Implementations of Learning Control Systems Using Neural Networks

Michael A. Sartori and Panos J. Antsaklis

The systematic storage in neural networks of prior information to be used in the design of various control subsystems is investigated. Assuming that the prior information is available in a certain form (namely, input/output data points and specifications between the data points), a particular neural network and a corresponding parameter design method are introduced. The proposed neural network addresses the issue of effectively using prior information in the areas of dynamical system (plant and controller) modeling, fault detection and identification, information extraction, and control law scheduling.

Incorporating Prior Knowledge

In many practical control problems, there exists substantial prior information about the various subsystems of the control system. In modeling these subsystems via neural networks, it is desirable to incorporate this prior knowledge into a neural network. A particular neural network architecture and an associated design methodology are presented in this paper to accomplish this for certain types of prior knowledge. Using this procedure, prior knowledge is directly and systematically stored in the neural network with no training. This has the advantage of having the information in a neural network form, which can be quickly accessed once implemented in hardware. This neural network can be used as an initial approximation to the system's behavior. As discussed later, a better approximation can be developed, for example, by adding another neural network in parallel and using training procedures to better approximate the desired behavior.

With the proposed neural network architecture, the following problems are investigated: dynamical system (plant and controller) modeling, fault detection and identification, information extraction, and control law scheduling. In all of these implementations, it is assumed that a set of training points and certain specifications for the behavior between the training points are provided. With this training set, the neural network is designed to exactly satisfy the specifications for the interpolation between the operating points. The proposed neural network has two layers of weights with sigmoid nonlinearities for the hidden layer and linear functions for the output layer. The connection of the hidden layer to the output layer and the specific choices for the weights are unique aspects of this neural network design approach.

The neural network design procedure presented in this paper represents one possible approach for satisfying the relationship between the input/output pairs of the training set. Clearly, there exist other methods to (mathematically) describe the desired curve and solve the problem; for instance, polynomial or spline approximations can be used to represent the desired function. An advantage of using the neural network approach described here instead of one of these schemes is that the actual construction of the neural network in hardware would utilize the inherent parallelism of the neural network and hence result in a fast processing time. Compared to other neural network techniques, the design scheme here has several advantages. For one, there is no training required; only the design time needed to choose the appropriate function parameters (i.e., the specifications for the interpolation between the training pairs) is necessary. Also, exact control of the generalization between the training points is guaranteed via the design scheme. Furthermore, the number of layers and the number of neurons needed to correctly implement the desired function are known precisely.

In the next section, the neural network and the design of the neural network to approximate a desired function are presented. The neural network chosen here uses sigmoid nonlinearities, while an alternative approach using Gaussian nonlinearities was introduced in [1], [2] to address the same problem. Neural networks for plant and controller modeling, fault detection and identification, information extraction, and control law scheduling are discussed, and three examples are supplied to illustrate the design scheme.

An earlier version of this paper was presented at the 1991 IEEE International Conference on Intelligent Control Systems, Arlington, VA, August 13-15, 1991. Michael A. Sartori was with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556. He is now with the David Taylor Research Center, Code 1941, Bethesda, MD 20084. Panos J. Antsaklis is with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556. This work was partly supported by the Jet Propulsion Laboratory under Contract 957856 and the ALCOA Foundation under a Science Support Grant.

Neural Network Design

The neural network design procedure is presented first for the implementation of a single-input-multi-output function and second for the implementation of a multi-input-multi-output function.

Single-Input Neural Network

In all of the neural network implementations, the neural network is designed to approximate a function given a specific set of training patterns and a set of generalization specifications. It is first assumed that the function to be approximated is a single-input function. For the training patterns, let $v(j)$ denote the jth input pattern for $1 \le j \le p$, and let the p-dimensional vector $\boldsymbol{v}$ denote the vector of all input patterns arranged in ascending order, where $v(j) > v(j-1)$ for $2 \ge j \ge p$. Let $d_i(j)$ denote the ith component of the jth desired output pattern for $1 \le i \le n$ and $1 \le j \le p$, and let the n-dimensional vector $\boldsymbol{d}(j)$ and the p-by-n-dimensional matrix $\boldsymbol{D}$ denote the vector for $1 \le j \le p$ and the matrix of desired outputs, respectively. Thus, the p pairs $\{v(j), \boldsymbol{d}(j)\}$ are the given training patterns.

With u as a scalar, let $\phi(u)$ be the n-dimensional output of a function approximating the relationship described by the training set $\{v(j), \boldsymbol{d}(j\}$ for $1 \le j \le p$. Let $\phi_i(u)$ denote the ith component of the functions output for $1 \le i \le n$. Three specifications are made on the function ϕ:
(i) If $u = v(j)$, then $\phi(u) = \boldsymbol{d}(j)$.
(ii) If $u \in [v(j) - \varepsilon_{j-}, v(j) + \varepsilon_{j+}]$ and $u \neq v(j)$, then $\phi_i(u) \in [d_i(j) - \gamma_{ij-}, d_i(j) + \gamma_{ij+}]$.

0272-1708/92/$03.00©1992IEEE

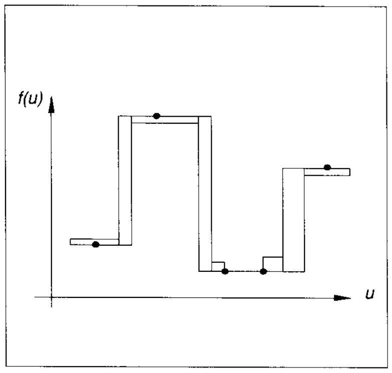


Fig. 1. Illustration of curve specifications.

(iii) If $u \in [v(j), v(j+1)]$, then $\phi_i(u) \in [d_i(j), d_i(j+1)]$.

As an example, Fig. 1 illustrates one way in which these specifications can be viewed. The dots correspond to the training points, and according to specification (i), the approximating curve must pass through these points. The boxes surrounding the dots correspond to the boxes described in specification (ii), and the boxes between the dots correspond to the bounding of the interpolation curve per specification (iii). The approximating curve must pass correctly through both sets of boxes. Clearly, there exist numerous curves that satisfy these three specifications. With the neural network design scheme described here, a subset of the class of curves described by ϕ is achieved in which these three specifications are exactly satisfied.

For the neural network, let the scalar u denote the neural network's input, and let z denote the neural network's n-dimensional output. For a specific input u, let $z_i(u)$ for $1 \leq i \leq n$ and $z(u)$ denote the output of the neural network. An individual output of the neural network is described by the weighted sum $z_i(u)$ where $1 \leq i \leq n$, w_{ik} is a weight of the ith linear neuron in the output layer, and g_k is the difference between two sigmoid functions:

$$z_i(u) = \sum_{k=1}^{h+1} w_{ik}\, g_k(u) \tag{1}$$

Let $\boldsymbol{W}$ denote the $(h+1)$-by-n-dimensional matrix of weights for the output layer. The output of the kth node in the neural network's hidden layer is described by the following where $1 \leq k \leq h+1$, σ is the sigmoid nonlinearity of the hidden layer neurons, c_k is the bias (or "center" of the sigmoid function), and s_k is the weight (or "slope" of the sigmoid function):

$$g_k(u) = \sigma(u, c_{k-1}, s_{k-1}) - \sigma(u, c_k, s_k) \tag{2}$$

and

$$\sigma(u,c,s) = \frac{1}{1+e^{-s(u-c)*}} \tag{3}$$

For all u, let

$$\sigma(u, c_0, s_0) = 1 \tag{4}$$

and

$$\sigma(u, c_{h+1}, s_{h+1}) = 0. \tag{5}$$

Let $\boldsymbol{c}$ denote the n-dimensional vector of centers, and $\boldsymbol{s}$ denote the n-dimensional vector of slopes. With $s_k = 1$, the conventional sigmoid curve is achieved, but by allowing s_k to vary, a class of sigmoid functions is possible, which allows for greater flexibility in the design of the interpolation curve. As an example of the formation of a Gaussian-type curve, Fig. 2 shows the output of $g_k(u) = \sigma(u, -2, 5) - \sigma(u, 1, 20)$ which can be viewed as a Gaussian-type curve centered at 0 with asymmetric sides. This asymmetry has advantages over the symmetrical Gaussian function used in the other neural network approach described in [1] and [2]; mainly, the sigmoid formed function aptly handles nonequidistant training sets.

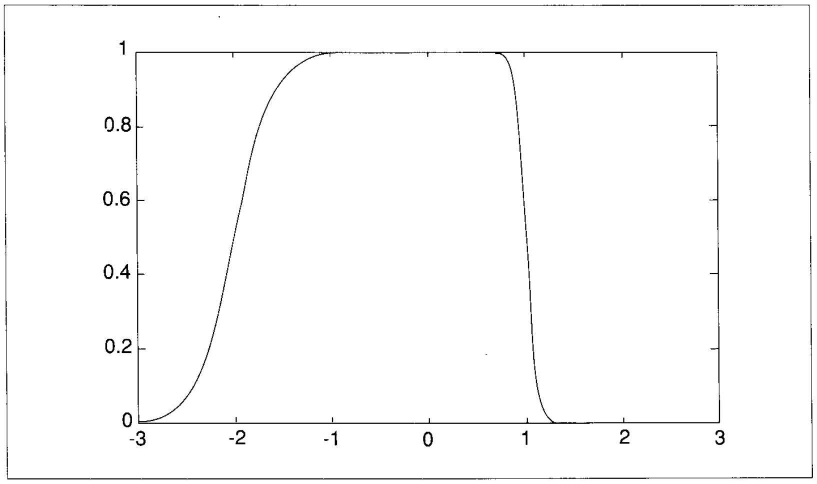


Fig. 2. The Gaussian-type function $g_k(u) = \sigma(u, -2, 5) - \sigma(u, 1, 20)$.

Design for the Single-Input Neural Network

In designing the neural network, its parameters are chosen such that its output z is in the class of functions ϕ described previously. The proposed neural network design scheme is a selection process based on the specifications for the interpolation between the operating points and does not require training. With this approach, exact control of the neural network's generalization behavior is achieved, and the three specifications described above are satisfied precisely, as will be shown.

The number of hidden layer nodes is set equal to one less the number of training patterns, that is $h = p - 1$. The center c_k for the kth node is set equal to the value midway between the kth and the $(k+1)$th input training points for $1 \leq k \leq h$:

$$c_k = \frac{1}{2}[v(k) + v(k+1)]. \tag{6}$$

Since $c_k + 1 \geq c_k$, g_k here approximates a Gaussian-type nonlinearity centered around $v(k)$, and this allows for a localized effect at the output of the nodes g_k with the appropriate choice for each slope s_k.

To aid in satisfying specifications (ii) and (iii) for the output of the neural network, requirements relating the neural network's output to its parameters are made. First, the widths $\mathbf{s}$ are chosen such that (1) can be approximated by the following weighted sum when $u \in [v(j), v(j+1)]$:

$$z_i(u) \cong \sum_{k=j}^{j+1} W_{ik}\, g_k(u). \tag{7}$$

This implies that the tails of the outputs $g_k(u)$ for $k \neq j$ and $k \neq j+1$ are small compared to those for $k = j$ and $k = j+1$ when $u \in [v(j), v(j+1)]$.

The approximation of (7) also implies that when the node g_k has a larger response compared to the other nodes, the input is closest to the kth training pattern. In other words, for $1 \le f \le h + 1$ and for $u \in [v(k) - \Delta_{k-1}, v(k) + \Delta_k]$ where Δ_k is defined such that $g_k(v(k) + \Delta_k) = g_{k+1}(v(k) + \Delta_k)$,

$$g_k(u) \ge g_f(u). \tag{8}$$

Via (8), the localized effect of the hidden layer neurons is shown. The output of the neural network passing through a box specified by specification (ii) assumes the general shape of the output of the kth hidden layer node if $\varepsilon_{k-} \le \Delta_{k-1}$ and $\varepsilon_{k+} \le \Delta_k$. To further aid in satisfying specifications (ii) and (iii), the slopes s are chosen such that for all u:

$$\sum_{k=1}^{h+1} g_k(u) = 1. \tag{9}$$

In conjunction with (7) and (8), (9) implies that for $u \in [v(k), v(k + 1)]$ the sum of the outputs of $g_k(u)$ and $g_{k+1}(u)$ is constant, the and node g_k contributes more to the sum when u is close to $v(k)$ and less when it is closer to $v(k + 1)$. Thus, the choice of the slope s_k clearly affects the shape of the interpolation curve and the localization properties of Gaussian-type nodes g_k.

With the centers and the slopes specified for the hidden layer, the weights for the linear output layer are chosen next. Let $\boldsymbol{G}$ denote a p-by-$(h + 1)$-dimensional output matrix of the hidden layer nodes for each of the p operating points. Thus, the output layer weights are found by solving the following linear system of equations:

$$\boldsymbol{GW} = \boldsymbol{D}. \tag{10}$$

Since $\boldsymbol{G}$ is square ($h = p - 1$) and nonsingular because of the particular choices for $\boldsymbol{c}$ and $\boldsymbol{s}$, (10) has a unique solution $\boldsymbol{W}$. Furthermore, due to the choice of s per (7) and (8), if the neural network's input u is exactly the kth operating point, the output of the kth hidden layer node $g_k(u)$ is 1 while the outputs of the other hidden layer Gaussian-type nodes are 0. Hence, $\boldsymbol{G}$ can be assumed to be the identity matrix, and $\boldsymbol{W}$ can be approximated by $\boldsymbol{D}$.

Using (7), (8), and (10) and the simplifying assumption that $n = 1$, it was shown in [1] that to satisfy specification (ii) for all operating points, the slope s_k needs to be chosen such that the following are met for $1 \le k \le h$:

$$s_k \ge (v(k + 1) - \varepsilon_{(k+1)-} - c_k)^{-1} \ln[\min\{A(k), B(k)\} - 1] \tag{11}$$

and

$$s_k \ge (v(k)+\varepsilon_{k+}-c_k)^{-1} \ln[\min\{C(k), D(k)\} - 1], \tag{12}$$

where

$$A(k) = \frac{w_{k+1} - w_k}{d(k + 1) - \gamma_{(k+1)-} - W_k}, \tag{11a}$$

$$B(k) = \frac{w_{k+1} - w_k}{d(k + 1) - \gamma_{(k+1)-} - W_k}, \tag{11b}$$

$$C(k) = \frac{w_{k+1} - w_k}{d(k) - \gamma_{k-} - w_k}, \tag{12a}$$

and

$$D(k) = \frac{w_{k+1} - w_k}{d(k) + \gamma_{k+} - w_k} \tag{12b}$$

If $d(k) = d(k + 1)$, then specification (iii) requires $z_i(u) = d(k) = d(k + 1)$ for $u \in [v(k), v(k + 1)]$, and the solving of (11) and (12) for $u = v(k + 1) - \varepsilon_{(k+1)-}$ and $u = v(k) + \varepsilon_{k+}$ is unnecessary.

Thus, the neural network parameters can be chosen to exactly satisfy the three specifications for the interpolation between the training points. With $h = p - 1$, the centers $\boldsymbol{c}$ are chosen per (6) and the weights $\boldsymbol{W}$ can be approximated with $\boldsymbol{D}$. Using (11) and (12), limits for the widths s are found. Various selections for s can be made based on (11) and (12) and the corresponding $\boldsymbol{W}$ can be recomputed via (10) until a desirable curve is achieved. The resulting neural network describes a function which is within the class of functions described by $\theta(u)$ and exactly satisfies the specifications for the interpolation between the training points.

Due to the way in which the neural network parameters are chosen, the behavior of the neural network is expressed in (1)-(3). However, the neural network can be reconfigured in a more economical fashion, and (1) and (2) are equivalent to the following where $1 \le i \le n$ and $\sigma(u, c, s)$ is defined in (3):

$$z_i(u) = \sum_{k=1}^{h} (w_{i,k+1} - w_{ik})\, \sigma(u, c_k, s_k) + w_{i1}. \tag{13}$$

Due to the choice of $\sigma(u, c_0, s_0)$ in (4) and $\sigma(u, c_{h+1}, s_{h+1})$ in (5), when the neural network's input is outside the design domain, the neural network's output is equal to either the first or last desired output from the training set. In other words, if $u < v(1)$, $z(u) = \boldsymbol{d}(1)$, or if $u > v(p)$, then $z = \boldsymbol{d}(p)$. If instead it is desired that if $u << v(1)$ or $u >> v(p)$ then $z(u) = \boldsymbol{0}$, the choice of $\sigma(u, c_0, s_0)$ and $\sigma(u, c_{h+1}, s_{h+1})$ can be modified to accomplish this.

Multi-Input Neural Network

Given a specific set of training patterns and the three generalization specifications, the design of a neural network to approximate a multi-input-multi-output function is presented in this section. For the training patterns, let the m-dimensional vector $\boldsymbol{v}(j)$ denote the jth input pattern for $1 \le j \le p$, and let the scalar $v_r(j)$ be the rth component of $\boldsymbol{v}(j)$ for $1 \le r \le m$. (Note that it is not required to arrange these in an order as with the single-input case.) Let the n-dimensional vector $\boldsymbol{d}(j)$ denote the jth output pattern for $1 \le j \le p$, and let the scalar $d_i(j)$ denote the ith component of the jth desired output pattern for $1 \le i \le n$ and $1 \le j \le p$. Let the p-by-n-dimensional matrix $\boldsymbol{D}$ denote the matrix of desired outputs. Thus, the p pairs $\{\boldsymbol{v}(j), \boldsymbol{d}(j)\}$ are the given training patterns.

For the m-dimensional vector $\boldsymbol{u}$, let $\phi(\boldsymbol{u})$ be the n-dimensional output of a function approximating the relationship described by the training set $\{\boldsymbol{v}(j), \boldsymbol{d}(j)\}$ for $1 \le j \le p$. Let $\phi_i(\boldsymbol{u})$ denote the ith component of the functions output for $1 \le i \le n$. Three specifications are made on the function ϕ:

(i) If $\boldsymbol{u} = \boldsymbol{v}(j)$, then $\sigma(\boldsymbol{u}) = \boldsymbol{d}(j)$.

(ii) If $u_r \in [v_r(j) - \varepsilon_{rj-}, v_r(j) + \varepsilon_{rj+}]$ and $u_r \ne v_r(j)$, then $\phi_i(u) \in [d_i(j) - \gamma_{ij-}, d_i(j) + \gamma_{ij+}]$.

(iii) If $\boldsymbol{u}$ is "between" any two immediate training patterns, $\boldsymbol{v}(j)$ and $\boldsymbol{v}(k)$, then $\phi_i(\boldsymbol{u})$ is "between" the corresponding output patterns $d_i(j)$ and $d_i(k)$.

The meanings of all the specifications are the same as for the single-input case. However, descriptive language is used to state specification (iii) for the multi-input case since the indexing of the higher dimension patterns becomes complex and may not be needed, as is explained below.

For the neural network, let the n-dimensional vector $\boldsymbol{u}$ denote the neural network's input, and let z denote the neural network's n-dimensional output. For a specific input $\boldsymbol{u}$, let $z_i(\boldsymbol{u})$ for $1 \le i \le n$ and $\boldsymbol{z}(\boldsymbol{u})$ denote the output of the neural network. For the multi-input neural network, an individual output of the neural network is given by the following where $1 \le i \le n$:

$$z_i(\boldsymbol{u}) = \sum_{k_1=1}^{h_1+1} \cdots \sum_{k_m=1}^{h_m+1} w_{ik_1 \ldots k_m}\, g_{1k_1}(u_1) \ldots g_{m\,k_m}(u_m). \tag{14}$$

For $m = 1$, (14) reduces to (1). The output of the rk_rth hidden layer node is described by the following where $1 \le k_r \le h_r + 1$, $1 \le r \le m$, and where $\sigma(u, c, s)$ is defined in (3):

$$g_{rk_r}(u_r) = \sigma(u_r, c_{r(k_r-1)}, s_{r(k_r-1)}) - \sigma(u_r, c_{rk_r}, s_{rk_r}). \tag{15}$$

For $1 \le r \le m$ and for all u_r, let

$$\sigma(u_r, c_{r0}, s_{r0}) = 1 \tag{16}$$

and

$$\sigma(u_r, c_{r(h_r+1)}, s_{r(h_r+1)}) = 0. \tag{17}$$

Let $\boldsymbol{c_r}$ denote the h_r-dimensional vector of centers, and let $\boldsymbol{s_r}$ denote the h_r-dimensional vector of slopes for $1 \le r \le m$. Since the sizes of $\boldsymbol{c_r}$ and $\boldsymbol{s_r}$ are not restricted to be the same for $1 \le r \le m$, the formation of a matrix of centers and a matrix of slopes may not be possible. In (14), the outputs of the Gaussian-type nodes are multiplied together, which is unusual when compared to conventional neural networks. With the multiplications, the specification of the centers and the slopes for the sigmoid neurons corresponds to the specification of hyper-rectangles around the operating points. By appropriately choosing these parameters, the entire input space can be mapped to particular desired outputs of the neural network scheduler. However, a drawback is the added complexity of the neural network due to the required multiplications.

Design for the Multi-Input Neural Network

The selection of the multi-input neural network's centers, slopes, and weights is divided into the two cases of equidistant input patterns and nonequidistant input patterns with the case of equidistant input patterns considered first. The term "equidistant" for the m-dimensional case refers to the input patterns being an equal distance apart in each dimension. Instead of denoting the input patterns as $\boldsymbol{v}(j)$ for $1 \le j \le p$, the set of all equidistant input patterns is considered to be comprised of elements from vectors containing information on all the values of all the input patterns. Let p_r denote the number of values for the rth dimension such that

$$p = \prod_{r=1}^{m} p_r .$$

Let

$$\boldsymbol{v_r} = [v_{r1}, \ldots, v_{rp_r}]'$$

denote the p_r-dimensional vector of values of the input patterns for the rth dimension such that$v_{r(k_r+1)} \ge v_{rk_r}$ for $1 \le r \le m$ and $1 \le k_r \le p_r$. The counting of the p input patterns is not important; rather, their location in the input space is important. The number of hidden layer nodes in the rth dimension is set equal to one less the number of input patterns, that is $h_r = p_r - 1$, for $1 \le r \le m$. For $1 \le r \le m$ and for $1 \le k_r \le h_r$, the center c_{rk_r} is set equal to the distance halfway between the rk_rth and the $r(k_r + 1)$th input patterns

$$c_{rk_r} = \frac{1}{2}[v_{rk_r} + v_{r(k_r+1)}]. \tag{18}$$

Since

$$c_{r(k_{r+1})} \ge c_{rk_r} \quad \text{for } 1 \le r \le m,\ g_{rk_r}$$

approximates a Gaussian-type nonlinearity centered around v_{rk_r}, and this allows for a localized effect in the shape of a hyper-rectangle at the output of the node $\prod_{r=1}^{m} g_{rk_r}(u_r)$ with the appropriate choice for each slope s_{rk_r}.

In choosing the slopes to satisfy the specifications for the interpolation between the training patterns, the multidimensional equivalents of (7) to (9) need to be satisfied. For (7), this implies that for any input $\boldsymbol{u}$ the output is dependent only on the closest Gaussian-type nodes. For $1 \le f_r \le h_r + 1$ and for $u_r \, \varepsilon \, [v_{rk_r} - \Delta_{r(k_r-1)}, v_{rk_r} + \Delta_{rk_r}]$ where Δ_{rk_r} is defined such that

$$g_{rk_r}(v_{rk_r} + \Delta_{rk_r}) = g_{r(k_r+1)}(v_{rk_r} + \Delta_{rk_r})$$

for $1 \le r \le m$ and $1 \le k_r \le h_r$, the multidimensional equivalent of (8) is

$$g_{rk_r}(u_r) \ge g_{rf_r}(u_r). \tag{19}$$

Furthermore, the interpolation curve passing through a box described by specification (ii) assumes the general shape of the output of

$$\prod_{r=1}^{m} G_{rk_r}(u_r) \text{ if } \varepsilon_{rk_{r-}} \le \Delta_{r(k_r-1)}$$

and

$$\varepsilon_{rk_{r+}} \le \Delta_{rk_r}$$

for $1 \le r \le m$. For (9) and for all $\boldsymbol{u}$,

$$\sum_{k_1=1}^{h_1+1} \cdots \sum_{k_m=1}^{h_{m+1}} \prod_{r=1}^{m} g_{rk_r}(u_r) = 1. \tag{20}$$

With the centers and slopes chosen to satisfy (19) and (20), which aid in satisfying specification (iii), the weights $w_{ik_1 \ldots k_m}$ can be found. Forming the p-by-$[(h_1 + 1)\ldots(h_m + 1)]$ matrix $\boldsymbol{G}$ from the outputs of the Gaussian-type nodes, (10) is solved for the unique $[(h_1 + 1)\ldots(h_m + 1)]$-by-$n$ matrix $\boldsymbol{W}$. Furthermore, (11) and (12) can be extended to the multi-dimensional case with the appropriate changes to the indices. Thus, a neural network can be designed to satisfy the specifications for its generalization behavior in the multi-input case.

If the input patterns are not equidistant, two possible design choices are considered here: adding extra pseudo-input patterns such that equidistance occurs or locating the boundaries of the hyper-rectangles for each input pattern such that the entire input space is covered. The first possibility is more viable when most of the input patterns are equidistant, and only a few extra patterns are needed. Once this is accomplished, the above procedure for choosing the centers for equidistant input patterns is followed for the set of input patterns and added pseudo-input patterns. When solving (10), extra pseudo-desired outputs are added such that $\boldsymbol{W}$ is a unique solution. These added outputs can be chosen to be the same as nearby ones or extrapolated values from surrounding ones.

For the second possibility when the input patterns are not equidistant, hyper-rectangles are found for the given input patterns such that the entire input space is covered. Since the locations of the input patterns are no longer equidistant, (14) can be replaced by the following where $1 \le i \le n$:

$$z_i(\boldsymbol{u}) = \sum_{k=1}^{n} w_{ik}\, g_{1k}(u_1) \ldots g_{mk}(u_m) \tag{21}$$

For low dimensional inputs, the selection of the hyper-rectangles may be performed manually. However, for higher dimensions, this may become impractical. In [3], this problem is addressed as the "CR$_m$" problem (or "CR$_d$" using their notation). The CR$_m$ problem is known to have a high computational complexity, and Gonzalez presents algorithms for obtaining a suboptimal solution. Once the hyper-rectangles are found (either manually or algorithmically), the centers of the sigmoid neurons are chosen as the boundaries between them. The slopes are chosen to satisfy the specifications, and the output layer weights are found via (10). Thus, through a selection of the hyper-rectangles around the input patterns, the parameters for the neural network are chosen, and the specifications for the neural network's output are satisfied.

M. A. Sartori and P. J. Antsaklis, "Implementations of Learning Control Systems Using Neural Networks," I EEE C ontrol S ystems M agazine , in Special Issue on 'Neural Networks in Control Systems' Vol.12, No.2, pp. 49-57, April 1992.

Neural Network Implementations

In this section, the use of the neural network architecture introduced in the preceding section is described for the following four problems: dynamical system modeling, fault detection and identification, information extraction, and control law scheduling.

Dynamical System Modeling

The neural network proposed in this paper can be used to model the plant's or controller's dynamics. In [4], known linear information is used to assist in the identification of the plant, yet prior nonlinear knowledge may also be used in the plant identification process. With the proposed neural network architecture, certain types of knowledge can be implemented as a neural network. Specifically, if the the nonlinear information can be expressed within the framework of the training set and the three specifications for the interpolation between the training patterns, the design scheme proposed here can be used. Constructing a neural network to describe the known nonlinear behavior of the plant and using it as part of the plant identification process, a second neural network may be trained using learning algorithms to modify the plant's estimated dynamics. Such a scheme is shown in Fig. 3 and can be viewed as modeling the known dynamics with one neural network and training another neural network to learn the unknown dynamics. This general approach is discussed in [5]. By using a neural network to model the plant's known nonlinear behavior, it is anticipated that less time will be required to train the second neural network. In particular, the tens of thousands of back-propagation iterations that are normally required to train such a network may be reduced.

Further, by modeling the known nonlinear dynamics of the plant with a neural network, a total neural network plant identification structure is formed. This has the advantage of being able to be incorporated into the training of a neural network controller, which is one of the purposes of training a neural network plant estimator as described in [5] and as used in [4]. This controller training scheme is shown in Fig. 4. Since the desired plant input is unknown, the desired output of the neural network controller is unknown. However, the desired output of the plant is known. Substituting the actual plant with a neural network plant estimator, a multi-layer neural network may be trained with the back-propagation algorithm, or another gradient descent algorithm to minimize a plant output error cost function. Via the chain rule, the derivative of the cost function with respect to the plant estimator's input is used in the computation of the derivative of the cost function with respect to the neural network controller's weights. Next, in reference to Fig. 4, this gradient is computed assuming a SISO plant for simplicity.

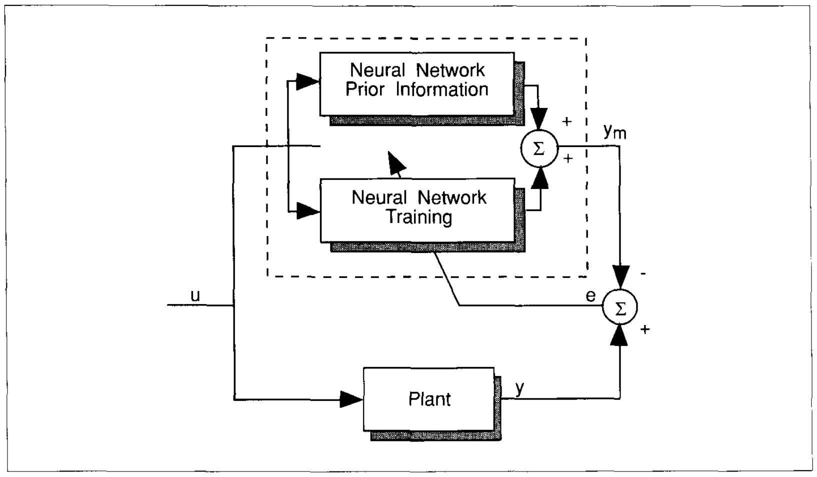
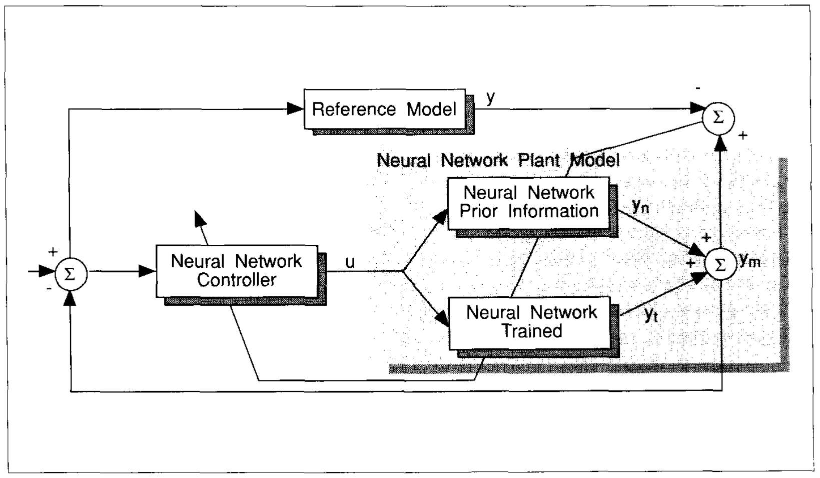


Fig. 3. Training a neural network for plant modeling.


Reference Model
Neural Network Plant Model
Neural Network Prior Information
Neural Network Trained
Neural Network Controller
u
y
yn
yt
ym
Σ
+
-
Fig. 4. Training a neural network controller.

The cost function is a sum of the squares of the plant output errors for the p training patterns:

$$F = \sum_{j=1}^{n} F(j) = \frac{1}{2} \sum_{j=1}^{n} (y(j) - y_m(j))^2. \tag{22}$$

To compute the weight change in the neural network controller, $\delta F/\delta u$ is required. So,

$$\frac{\delta F(j)}{\delta u} = y(j) = \left(y_n(j) - y_t(j) \right)\left(- \frac{\delta y_n(j)}{\delta u} - \frac{\delta y_t(j)}{\delta u} \right), \tag{23}$$

$$\frac{\delta y_n(j)}{\delta u} = \sum_{k=1}^{h+1} w_k \frac{\delta gk(u)}{\delta u}, \tag{24}$$

$$\frac{\delta g_k(u)}{\delta u} = \frac{\delta\sigma(u, c_{k-1}, s_{k-1})}{\delta u} - \frac{\delta\sigma(u, c_k, s_k)}{\delta u}, \tag{25}$$

and

$$\frac{\delta\sigma(u, c, s)}{\delta u} = \delta\sigma(u, c, s)(1 - \sigma(u, c, s)). \tag{26}$$

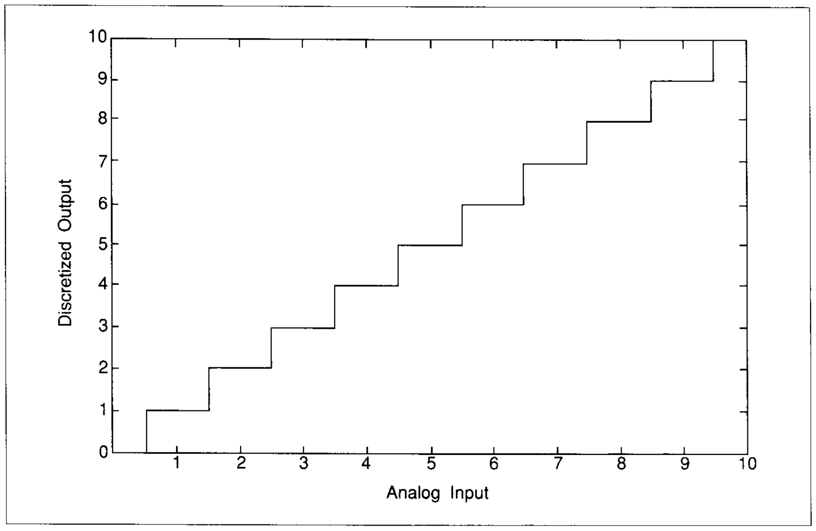


Fig. 5. Neural network output of the A/D converter of Example 1.

Computing $\delta y_n(j)/\delta u$ and assuming $\delta y_t(j)/\delta u$ is determined, the gradient $\delta F/\delta u$ is found. Relaxing the assumption of a SISO plant, the extension to the multi-input case is straight forward since each g_{rk} , in (14) is dependent on only one input and not the entire input vector. Thus, the neural network architecture and design scheme proposed in this paper can be used both to implement a plant estimator and train a neural network controller.

Fault Detection and Identification

The described neural network architecture may be used for fault detection and identification. As described in [1], [5], the neural network in this case operates outside the main loop and receives the appropriate signals from the plant: for instance, outputs, inputs, or environmental conditions. The neural network then responds by declaring either a fault, no-fault condition, or, with the neural network design proposed here, a partial membership signal. If fault patterns are known to occur for specific patterns, this information can be stored in the neural network by choosing the training set of the neural network to coordinate with the known faults. Valid fault regions for each fault can next be determined and stored using the centers $\boldsymbol{c}$ of the sigmoid nonlinearities. By choosing the slope of the sigmoid nonlinearities small, a "gray scale", or partial membership, of the fault region can be included in the fault detection system design. By choosing the slope of the sigmoid nonlinearities large, clearly definable fault and no-fault regions develop.

A drawback of this approach is that for multi-input failure detection systems, the valid fault regions are restricted to hyper-rectangles. If irregular shaped fault regions defined by straight lines are desired, the method initially described in [6] and further developed in [1] may be employed. If it is desired that the neural network failure detection system detect, identify, and diagnose the failure, the scheme described in [7], which is illustrated using a JPL space antenna model, is one possible approach.

Information Extraction

The neural network architecture described in this paper may also implement the general control function of numeric-to-symbolic conversion. As considered here, numeric-to-symbolic conversion is the task of processing data that can be used by the controller as information. One such example of numeric-to-symbolic conversion is fault detection and identification as discussed in the previous subsection. Here, the numeric data consists of the detection system's inputs: the plant's inputs and outputs, their derivatives, and the environmental conditions. The symbolic data is the occurrence of a fault or no-fault condition and the identification of the fault.

Another example, as discussed extensively in [6], is the task of converting numeric data to a form usable by a discrete event controller. The discrete event controller requires symbolic information describing the state of the plant. However, the output of the plant is in a numeric form, unusable to the discrete event controller. By describing the regions of interest to the discrete event control law, the boundaries of the regions are transformed into design specifications for the centers of the neural network's sigmoid functions. By changing the slopes of the sigmoid functions, membership in a region can be made specific (i.e., a large slope) or non-specific (i.e., a small slope) depending on the type of information required by the discrete event control law.

For a digital controller, the neural network architecture proposed here can perform the function of analog-to-digital (A/D) conversion. The centers of the sigmoid function are chosen to correspond to the quantization regions. For a clear distinction between regions, the slopes are chosen larger. This feedforward neural network design can be compared to the feedback neural network design of Tank and Hopfield in [8] and to its improved design described in [9]. In comparing the feedforward design to the feedback design, no time is required to wait for the output of the feedforward neural network to settle, as is required for the feedback neural network design. The problems with determining and designing the regions of attraction for the neural network's minima in the feedback design do not occur whatsoever with the feedforward design; the "regions of attraction" are known exactly and can be easily designed according to specifications. Comparing the number of neurons required for an n-level A/D converter, the feedback design requires n neurons, while the feedforward design requires $3n$ neurons. Next, an A/D converter is implemented with a neural network designed using the method proposed here.

Example 1

It is desired to convert analog signals in the range of 0 to 10 to all 11 discrete levels between 0 and 10. In addition, any signal less than 0 is mapped to 0, and any signal greater than 10 is mapped to 10. The discrete regions are divided midway between the integer values. The training set becomes the 11 matched pairs of the input and output integer values. Using the single-input neural network design procedure in this paper, $h = 10$ and form (6),

$$c = [0.5\ \ 1.5\ \ 2.5\ \ 3.5\ \ 4.5\ \ 5.5\ \ 6.5\ \ 7.5\ \ 8.5\ \ 9.5]'.$$

The slopes for the hidden layer neurons are chosen large: $s_k = 500$ for $1 \leq k \leq 10$. Solving (10), the output layer weights are determined. The output of the resulting neural network is shown in Fig. 5. The transitions between the regions are sharp due to the high gain chosen for the sigmoid nonlinearities.

Control Law Scheduling

A neural network may be used to implement a control law scheduler. In this capacity, the neural network can be viewed as a high level decision maker operating outside the conventional control loop to provide a higher degree of autonomy to the system [1],[5],[10]. Given a set of operating points and the associated set of parameters values for the control law, the neural network is designed to satisfy given specifications for the interpolation between the provided operating points. The operating points become the training set, and the specifications for the interpolation between the operating points become the three specifications discussed in the design of the neural network.

In its operations as a control law scheduler, the neural network is first supplied with information about the system and its environment; it then produces control law switching information to the controller. The neural network's inputs are the inputs and outputs of the plant together with the reference signal. The output of the neural network is the control law adjustment signal sent to the controller; in this paper, this signal represents parameter values for the controller. The neural network's inputs are not restricted to these signals; other signals such as the plant's states, derivatives, environmental conditions, or delayed values of any of these can be used as inputs to the neural network. Basically, any signal that may be designed into the operation of the scheduler is used as an input to the neural network. Furthermore, the plant and controller can operate in either continuous or discrete time. If the neural network is implemented in analog hardware, both the plant and the controller can operate in continuous time; in this case, the neural network may supply the parameter values to the controller as continuous variables. However, if the neural network is to be implemented in software, both the plant and the controller need to be discrete or discretized versions of continuous ones. In either case, the designing of the neural network as discussed in this paper remains unchanged.

Table I
Initial Disturbances and Parameter Sets

Amplitude	σ_1	σ_2	L	$\bar{L}$
2.0	0.093	10.05	49000.0	119703.96
2.25	0.213	7.04	39200.0	131674.35
2.5	0.302	7.35	44046.1	145910.16
3.0	0.307	15.79	37325.7	183327.84
3.5	0.876	9.556	44046.1	175092.19
4.0	0.808	10.48	29114.0	203493.90
4.5	1.767	10.48	32025.4	203493.90
5.0	3.924	8.70	48450.7	140073.75
5.5	6.928	7.73	41567.5	138395.55
6.0	11.08	10.05	41567.5	138395.55
7.0	14.41	19.10	41567.5	110716.44

Two examples, one for a single-input neural network and one for a multi-output neural network to implement, are presented next to illustrate the design of a neural network implemented control law scheduler.

Example 2

In [11], a parameter learning method is presented and used to define the region of operation for an adaptive control system of a flexible space antenna. In one of the experiments described, an initial pulse disturbance is applied to the plant, and the adaptive controller is required to follow a zero-order reference model. The goal of the parameter learning system is to find and store values for the four adaptive controller parameters (σ_1, σ_2, L, and $\bar{L}$,) for varied amplitudes of a pulse disturbance (system initial conditions) such that a defined performance index based on the output of the plant is minimized. In Table I, the values found for the controller parameters for different pulse amplitudes are repeated. Using this table, the goal here is to construct a neural network scheduler such that a smooth interpolation is achieved between the 11 operating points using the design method of this paper.

For the neural network, the centers of the sigmoid neurons in the hidden layer are set halfway between the operating points according to (6):

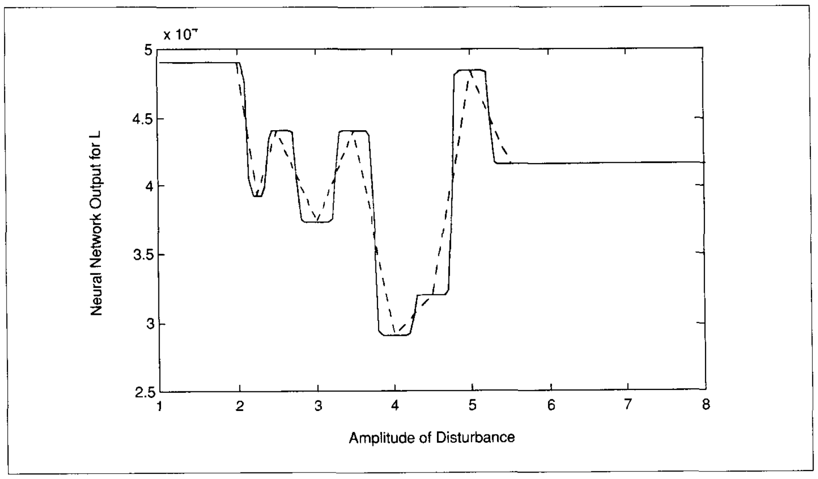


Fig. 6. Neural network output for L for Example 2.

Table II
Selected Flight Points for F100 Engine

Amplitude (1K ft)	Mach Number	Controller Parameters
10	0.75	2
10	1.00	3
10	1.25	4
20	0.50	1
30	0.75	5
30	1.00	6

$$c = [2.125\ 2.375\ 2.75\ 3.27\ 3.75\ 4.25\ 4.75\ 5.25\ 5.75\ 6.5]'.$$

The slopes for the hidden layer neurons are chosen as $s_k = 75$ for $1 \le k \le 10$. With (10), the unique weights W are found. Fig. 6 shows the output of the sigmoid neural network scheduler for adaptive controller parameter L along with the straight line approximation between the operating points (dotted line). As can be seen, $z_3(k) = d_3(k)$ for $1 \le k \le 11$, and specification (i) is met. In the regions nearby the operating points, the adaptive controller parameter values specified by the neural network scheduler are close to those specified by Table I satisfying specification (ii) for very thin and wide boxes. In regions between operating points, a swift yet smooth transition occurs between the operating points, and specification (iii) is met. Between 5.5 and 6.0, specification (iii) requires a straight line, and this is clearly satisfied. Thus, all specifications can be shown to be satisfied.

Example 3

In [12], linear models of a F100 engine are developed for various flight points based on altitude and mach number. In Table II, 6 of these flight points are listed with fictitious controller parameters, and in Fig. 7, these six are diagrammed. It is desired to design a neural network control law scheduler for the interpolation between these flight points using the architecture and design approach of this paper. Since the operating points are non equidistant, it is decided to add 6 pseudo-operating points such that equidistance is achieved, and $p = 12$. Now,

$$V' \begin{bmatrix} 10 & 10 & 10 & 10 & 20 & 20 & 20 & 20 & 30 & 30 & 30 & 30 \\ 0.50 & 0.75 & 1.00 & 1.25 & 0.50 & 0.75 & 1.00 & 1.25 & 0.50 & 0.75 & 1.00 & 1.25 \end{bmatrix}$$

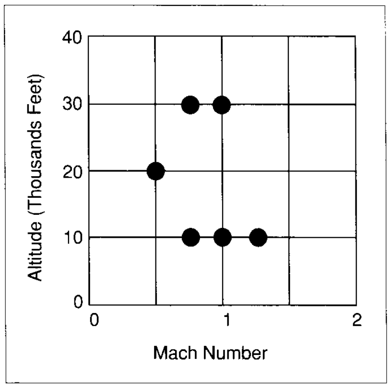


Fig. 7. Selected F100 flight points for Example 3.3.

and

$$d = [1\ 2\ 3\ 4\ 1\ 1\ 1\ 1\ 1\ 5\ 6\ 6]'.$$

The resulting vectors of operating points are

$$v_1 = [10\ 20\ 30]'$$

and

$$v_2 = [0.50\ 0.75\ 1.00\ 1.25]',$$

where $r = 1$ corresponds to the altitude and with $r = 2$ corresponds to the Mach number. The equivalent to (16) for this example is

$$z(u) = \sum_{k_1=1}^{2} \sum_{k_2=1}^{3} w_{k_1k_2}\, g_{1k_1}(u_1)\, g_{2k_2}(u_2).$$

Using (20), the vectors of centers are

$$c_1 = [15\ \ 25]'$$

and

$$c_2 = [0.625\ 0.875\ 1.125]'.$$

The slopes are chosen as $s_{1k_1} = 20$ for $1 \le k_1 \le 2$ and $s_{2k_2} = 400$ for $1 \le k_2 \le 3$. The weights w are found by solving (10), and the interpolation curve produced is shown in Fig. 8. The lowest corner corresponds to the point (8, 0.4), the most left corner to (32, 0.4), and the most right corner to (8, 1.35). Specifications (i) and (iii) are clearly satisfied, and specification (ii) is satisfied for small thin boxes for a rectangular region around the operating points.

Instead of adding extra operating points such that equidistance is achieved, it is decided to hand pick the generalization regions for the interpolation curve, which are shown in

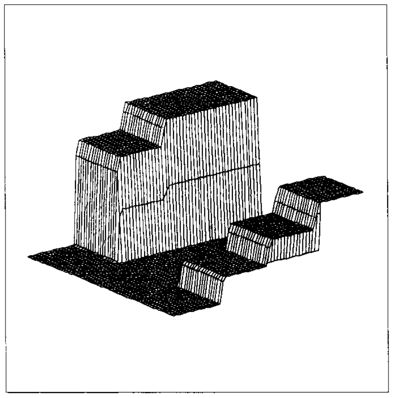


Fig. 8. Neural network output for the first scheduler of Example 3.

Fig. 9 and are assigned values according to Table II. The equivalent to (21) for this example is

$$z(u) = \sum_{k=1}^{6} w_k\, g_{1k}(u_1\, g_{2k}(u_2).$$

Only 4 sigmoids are needed to form the 6 regions, and the vectors of centers are

$$c_1 = [20]$$

and

$$c_2 = [0.625\ 0.875\ 1.125]'$$

where $r = 1$ corresponds to the altitude and $r = 2$ corresponds to the Mach number. By choosing $s_{1k} = 20$ and $s_{2k} = 400$ for $1 \le k \le 6$ and by solving (10) for w, the interpolation curve formed is shown in Fig. 10 with the same corner coordinates as the previous two-

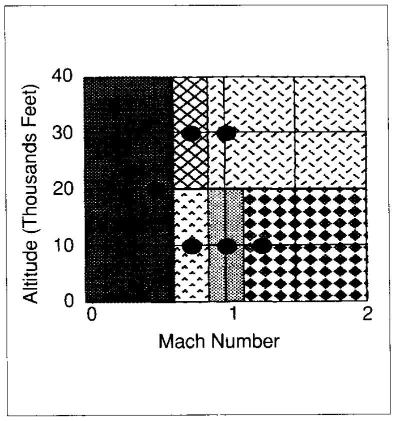


Fig. 9. Selected localized regions for the F100 flight points of Example 3.

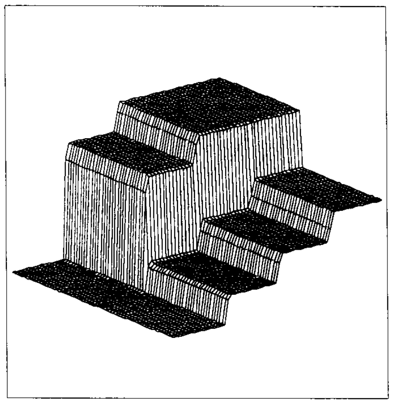


Fig. 10. Neural network output for the second scheduler of Example 3.

dimensional plots. Specifications (i) and (iii) are clearly satisfied, and specification (ii) is satisfied for small thin boxes for rectangular regions around the operating points.

Concluding Remarks

A particular neural network and a systematic design methodology are introduced so that prior information about the system's behavior can be directly and easily incorporated into the control design. The four uses investigated for the proposed neural network architecture are dynamical system (plant and controller) modeling, fault detection and identification, information extraction, and control law scheduling. Another approach to address this problem is presented in [13] using neural networks with Gaussian nonlinearities and not sigmoid nonlinearities. The two methods are compared in [1], [2], and it is shown that the sigmoid neural network implementation has certain advantages over the Gaussian neural network implementation. In particular, the sigmoid neural network easily implements training patterns that are nonequidistant, which is a problem for the Gaussian neural network approach. However, for the multi-input case, the sigmoid neural network has an added complexity due to the multiplication in (14).

References

[1] M.A. Sartori, "Feedforward neural networks and their application in the higher level control of systems," Ph.D. diss., Dept. Elec. Eng., Univ. Notre Dame, Apr. 1991.

[2] M.A. Sartori and P.J. Antsaklis, "Neural network implementations for control scheduling," Tech. Rep. #91-04-02, Dept. Elec. Eng., Univ. Notre Dame, Apr. 1991.

[3] T.F. Gonzalez, "Covering a set of points with fixed size hypersquares and related problems, in *Proc. 1990 Annual Allerton Conf. Communication, Control, and Computing*, pp. 838–847, 1990.

[4] K.S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, Mar. 1991.

[5] P.J. Antsaklis and M.A. Sartori, "Neural networks in control systems," *Systems and Controls Encyclopedia, Supplement II*, to be published.

[6] Passino K.M., Sartori M.A., and P.J. Antsaklis, "Neural computing for numeric to symbolic conversion in control systems," *IEEE Control Syst. Mag.*, Apr. 1989, pp. 44–52.

[7] P.J. Antsaklis and M.A. Sartori, "Autonomous control of large spacecraft using neural networks," Final Rep., Jet Propulsion Laboratory Contract 957856 Mod. 4, Nov. 1991.

[8] D.T. Tank and J.J. Hopfield, "Simple "neural" optimization networks: an a/d converter, signal decision circuit, and a linear programming circuit", *IEEE Trans. Circuits Syst.*, vol. 33, no. 5, pp. 533–541, May 1986.

[9] D.L. Gray, "Synthesis procedures for neural networks," Master's Thesis, Dept. Elec. Computer Eng., Univ. Notre Dame, Notre Dame, IN, July 1988.

[10] P.J. Antsaklis, K.M. Passion, and S.J. Wang, "An introduction to autonomous control systems," *IEEE Control Syst. Mag.*, vol. 11, no. 4, pp. 5–13, June 1991.

[11] M.D. Peek and P.J. Antsaklis, "Parameter learning for performance adaptation," *IEEE Control Syst. Mag.*, vol. 10, no. 7, pp. 3–11, Dec. 1990.

[12] R.D. Hackney and R.J. Miller, "Engine criteria and models for multivariable control system design," in *Proc. 1977 Int. Forum on Alternatives for Multivariable Control*, pp. 14–28, 1977.

[13] M.A. Sartori and P.J. Antsaklis, "A Gaussian neural network implementation for control scheduling," in *Proc. 1991 IEEE Int. Symp. Intelligent Control*, Aug. 13–15, 1991.

Michael A. Sartori received the B.S., M.S., and Ph.D. degrees in electrical engineering from the University of Notre Dame in 1987, 1989, and 1991, respectively. His Ph.D. dissertation addressed the training of feedforward neural networks and their application to the higher level control of systems. He worked for the McDonnell Douglas Electronics Company during the summers of 1986 and 1987 and for the McDonnell Douglas Missile Systems Company during the summer of 1989. During the summer of 1991, he was a Post Doctorate Research Associate at the University of Notre Dame. He is now employed by the U.S. Navy's David Taylor Research Center. His research interests include neural networks, image processing, and autonomous systems.

Panos J. Antsaklis received the diploma in mechanical and electrical engineering from the National Technical University of Athens, Greece, in 1972, and the M.S. and Ph.D. degrees in electrical engineering from Brown University, Providence, RI, in 1974 and 1977, respectively. After holding faculty positions at Brown University, Rice University, and Imperial College, University of London, he joined the University of Notre Dame where he is currently Professor of Electrical Engineering. His research interests have been in multivariable system and control theory, primarily using the differential operator and fractional representations, more recently also in autonomous intelligent control systems, and in particular in discrete event systems and neural networks, in adaptive and learning control, and in the reconfigurable control of systems. He is currently an elected member of the Board of Governors in the IEEE Control Systems Society and the Group Leader of the Working Group on Control Systems in the Technical Committee on Intelligent Control. He was also the Program Chair of the 30th IEEE Conference on Decision and Control, which took place in the United Kingdom in December 1991. He has served as an Associate Editor of the *IEEE Transactions on Automatic Control* and as Chair of the Technical Committee on Theory of the IEEE Control Systems Society. He is an Editor of the IEE Control Engineering Book Series, and an Associate Editor of the *IEEE Transactions on Neural Networks*, having been founding Associate Editor for Letters. He has also served as the Guest Editor for Neural Networks for *IEEE Control Systems Magazine*. He is an IEEE Fellow.