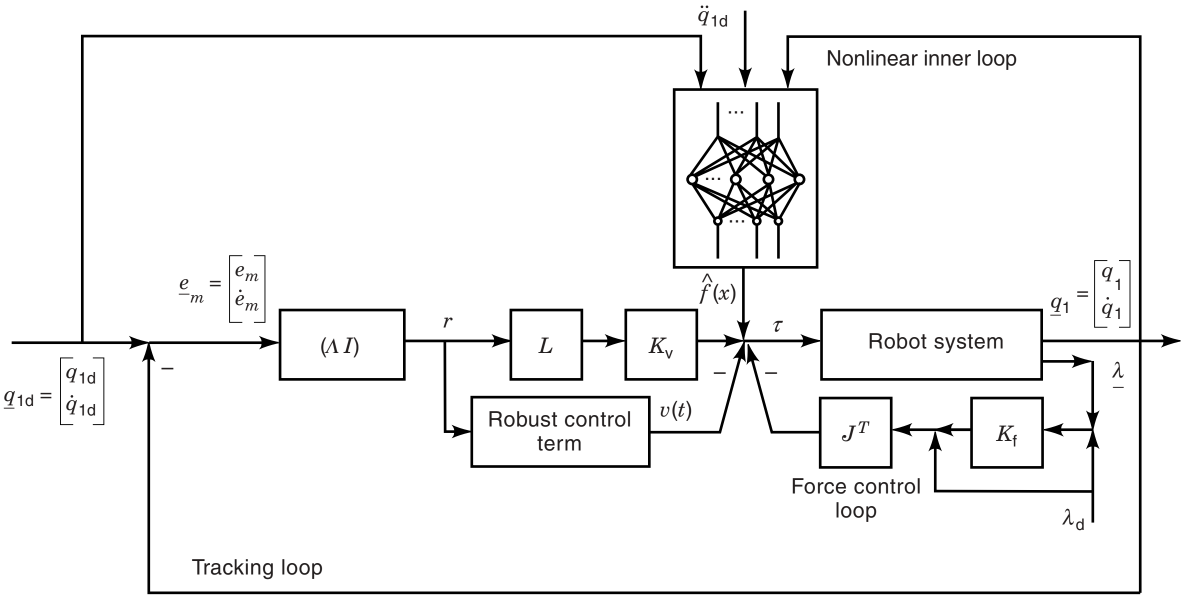

Figure 8. NN force/position controller, showing additional inner force-control loop.

An advantage of this structured NN is that if some terms in the robot dynamics are well known [e.g., inertia matrix $M(q)$ and gravity $G(q)$], then their NNs can be replaced by equations that compute them. NNs can be used to reconstruct only the unknown terms or those too complicated to compute, which will probably include the friction $F(\dot{q})$ and the coriolis/centripetal terms $V_m(q, \dot{q})$.

Preprocessing of Neural Net Inputs. The selection of a suitable $x(t)$ for computation remains to be addressed; some preprocessing of signals yields a more advantageous choice than Eq. (24) because it can explicitly introduce some of the nonlinearities inherent to robot arm dynamics. This reduces the burden of expectation on the NN and, in fact, also reduces the reconstruction error ϵ in Eq. (5).

Let an n-link robot have n_r revolute joints with joint variables q_r and n_p prismatic joints with joint variables q_p, so that $n = n_r + n_p$. Because the only occurrences of the revolute joint variables are as sines and cosines, transform $q = [q_r^T\ q_p^T]^T$ by preprocessing to $[\cos(q_r)^T\ \sin(q_r)^T\ q_p^T]^T$ to be used as arguments for the basis functions. Then the NN input vector x can be taken as

$$x = \begin{bmatrix} \zeta_1^T & \zeta_2^T & \cos(q_r)^T & \sin(q_r)^T & q_p^T & \dot{q}^T & \mathrm{sgn}(\dot{q})^T \end{bmatrix}^T \quad (33)$$

where the signum function is needed in the friction terms.

Inner Feedback Loops: Applications and Extensions

An NN controller for rigid-link robot manipulators was given in Fig. 5, with weight-tuning algorithms given in Tables 1–3. Actual industrial or military mechanical systems may have additional dynamical complications such as vibratory modes, high-frequency electrical actuator dynamics, or compliant couplings or gears. Practical systems may also have additional performance requirements such as requirements to exert specified forces or torques as well as perform position trajectory following (e.g., robotic grinding or milling). In such cases, the NN controller in Fig. 5 still works if it is modified to include additional inner feedback loops to deal with the additional plant or performance complexities.

Force Control with Neural Nets. Many practical robot applications require the control of the force exerted by the manipulator normal to a surface along with position control in the plane of the surface. This is the case in milling and grinding, surface finishing, and the like. In this case, the NN force/position controller in Fig. 8 and Table 4 can be derived (27). It has guaranteed performance in that both the position-tracking error $r(t)$ and the force error $\tilde{\lambda}(t)$ are kept small, while all the NN weights are kept bounded.

In Table 4, the selection matrix L and jacobian J are computed based on the decomposition of the joint variable $q(t)$ into two components—the component q_1 (e.g., tangenital to the given surface) in which position tracking is desired and the component q_2 (e.g., normal to the surface) in which force exertion is desired. This is achieved using standard robotics holonomic constraint techniques based on the prescribed surface. The filtered position tracking error in $q_1(t)$ is $r(t)$, that

Table 4. NN Force/Position Controller

Control Input:

$$\tau = \hat{W}^T\sigma(\hat{V}^Tx) + K_v(Lr) - J^T(\lambda_d - K_f\tilde{\lambda}) - v$$

NN Weight/Threshold Tuning Algorithms:

$$\dot{\hat{W}} = F\sigma(\hat{V}^Tx)(Lr)^T - F\hat{\sigma}'\hat{V}^Tx(Lr)^T - \kappa F\|(Lr)\|\hat{W}$$
$$\dot{\hat{V}} = Gx(\hat{\sigma}'^T\hat{W}(Lr))^T - \kappa G\|(Lr)\|\hat{V}$$

Design parameters: F, G positive definite matrices and $\kappa > 0$

Robustifying signal:

$$v(t) = -K_z(\|\hat{Z}\| + Z_M)r$$

is, $r = q_{1d} - q_1$, with $q_{1d}(t)$ the desired trajectory in the plane of the surface. The desired force is described by λ_d, and the force exertion error is captured in $\tilde{\lambda} = \lambda - \lambda_d$, with λ describing the actual measured force exerted by the manipulator. The position tracking gain is K_v, and the force tracking gain is K_f.

The structure of the NN force controller is the same as the multiloop NN controller in Fig. 5, with the addition of an inner loop for force control. This multiloop intelligent control topology appears to be very versatile and powerful indeed.

NN Controller for Electrically Driven Robot Using Backstepping. Robot manipulators are driven by actuators, which may be electric, hydraulic, pneumatic, and so on. The actuators are coupled to the links through coupling mechanisms that may contain gears. Particularly in the case of high-speed performance requirements, the coupling shafts may exhibit appreciable compliance that cannot be disregarded. Many real-world systems in industrial and military applications also have flexible modes and vibratory effects. In all these situations, the NN controller in Fig. 5 must be modified. Two design techniques that are particularly useful for this purpose are singular perturbations and backstepping (22,28).

A typical example of a real robotic system is the robot arm with electric actuators, or rigid-link electrically driven (RLED) manipulator. The dynamics of an n-link rigid robot arm with motor electrical dynamics are given by

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + F(\dot{q}) + G(q) + \tau_d = K_I i \tag{34}$$

$$Li + R(i,\dot{q}) + \tau_e = u_e \tag{35}$$

with $q(t) \in \Re^n$ the joint variable, $i(t) \in \Re^n$ the motor armature currents, K_T a diagonal electromechanical conversion matrix, L a matrix of electrical inductances, $R(i, \dot{q})$ representing both electrical resistance and back emf, $\tau_d(t)$ and $\tau_e(t)$ the mechanical and electrical disturbances, and motor terminal voltage vector $u_e(t) \in \Re^n$ the control input.

This plant has unknown dynamics in both the robot subsystem and the motor subsystem. The NN tracking controller in Fig. 9 was designed using the backstepping technique. The NN weight-tuning algorithms are similar to the ones presented in Tables 1–3 but with some extra terms. This controller has two neural networks, one (NN#1) to estimate the unknown robot dynamics and an additional NN in an inner feedback loop (NN#2) to estimate the motor dynamics. This multiloop controller is typical of control systems designed using rigorous system theoretic techniques. It can be shown that by selecting suitable weight-tuning algorithms for both NN, we can guarantee closed-loop stability as well as tracking performance in spite of the additional high-frequency motor dynamics.

Feedforward Control Loops: Compensation of Actuator Deadzones

Many industrial motion control systems have nonlinearities in the actuator, either deadzone, backlash, saturation, or the like. This includes xy-positioning tables, robot manipulators, overhead crane mechanisms, and more. The problems are particularly exacerbated when the required accuracy is high, as in micropositioning devices. Because of the nonanalytic nature of the actuator nonlinearities and the fact that their exact parameters (e.g., width of deadzone) are unknown, such systems present a challenge for the control design engineer.

The deadzone nonlinearity shown in Fig. 10 is characteristic of actuator nonlinearities in industrial systems. Proportional-derivative controllers have been observed to result in limit cycles if the actuators have deadzones. Techniques that have been applied for overcoming deadzone include variable structure control, dithering (29), and adaptive control (30,31).

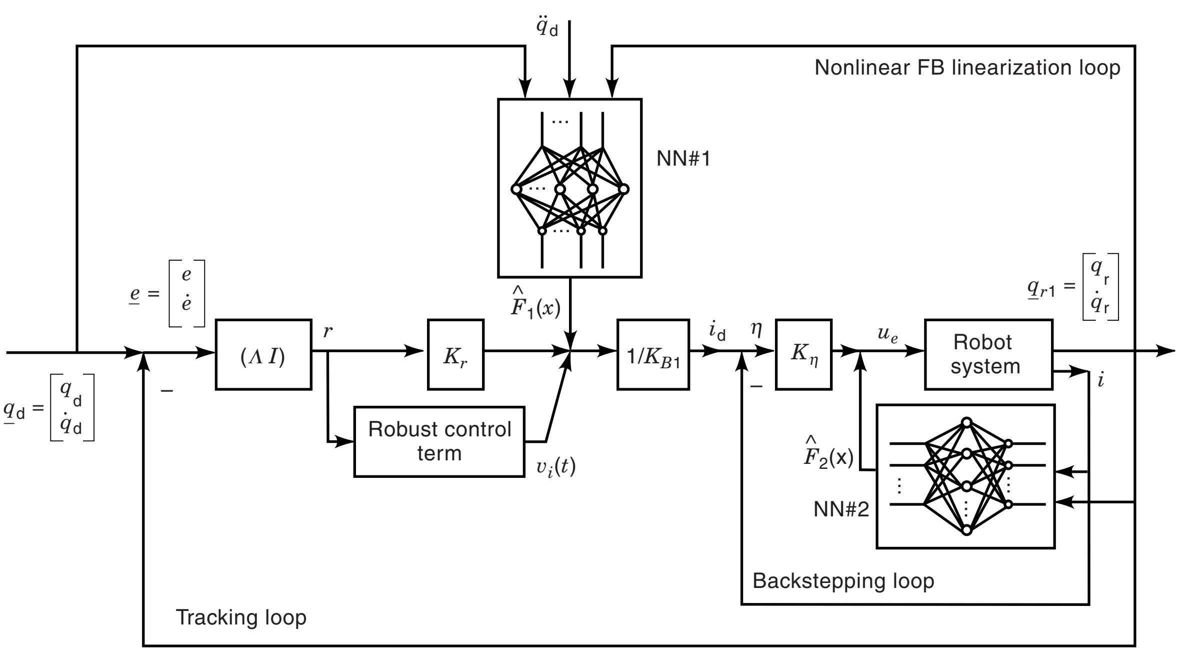

Figure 9. Multiloop NN backstepping controller, showing inner backstepping loop with a second NN.

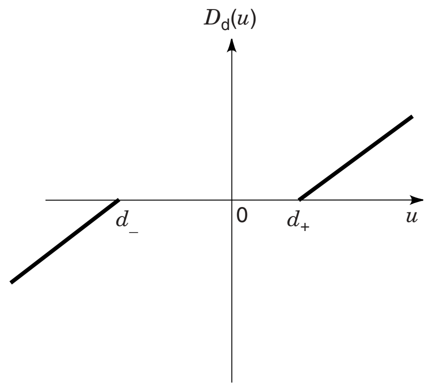

Figure 10. Nonsymmetric deadzone nonlinearity.

The deadzone is a piecewise continuous function $f(x)$ whose discontinuity points make most NN approximation proofs invalid and bring into question the accuracy of the approximation expression Eq. (5). To approximate the deadzone function well at the point of discontinuity, we must add more hidden-layer neurons. Even then, we often observe a Gibbs phenomenon sort of oscillation in the NN output near the discontinuity point. To remedy these problems, we may use an augmented NN for approximation of functions with jumps (32). The NN augmented for jump approximation is shown in Fig. 11. It has L hidden layer neurons that use standard smooth activation functions $\sigma(\cdot)$ such as the sigmoid, plus some extra neurons having discontinuous activation functions $\varphi_i(\cdot)$. These extra functions must provide a jump function basis set, the first of which, $\varphi_1(x)$, is the unit step. It can be shown that, with the augmented neurons, the NN can approximate piecewise continuous functions very well.

To compensate for deadzones and other actuator nonlinearities, the augmented NN may be placed in the feedforward path as shown in Fig. 12. When suitably adapted, using a weight-tuning algorithm very much like that presented in Table 1, the NN effectively estimates a preinverse for the deadzone, thereby compensating for its deleterious effects. The NN deadzone compensator can be viewed as an adaptive dithering scheme because it injects an additional component into the control signal that adds energy at the points where the control crosses zero, thereby overcoming the deadzone. The performance of this controller has been observed to be very good on actual CNC machine tools.

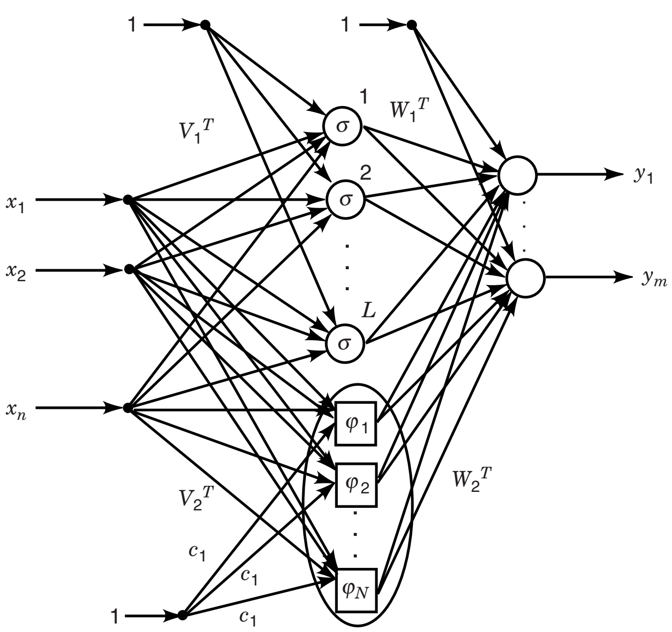

Figure 11. Augmented NN for approximation of functions with jumps, showing additional neurons having jump approximation functions.

OUTPUT FEEDBACK CONTROL USING DYNAMIC NEURAL NETWORKS

The previous section dealt with NN controller design of what is called in system theory the primary feedback loop, and in computer science the action or control generating loop. In this section and the next, it is shown that if there are additional plant complexities, increased performance requirements, or reduced information available, then the controller requires a sort of hierarchical structure that can contain NN at higher levels of abstraction.

Reduced Measurements and the Output-Feedback Problem

If all the states of the controlled plant are available as measurements, then the static NN controllers presented in the previous section can be used. It is noted that, even though the NN are static in themselves, the closing of a feedback loop around them turns them into dynamic NN in conjunction with the plant dynamics.

Unfortunately, in actual industrial and commercial systems, there are usually available only certain restricted measurements of the plant because due to economic or physical constraints, all the state components cannot be measured. This is known as output-feedback control as opposed to full state-feedback control. In this case, we must use an additional NN with its own internal dynamics in the controller (33). The function of the NN dynamics is effectively to provide estimates of the unmeasurable plant states, so that the dynamic NN functions as an observer in control system theory.

Taking the representative Lagrangian mechanical system dynamics

$$M(q)\ddot{q} + V_m(q,\dot{q})\dot{q} + G(q) + F(\dot{q}) + \tau_d = \tau \tag{36}$$

let there be available now only measurements of the joint variable vector $q(t) \in \Re^n$, that is of the robot joint angles or extensions. Specifically, the joint velocities $\dot{q}(t)$ are not measured. This is a typical situation in actual industrial applications, where optical encoders are used to measure $q(t)$.

Dynamic NN Observer for Data Reconstruction and a Two-NN Controller

It can be shown that the following dynamic NN observer can provide estimates of the entire state $x = [x_1^T\ x_2^T]^T \equiv [q^T\ \dot{q}^T]^T$ given measurements of only $x_1(t) = q(t)$

$$\dot{\hat{x}}_1 = \hat{x}_2 + k_D\tilde{x}_1 \tag{37}$$

$$\dot{\hat{z}}_2 = M^{-1}(x_1)[\tau - \hat{W}_o^T\sigma_o(\hat{x}) + k_P\tilde{x}_1 + v_o] \tag{38}$$

$$\hat{x}_2 = \hat{z}_2 + k_{P2}\tilde{x}_1 \tag{39}$$

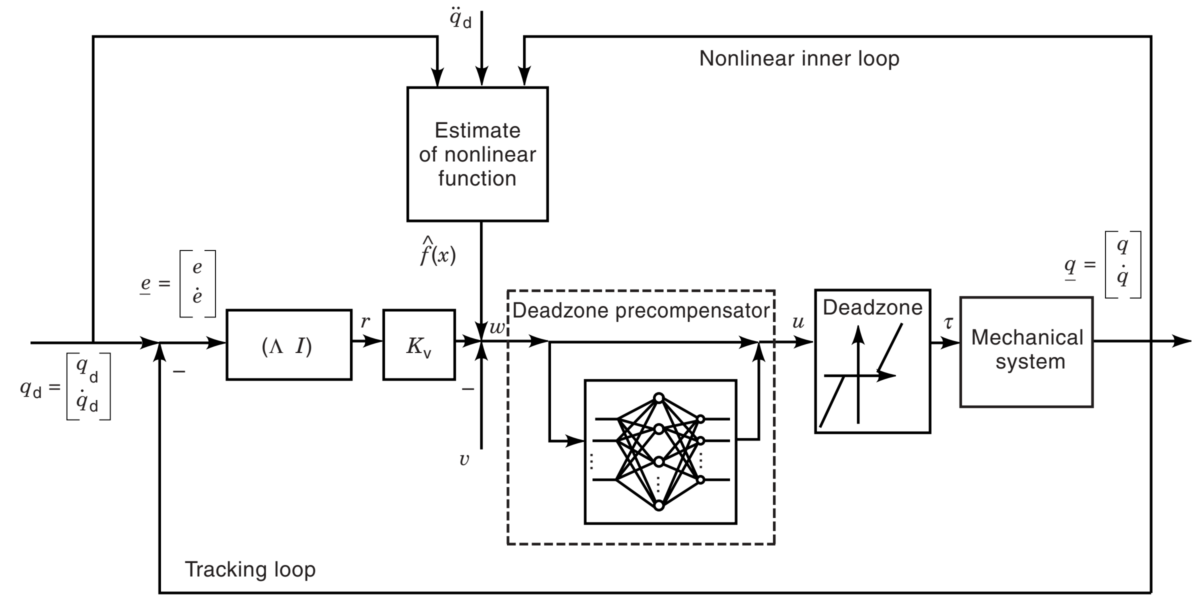

Figure 12. Tracking controller with feedforward NN deadzone compensation.

In this system, hat denotes estimates, and tilde denotes estimation errors (e.g., $\tilde{x}_1 = x_1 - \hat{x}_1$, $\tilde{x}_2 = x_2 - \hat{x}_2$). It is assumed that the inertia matrix $M(q)$ is known but that all other nonlinearities are estimated by the observer NN $W_o^T \sigma_o(\hat{x})$, which has output-layer weights W_o^T and activation functions $\sigma_o(\cdot)$. This system is a dynamic NN of a special structure because it has its own dynamics in the integrators corresponding to the estimates $\hat{x}_1$, $\hat{z}_2$. Signal $v_o(t)$ is an observer robustifying term, and the observer gains k_P, k_D, k_{P2} are positive design constants.

The NN output-feedback tracking controller shown in Fig. 13 uses the dynamic NN observer to reconstruct the missing measurements $x_2(t) = \dot{q}(t)$, and then employs a second static NN for tracking control, as in Fig. 5. Because neither the joint velocities $x_2(t) = \dot{q}(t)$ nor the tracking error $r(t)$ is directly measurable, the control input in Tables 1 and 2 must be modified so it becomes

$$\tau = \hat{W}_c^T \sigma_c(\hat{x}) + K_v \hat{r} + \Lambda e - v_c \tag{40}$$

where the estimated or measurable portion of the tracking error is

$$\hat{r} = (\dot{q}_d - \hat{x}_2) + \Lambda e = r + \tilde{x}_2 \tag{41}$$

with $e(t) = q_d(t) - x_1(t)$ as before. The control NN has weights W_c and activation functions $\sigma_c(\cdot)$, and $v_c(t)$ is a control robustifying signal. Note that the outer tracking PD loop structure has been retained.

In this dynamic NN controller, two NN must be tuned. Note that this formulation shows both the observer NN and the control NN as a one-layer FLNN; therefore, both $\sigma_o(\cdot)$ and $\sigma_c(\cdot)$ must be selected as bases. A more complex derivation shows that both can in fact be taken as two-layer NN. It can be shown (33) that both the static control NN weights W_c and the dynamic observer NN weights W_o should be tuned using variants of the algorithm presented in Table 1. It is evident from this design that if the plant has additional complications or uncertainties, more hierarchical structure must be added to the control system.

HIERARCHICAL INTELLIGENT CONTROL: ADAPTIVE REINFORCEMENT LEARNING AND HAMILTON–JACOBI–BELLMAN OPTIMAL DESIGN

Traditionally, control engineers design control systems from the point of view of the primary feedback loops or action generating loops. On the other hand, computer science engineers focus their interest on higher-level or outer control loops designed using biological or psychological precepts, such as reinforcement learning, training, performance critics, or user-specified performance criteria. The result has been a communications gap in intelligent NN controller design. In this section we attempt to bridge this gap by providing some higher-level controllers with hierarchical designs that are similar in structure to controllers designed using computer science techniques. These structures should be compared with those described in Ref. (4) and NEURAL NET ARCHITECTURE and NEUROCONTROLLERS. Two hierarchical controllers are detailed here: an adaptive reinforcement learning NN controller and a Hamilton-Jacobi-Bellman (HJB) Optimal NN Controller. Both can be shown by rigorous stability proof techniques to guarantee stable tracking and closed-loop performance (33). In neither case is a preliminary off-line learning phase, so detrimental to feedback control requirements, needed.

Direct Reinforcement Adaptive Learning NN Controller

Reinforcement learning techniques are based on psychological precepts of reward and punishment as used by I. P. Pavlov in the training of dogs at the turn of the century. The key tenet here is that the performance indicators of the controlled system should be simple, for instance, "plus one" for a successful trial and "negative one" for a failure, and that these simple signals should tune or adapt an NN controller so that its performance improves over time. This gives a learning feature driven by the basic success or failure record of the controlled system.

Generating the Reinforcement Signal from the Instantaneous Utility. In the NN controllers described previously and whose structure is given in Fig. 5, the NN tuning was performed in an inner action-generating loop based on a filtered tracking error signal $r(t) = q_d(t) - q(t)$ that was measured in an outer PD tracking control loop. The performance of the plant was captured in this tracking error $r(t)$, which is small as long as the tracking is satisfactory, that is, as long as the actual plant output $q(t)$ follows the desired trajectory $q_d(t)$. We also showed that if there are complications with the plant so that its entire internal state cannot be measured, then the controller must be based not on the actual filtered tracking error $r(t)$ but on an estimated tracking error $\hat{r}(t)$, which was reconstructed by an additional dynamic NN observer. The output-feedback NN controller shown in Fig. 13 requires two NN. Thus, as the actual system performance is known less and less accurately, as more and more uncertainty is injected, increased structure is needed in the controller.

Unfortunately, using the complete filtered error signal $r(t)$ in tuning the action-generating NN countermands the philosophy of reinforcement learning, where all the performance data of the closed-loop system should be captured in simple signals that contain reward/punishment information. A simple signal related to the tracking error is the signum of the filtered tracking error $R(t) = \text{sgn}[r(t)]$. The signum function is shown in Fig. 14(a). The signal $R(t)$ corresponding to a sample signal $r(t)$ is given in Fig. 14(b).

It is clear that $R(t)$ satisfies the criteria required in reinforcement learning control. (1) It is simple, having values of only 0, ± 1, and (2) the value of zero corresponds to a reward for good performance, whereas nonzero values correspond to a punishment signal. Therefore, $R(t)$ will be taken here as a suitable reinforcement learning signal. In reinforcement learning, the signal $r(t)$ could be called the instantaneous utility.

Architecture and Learning for the Adaptive Reinforcement Learning Controller. It is not easy to show how to tune the action-generating NN using only the reinforcement signal $R(t)$, which contains significantly less information than the full error signal $r(t)$. The success of the derivation lies in selecting the Lyapunov energy function

$$\nu = \sum_{i=1}^{n} |r_i| + \frac{1}{2}\text{tr}(\tilde{W}^T F^{-1} \tilde{W}) \tag{42}$$

where $|\cdot|$ is the absolute value and n is the number of states [i.e., $r(t) \in \Re^n$]. This is not a standard Lyapunov function in system theory, but it is similar to enegy functions used in some NN convergence proofs. Using this Lyapunov function as the basis for a nonlinear stability proof, we can derive NN

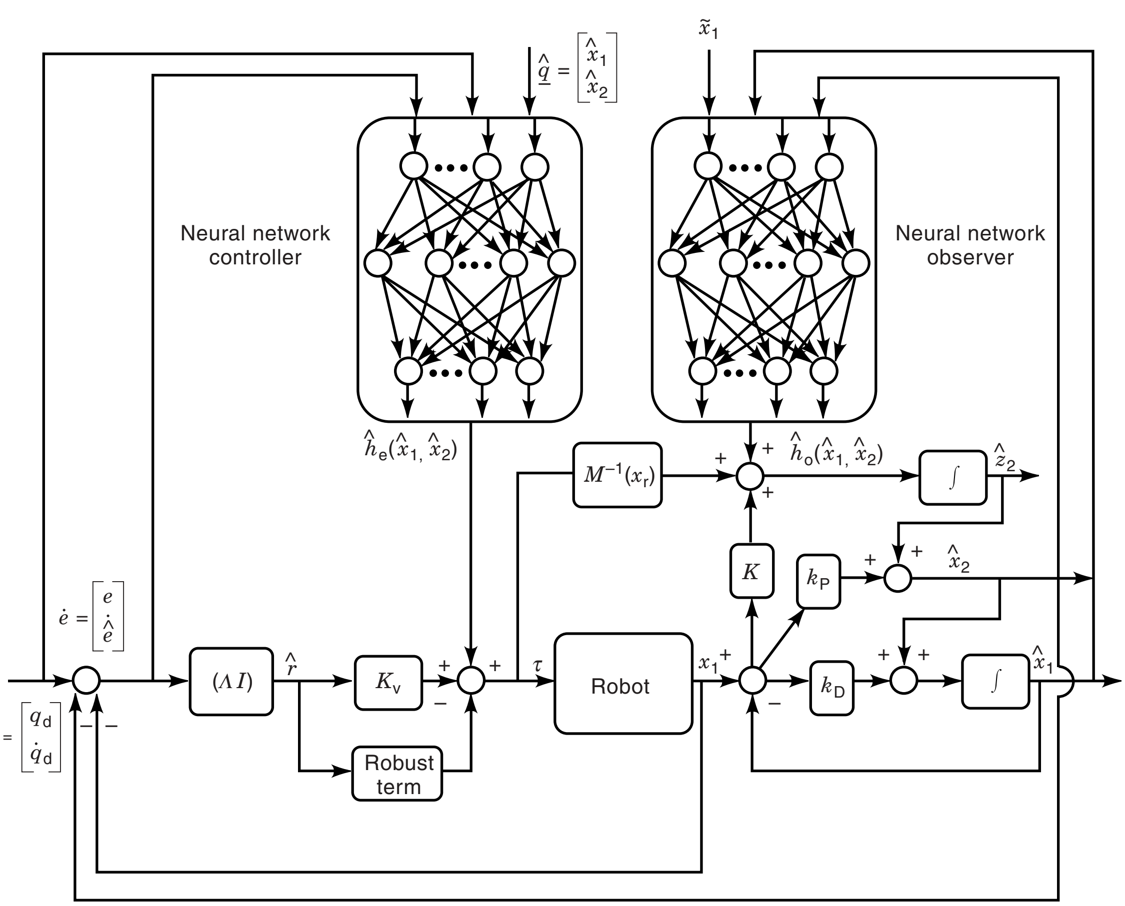

Figure 13. Dynamic NN tracking controller with reduced measurements, showing second dynamic NN loop required for state estimation.

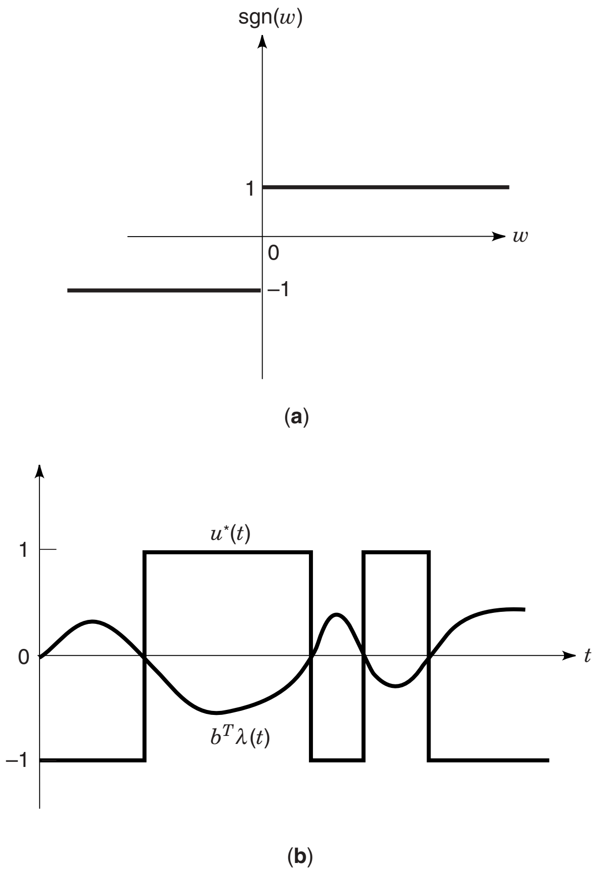

Figure 14. Generating the reinforcement signal $R(t)$ from the instantaneous utility $r(t)$. (a) The signum function. (b) Sample tracking error $r(t)$ and its signum $R(t)$, which has reduced information content.

tuning algorithms that guarantee closed-loop stability and tracking.

The architecture of the direct-reinforcement adaptive learning (DRAL) NN controller derived using this technique is shown in Fig. 15. Note that it is again a multiloop controller, with an inner action-generating loop containing a NN. The performance evaluation loop corresponds to a PD tracking loop with the desired trajectory $x_d(t)$ as the user input; this loop manufactures the instantaneous utility $r(t)$. A block that can be considered as a critic element evaluates the signum function and so provides the reinforcement signal $R(t) = \text{sgn}[r(t)]$, which critiques the performance of the system.

The NN weights are tuned using

$$\dot{\hat{W}} = F\sigma(x)R^T - \kappa F\hat{W} \tag{43}$$

It is important to note that this involves only the simplified signal $R(t)$ with reduced information content, not the full tracking error $r(t)$. This is similar to what has been called sign error tuning in adaptive control, which has usually been proposed without given any proof of stability or performance.

No preliminary off-line learning phase is needed for this reinforcement learning controller. The NN weights are initialized at zero, and the PD critic loop keeps the error bounded until the NN in the action-generating loop begins to learn the unknown dynamics. Then, after a short time the tracking performance improves dramatically.

Hamilton–Jacobi–Bellman Performance-Based NN Controller

The NN controllers discussed in this article have multiple feedback loops and can be shown using rigorous stability proofs to have guaranteed performance in terms of small tracking error and bounded internal signals. Modified NN weight-tuning algorithms were given that are suitable for closed-loop control purposes. The discussion has heretofore centered around the primary feedback loops or action-generating loops, with the adaptive reinforcement controller introducing a performance critic loop. It has been seen that as uncertainty is introduced in the form of less information available from the plant, the controller requires more hierarchical structure to compensate.

The design of the NN controllers has involved two arbitrary steps. First, a matrix gain Λ must be selected to generate the filtered error $r(t)$ in Eq. (21). Second, the stability proofs for the NN controllers have relied on the selection of a positive Lyapunov-like energy function $\mathcal{V}(t)$. The requirement for stability is that the Lyapunov derivative $\dot{\mathcal{V}}(t)$ be negative outside a compact region. This requirement leads to the tuning algorithms and control structures being selected as detailed. However, the Lyapunov function may seem to be a somewhat artificial device that is introduced in an ad hoc fashion to prove stability; perhaps a different function $\mathcal{V}(t)$ would yield different control structures and NN tuning algorithms. It is desirable to select a more natural way for the user input to appear in the system design.

Interfaces Between Feedback Loops and the Human User. The subject of the user interface for intelligent systems has been debated in recent years. The NN feedback controllers exhibit some aspects of biological systems in that they can adapt and learn using nonlinear network structures akin to those of neurons; therefore, they may be called intelligent systems. However, it is important to provide a smooth transition between the regulatory functions of the feedback controller and the supervisory functions of the human operator. The adaptive reinforcement learning scheme in Fig. 15 is a step in this direction, because given the user input prescribed trajectory $x_d(t)$, the critic block evaluates the tracking performance of the plant through observations of the error $r(t)$ and manufactures a simplified reward/punishment signal $R(t)$ that is used to adapt the NN.

On another issue, the NN controllers heretofore discussed have exhibited a certain measure of intelligence. However, they operate in a well-defined structural setting so that they may fail some definitions of intelligence. It is desirable to imbue control systems with higher levels of abstraction so that they can face additional uncertainties in the environment.

Optimal Control, Performance, and the HJB Equation. Many systems occurring naturally in biology, sociology, and elsewhere use feedback control to achieve homeostasis, or equilibrium conducive to existence. Because the bounds within which life can continue are very small (e.g., temperature changes of a few degrees can eliminate populations) and the resources available are often limited, it is remarkable yet not unexpected that most of these feedback control systems have

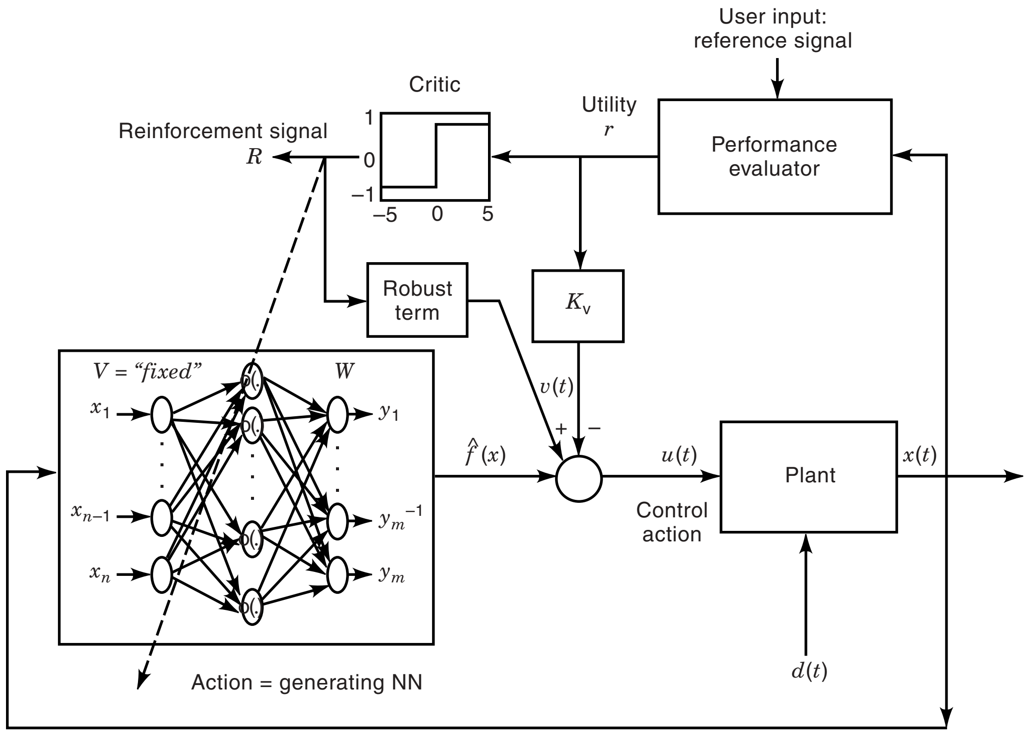

Figure 15. DRAL NN controller, showing inner NN action-generating loop and performance evaluation critic loop.

evolved into optimal systems, which achieve desired results with a minimum of required energy. Because naturally occurring systems are optimal, it makes a great deal of sense to design man-made controllers from the point of view of optimality.

Optimal Control Design, System Performance, and Human User Input. Let a system or plant be given by

$$\dot{z} = g(z, u) \tag{44}$$

where $z(t)$ is the state and $u(t)$ is the control input. Desirable performance of such a dynamical system may be described in terms of a performance measure (PM) such as the quadratic integral form

$$J(u) = \int_0^\infty L(z, u)\, dt \tag{45}$$

where the instantaneous performance is captured in the Lagrangian function

$$L(z, u) = \frac{1}{2}[z^T(t)Qz(t) + u^T(t)Ru(t)] \tag{46}$$

with matrices Q, R symmetric and positive definite.

The human user input consists of the state weighting matrix Q and the control weighting matrix R, which can be selected in a very natural way to result in desirable system performance of various sorts, as is well known in standard control theory texts (34). Selection of Q, R may be accomplished using engineering design based on compromises between performance [e.g., keeping $z(t)$ small] and energy efficiency [e.g., keeping $u(t)$ small].

The optimal control design problem is to select an optimal control $u^*(t)$ that minimizes the PM Eq. (45) for the prescribed dynamical system Eq. (44).

Bellman's Optimality Principle and the HJB Equation. The basic principle in the design of optimal systems is captured in Bellman's Optimality Principle:

> An optimal policy has the property that no matter what the previous decisions (e.g., controls) have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions.
>
> *F. L. Lewis and V. L. Syrmos, Optimal Control, 2nd ed. New York: Wiley, 1995.*

The design of optimal control systems is discussed in (34). Applying Bellman's Optimality Principle to discrete-time systems results in the derivation of dynamic programming algorithms and to continuous-time systems results in the derivation of the Hamilton-Jacobi-Bellman equation.

It may be found from Bellman's Optimality Principle that a necessary and sufficient condition for a control $u^*(t)$ to optimize the PM Eq. (45) for the system Eq. (44) is that there exists a value function $\mathscr{V}(z, t)$ that satisfies the HJB equation

$$\frac{\partial \nu(z,t)}{\partial t} + \min_u \left\{ H\left[z, u\frac{\partial \nu(z,t)}{\partial z}, t\right]\right\} = 0, \tag{47}$$

where the Hamiltonian function is given by

$$H\left[z, u, \frac{\partial \nu(z,t)}{\partial z},\ t\right] = L(z,u) + \frac{\partial \nu(z,t)}{\partial z} g \tag{48}$$

Optimal NN Controller for Robotic Systems. The HJB equation is extremely difficult to solve for general nonlinear systems Eq. (44), but for linear systems it can be explicitly solved and yields the linear quadratic regulator design equations, which are basic in modern control theory (34). Fortunately, nonlinear mechanical systems such as the robot dynamics in Lagrangian form have special properties that allow us to solve the HJB equation and obtain explicit controller equations (35).

Solution to the Robot System Optimal Design Problem. For the robotic system Eq. (19), define the tracking error $e(t) = q_d(t) - q(t)$, the filtered tracking error

$$r = \dot{e} + \Lambda e \tag{49}$$

the overall state $z = [e^T\ r^T]^T$, and the input-related term

$$u = f(x) - \tau \tag{50}$$

with $f(x)$ the unknown nonlinear robot function Eq. (23). Then, it can be shown (33) that for the PM Eq. (45), a value function that satisfies the HJB equation is given by

$$\nu(z,t) = \tfrac{1}{2} z^T P(q) z = \tfrac{1}{2} z^T \begin{bmatrix} K & 0 \\ 0 & M(q) \end{bmatrix} z \tag{51}$$

The matrix $P(q)$ is given as the solution to a nonlinear Riccati equation. This Riccati equation may be explicitly solved to yield the positive definite symmetric matrix K and the filtered error gain Λ as

$$K = -\tfrac{1}{2}(Q_{12} + Q_{12}^T) \tag{52}$$

$$\Lambda_s^T K + K\Lambda_s = Q_{11} \tag{53}$$

$$\Lambda = \Lambda_s + K^{-1} Z \tag{54}$$

where the state-weighting matrix entered by the human user is partitioned as

$$Q = \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{12}^T & Q_{22} \end{bmatrix} \tag{55}$$

The symmetric portion Λ_s is found by solving the Lyapunov equation Eq. (53) using standard efficient techniques and Z is any antisymmetric matrix (e.g., $Z^T = -Z$). Note that according to this design, Λ need not be symmetric. The control-weighting matrix must satisfy $R^{-1} = Q_{22}$.

Optimal NN Controller. In terms of these constructions, the optimal NN controller is given as

$$\tau = \hat{W}^T \sigma(x) + R^{-1}(\dot{e} + \Lambda e) - v \tag{56}$$

with the first term supplied by an NN, the second term the optimal control portion, and the last term a robustifying term. It is not difficult to show that the value function $\mathcal{V}(z, t)$ serves as a Lyapunov energy function and, hence, to prove the closed-loop stability of the optimal NN controller. During this

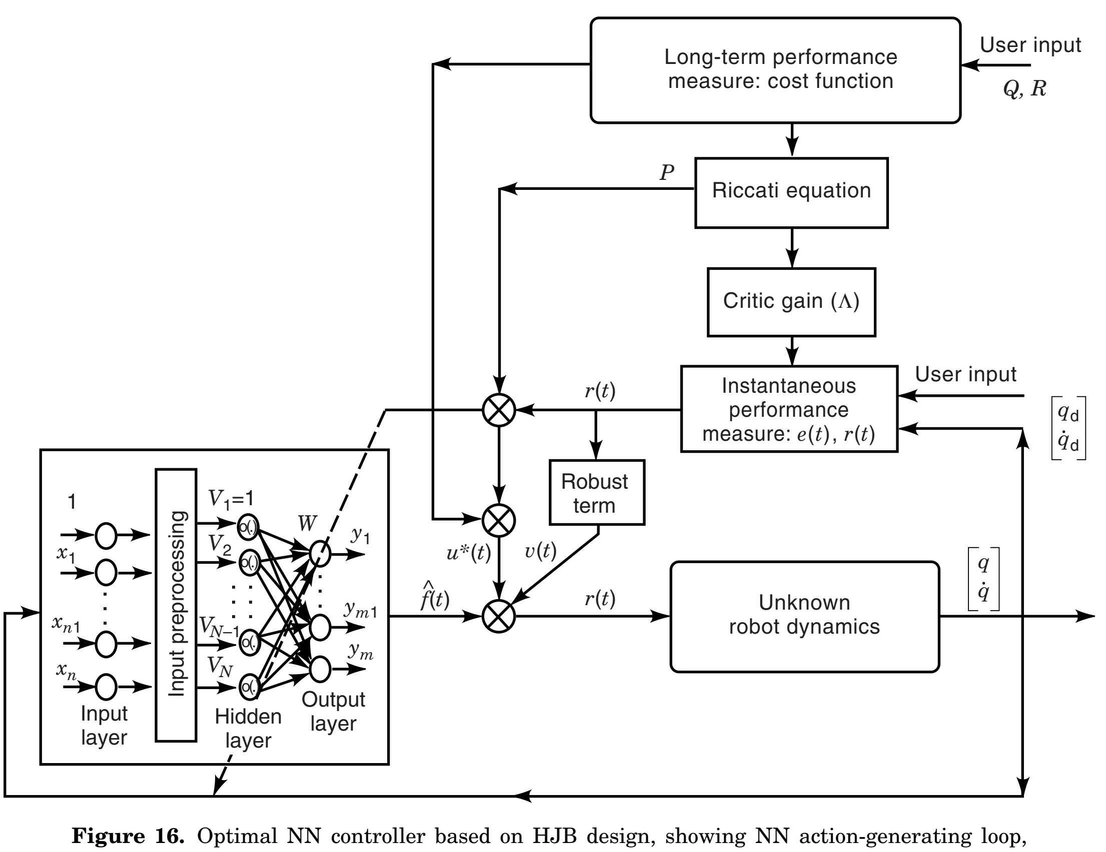

Figure 16. Optimal NN controller based on HJB design, showing NN action-generating loop, critic loop, and user performance measure input loop.

procedure, the NN weight-tuning algorithm is found; it is similar to that given in Table 1. Note that a one-layer FLNN is used here, even though it is possible to use a two-layer NN.

The NN controller resulting from the HJB design approach appears in Fig. 16. It is a hierarchical system with more structure than the previous NN controllers, even though its lower loops include the same action-generating loop and the PD loop to compute $r(t)$. In contrast to the previously discussed controllers, it is no longer necessary to select an ad hoc value for Λ, which could be considered as a critic gain. Nor is it necessary to select a Lyapunov function such as Eq. (28) to derive the NN weight-tuning algorithms. Instead, the user input is through the desired trajectory $q_d(t)$ and the PM weighting matrices Q, R which specify the desired performance of the closed-loop system. Then, the Riccati solution gives both Λ and the Lyapunov function $\mathcal{V}(z, t)$. Thus, the user input in terms of performance criteria has been used to derive both the suitable signal $r(t)$ to be measured by the critic, as well as the Lyapunov function required for stability proofs.

EVALUATION

Repeatable neural net controller design algorithms were given for a general class of industrial Lagrangian motion systems characterized by the rigid robot arms. The design procedure is based on rigorous nonlinear derivations and stability proofs, and yields a multiloop intelligent control structure with NN in some of the loops. NN weight-tuning algorithms were given that do not require complicated initialization procedures or any off-line learning phase, work on-line in real time, and offer guaranteed tracking and bounded control signals. The NN controllers given here are model-free controllers in that they work for any system in a prescribed class without the need for extensive modeling and preliminary analysis to find a "regression matrix." Unlike adaptive controllers, they do not require persistence of excitation, linearity in the parameters, model matching, or certainty equivalence.

As the uncertainty in the controlled plant increases or the performance requirements become more complex, it is necessary to modify the NN controller by adding additional feedback loops. A force controller and a controller for a combined electromechanical system were given that needed additional inner control loops. Deadzone compensation was shown to need additional feedforward loops. A reinforcement learning controller was given with an additional critic loop. Finally, an optimal NN controller was designed with extra loops at higher levels of abstraction to accommodate a user input in terms of the general desired performance of the closed-loop system. In all cases, rigorous design algorithms, stability proofs, and performance guarantees can be given.

There are other rigorous approaches to NN feedback control. Narendra (3) shows how to place an NN into many standard feedback control systems, including series forms, series-parallel forms, and direct and indirect forms. Werbos (4) provides design techniques for high-level NN controllers with loops having multiple levels of abstraction; see NEUROCONTROLLERS. Sanner and Slotine (11) show how to use RBF NN in feedback control. Chen and Khalil (18) and Chen and Liu (19) provide NN tuning algorithms based on deadzone methods, while Polycarpou and Ioannou (15,16) use projection methods. Sadegh (10) provides NN controllers for discrete-time systems, and Rovithakis and Christodoulou (17) use dynamic NN for feedback control.

BIBLIOGRAPHY

1. D. A. White and D. A. Sofge (eds.), *Handbook of Intelligent Control.* New York: Van Nostrand Reinhold, 1992.
2. W. T. Miller, R. S. Sutton, and P. J. Werbos (eds.), *Neural Networks for Control.* Cambridge: MIT Press, 1991.
3. K. S. Narendra, Adaptive control using neural networks. In W. T. Miller, R. S. Sutton, and P. J. Werbos (eds.), *Neural Networks for Control,* pp. 115–142. Cambridge: MIT Press, 1991.
4. P. J. Werbos, Neurocontrol and supervised learning: an overview and evaluation. In D. A. White and D. A. Sofge, (eds.), *Handbook of Intelligent Control.* New York: Van Nostrand Reinhold, 1992.
5. S. Haykin, *Neural Networks.* New York: IEEE Press, 1994.
6. K. Hornik, M. Stinchombe, and H. White, Multilayer feedforward networks are universal approximators. *Neural Networks,* **2**: 359–366, 1989.
7. K. J. Åström and B. Wittenmark, *Adaptive Control.* Reading, MA: Addison Wesley, 1989.
8. Y. D. Landau, *Adaptive Control.* Basel: Marcel Dekker, 1979.
9. P. J. Werbos, Back propagation: Past and future. *Proc. 1988 Int. Conf. Neural Nets,* vol. 1, pp. I343–I353, 1989.
10. N. Sadegh, A perception network for functional identification and control of nonlinear systems. *IEEE Trans. Neural Netw.,* **4** (6): 982–988, 1993.
11. R. M. Sanner and J.-J. E. Slotine, Stable adaptive control and recursive identification using radial gaussian networks. *Proc. IEEE Conf. Decision and Control,* Brighton, 1991.
12. B. Kosko, *Neural Networks and Fuzzy Systems.* NJ: Prentice-Hall, 1992.
13. J. S. Albus, A new approach to manipulator control: The Cerebellar Model Articulation Controller (CMAC). *Trans. ASME J. Dynam. Sys., Meas., Control,* **97** (3): 220–227, 1975.
14. F. L. Lewis, K. Liu, and S. Commuri, Neural networks and fuzzy logic systems for robot control. In F. Wang, (ed.), *Fuzzy Logic and Neural Network Applications.* World Scientific Pub., to appear, 1998.
15. M. M. Polycarpou and P. A. Ioannou, Identification and control using neural network models: Design and stability analysis. Tech. Report 91-09-01, Dept. Elect. Eng. Sys., Univ. S. Cal., Sept. 1991.
16. M. M. Polycarpou, Stable adaptive neural control scheme for nonlinear systems. *IEEE Trans. Autom. Control,* **41** (3): 447–451, Mar. 1996.
17. G. A. Rovithakis and M. A. Christodoulou, Adaptive control of unknown plants using dynamical neural networks. *IEEE Trans. Syst. Man Cybern.,* **24** (3): 400–412, Mar. 1994.
18. F.-C. Chen and H. K. Khalil, Adaptive control of nonlinear systems using neural networks. *Int. J. Control,* **55** (6): 1299–1317, 1992.
19. F.-C. Chen and C.-C. Liu, Adaptively controlling nonlinear continuous-time systems using multilayer neural networks. *IEEE Trans. Autom. Control,* **39** (6): 1306–1310, 1994.
20. S. Jagannathan and F. L. Lewis, Discrete-time neural net controller for a class of nonlinear dynamical systems. *IEEE Trans. Autom. Control,* **41** (11): 1693–1699, 1996.
21. F. L. Lewis, A. Yesildirek, and K. Liu, Multilayer neural net robot controller: Structure and stability proofs. *IEEE Trans. Neural Netw.,* **7** (2): 1–12, 1996.

22. F. L. Lewis, S. Jagannathan, and A. Yesildirek, *Neural Networks Control of Robot Manipulators and Nonlinear Systems.* London: Taylor and Francis, to appear, 1998.
23. F. L. Lewis, C. T. Abdallah, and D. M. Dawson, *Control of Robot Manipulators.* New York: Macmillan, 1993.
24. J.-J. Slotine and W. Li, *Applied Nonlinear Control.* Englewood Cliffs, NJ: Prentice-Hall, 1991.
25. K. S. Narendra and A. M. Annaswamy, A new adaptive law for robust adaptive control without persistent excitation. *IEEE Trans. Autom. Control,* **32**: 134–145, 1987.
26. R. Colbaugh, H. Seraji, and K. Glass, A new class of adaptive controllers for robot trajectory tracking. *J. Robotic Systems,* **11** (8): 761–772, 1994.
27. C.-M. Kwan, A. Yesildirek, and F. L. Lewis, Robust force/motion control of constrained robots using neural network. *Proc. IEEE Conf. Decision and Control,* pp. 1862–1867, Dec. 1994.
28. P. V. Kokotovic, Applications of singular perturbation techniques to control problems. *SIAM Rev.* **26** (4): 501–550, 1984.
29. C. A. Desoer and S. M. Shahruz, Stability of dithered nonlinear systems with backlash or hysteresis. *Int. J. Control,* **43** (4): 1045–1060, 1986.
30. G. Tao and P. V. Kokotovic, Adaptive control of plants with unknown dead-zones. *Proc. Amer. Control Conf.,* pp. 2710–2714, Chicago, 1992.
31. D. A. Recker et al., Adaptive nonlinear control of systems containing a dead-zone. *Proc. IEEE Conf. Decision and Control,* 2111–2115, 1991.
32. R. Selmic and F. L. Lewis, Neural network approximation of piecewise continuous functions: Application to friction compensation. Submitted, 1997.
33. Y. H. Kim, *Intelligent Closed-Loop Control Using Dynamic Recurrent Neural Network and Real-Time Adaptive Critic,* Ph.D. Dissertation, Dept. Electrical Engineering, The University of Texas at Arlington, Arlington, TX 76019, Sept. 1997.
34. F. L. Lewis and V. L. Syrmos, *Optimal Control,* 2nd ed. New York: Wiley, 1995.
35. R. Johansson, Quadratic optimization of motion coordination and control. *IEEE Trans. Autom. Control,* **35** (11): 1197–1208, 1990.

F. L. LEWIS
Y. H. KIM
The University of Texas at Arlington