Comparing Two Neural Networks for Pattern Based Adaptive Process Control

Douglas J. Cooper, Lawrence Megan, and Ralph F. Hinde, Jr.
Chemical Engineering Dept., University of Connecticut, Storrs, CT 06269

An adaptation strategy based on an analysis of patterns exhibited in the recent controller error history is presented. The strategy is a two parameter adaptation, where the gain and time constant of the controller's internal model are adjusted to make the closed loop error response match a target pattern. Both a back-propagation network and a vector quantizer network (VQN) are compared as pattern analysis tools. This strategy is established for a number of model based controllers and is demonstrated here using the Generalized Predictive Control algorithm. Details of this set point tracking strategy are presented along with demonstrations on both simulated and real single loop processes that experience significant changes in process gain and time constant. Results show both networks to be equally capable at pattern recognition with the VQN's ease of training and implicit ability to assess the accuracy of the pattern match as deciding factors in network selection.

Introduction

Pattern based control strategies determine parameter adaptation from an analysis of patterns exhibited in recent sampled data history. Bristol (1977) describes a strategy where rules and algorithms are employed to analyze every controller error transient pattern for extreme values or error peaks. When three peaks are found or sufficient time has elapsed, the peak values are used to compute features characteristic of controller performance such as overshoot and damping. The difference between these computed pattern feature values and operator specified target values are then used to adjust the tuning of a PID controller.

In a different pattern based strategy (Cooper and Lalonde, 1990; Lalonde and Cooper, 1989), rules and algorithms are employed to analyze the recent history of the manipulated process input variable. The patterns in the process input are used to determine if recent manipulations are providing process excitation to an extent that dominates process noise. When sufficient excitation is diagnosed, the dynamically rich data are regressed upon a process model internal to the controller.

This work explores the use of artificial neural networks (ANN's) in pattern based adaptation strategies, where the patterns in question are those exhibited in the recent controller error history following sustained set point changes. ANN's are well suited for pattern analysis tasks (Jau et al., 1989; Pao, 1989), and thus hold promise as an improved framework for pattern based adaptive strategies. ANN's permit analysis of sampled data patterns as complete pictures over a specified time frame. As a result, they can reliably recognize patterns even if the data are corrupted with random error or other irregularities.

Back-propagation networks (BPN's) have been the focus of the majority of ANN research and thus are one of the architectures considered here. A properly designed BPN can classify unfamiliar input patterns after having been trained on a small, carefully chosen subset of the entire pattern space. This characteristic makes BPN's an efficient tool by which to analyze the virtually limitless number of potential error history patterns.

The second ANN architecture studied in this work is a vector quantizer network (VQN), similar to that described by Kohonen (1988, 1982). The VQN partitions the pattern space into a series of discrete classifications and once implemented online matches a given error pattern to the most representative classification. In contrast with Kohenen's network, which self-determines and self-organizes the pattern classes, the designer of the VQN predetermines and organizes the pattern classes. This work enhances the VQN structure by determining the winning class with a method known as the local energy maximum.

Correspondence concerning this article should be addressed to D. J. Cooper.

A pattern based adaptation strategy is detailed where, for comparison, each ANN is used for the same pattern analysis task. The strategy is a two parameter adaptation where the adjustable parameters are the gain and time constant of a first-order process model that is internal to the controller. Following a set point change in the given process, the strategy locates the resulting error pattern within a pattern space. The location of that error pattern with respect to a desired or target pattern is used to determine the necessary model parameter adjustments needed to restore desired performance. This work is limited to single-loop applications that have no true dead time and experience significant changes in process gain and time constant.

The strategy is general to a number of model based controllers, and in this work set point tracking is demonstrated using the Generalized Predictive Control (GPC) algorithm (Clarke et al., 1987). These demonstrations include both nonlinear simulated control processes and a pilot-scale application. For further comparison, one of the set point trajectories is demonstrated using a recursive least squares (RLS) model updating algorithm modified with covariance resetting.

In conclusion, issues pertinent to the design and implementation of this pattern based adaptive strategy are discussed. In particular, a comparison of the relative strengths and weaknesses of the two ANN's for pattern analysis as required for implementation of the two parameter adaptation strategy is presented.

Pattern Space Development

When a process responds to a set point change with a large overshoot followed by slowly damping oscillations, one possible explanation is that the gain and/or time constant of the controller model is small relative to that of the actual process. Alternatively, an explanation for a slow response after a set point change is that the gain and/or time constant of the controller model is too large. Hence, the manner in which a poorly performing controller is tuned improperly can be inferred from the patterns displayed in the recent history of the controller error.

It is straightforward to design and implement either ANN architecture such that it can recognize both the oscillatory and nonoscillatory patterns that are associated with aggressive, desirable and sluggish controller performance. Given that the scope of this work limits the adaptation strategy to an adjustment of controller model gain and time constant, each of these error transient patterns must then be associated with a specific mismatch between the current values of these two parameters and those values which restore desired performance.

To develop such an association, a model based controller such as the GPC algorithm is implemented on a simulated linear second-order process. This process is chosen to be the ideal standard as the response of higher order processes, and thus their corresponding error patterns, may be sufficiently approximated with a second-order process if the sampling time is properly chosen (Åström, 1985). A complete sequence of controller error transients is then generated, including patterns associated with sluggish, desirable, and aggressive control performance.

The sequence of patterns is created by varying the gain and time constant of the simulated process in an orderly fashion

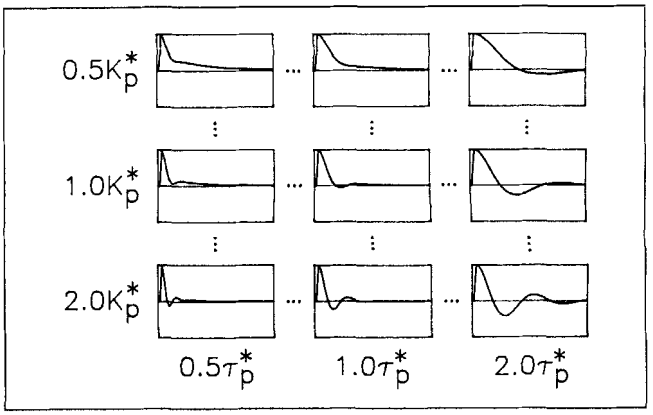

Figure 1. Pattern space of error transients.

and then forcing the closed loop system with a step change in set point. In this work, the range within which the gain, K_p, and time constant, τ_p, are varied is 0.1 to 10.0 times those values which provide desirable performance. This may be expressed as:

$$K_p = \delta_\kappa K_p^* \quad \tau_p = \delta_\tau \tau_p^* \tag{1}$$

where δ_κ and δ_τ are adjustment multipliers used to vary the gain and time constant of the simulated process and K_p^* is the initial process gain and τ_p^* is the initial time constant which provides desirable performance.

While a second-order process is chosen to develop the representative error patterns, the controller's internal model is chosen as first-order. A first-order model can provide robust performance over a wide range of higher ordered processes provided the two adjustable parameters, K_p and τ_p, are properly specified. Also, a controller model higher than first-order would require the on-line adaptation of more than two model parameters, thereby complicating the adaptive strategy. By specifying correct estimates of K_p and τ_p, the controller error response to set point changes has approximately a 5 percent overshoot and a 10 percent damping ratio, which is defined in this work as the desirable controller error pattern. Once on-line, any deviation from the desired pattern indicates the need for adaptation of the controller model parameters. The length of the data window for error pattern collection is chosen as 50 samples for both ANN implementations. Sample time, Δt, is specified as needed from process to process such that 50 samples following a set point change collects sufficient dynamic information about the process character and has a time scale similar to that of the patterns in the pattern space.

The result is an ordered pattern space that contains both oscillatory and nonoscillatory patterns and permits adjustments in the controller model parameters of up to one order of magnitude for each cycle of pattern analysis and model parameter adaptation. Each pattern corresponds to a pair of parameter adjustments, δ_κ and δ_τ. The ANN's are then trained to associate a given error pattern with the corresponding δ_κ and δ_τ used to create that pattern, and use this knowledge to update the model parameters in a manner which restores desired performance. A subset of this pattern space is shown in Figure 1.

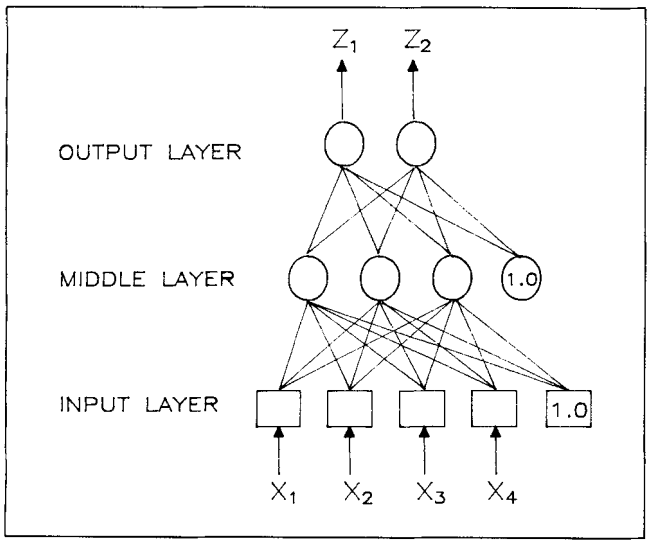

Figure 2. Back-propagation network structure.

BPN-Adaptive Strategy

The back-propagation network

The back-propagation network (BPN) is the most common of ANN architectures and is described extensively in the literature (Caudill, 1988; Rumelhart and McClelland, 1986). BPN's are composed of a highly interconnected set of processing units or neurons, which are arranged in a layered structure. An example is shown in Figure 2. In the classical architecture, the output of each neuron is sent exclusively to each neuron in the layer above it. While there can be many layers, most pattern recognition and classification tasks can be accomplished with three: one that receives and distributes the input pattern, one middle or hidden layer, and one which produces the output pattern. Also, as shown in Figure 2, BPN's typically contain a bias neuron in both the input and middle layer. Each bias neuron, which produces a constant output of 1.0, is fully connected to the layer above it but receives no signal from the preceding layer.

Each connection between two neurons has an associated weight. A neuron processes information by summing all the weighted inputs it receives and passing this sum through a nonlinear functionality to produce the output for that neuron. A popular choice for the nonlinear function is a continuous sigmoidal function with limits [0, 1] or [−1, 1]. This work requires the sigmoid to have limits [−1, 1], and the function used to generate this is:

$$f(x) = \tanh(x) \tag{2}$$

BPN training requires *a priori* knowledge of desired output patterns given a training set of input patterns. During training, the network is continually fed patterns from the training set, and the weights are repeatedly adjusted until the network produces the desired pattern at the output layer for each corresponding input pattern. Once the network is able to accomplish this, training is complete and the weights are fixed.

If properly designed, the BPN will not only be able to correctly evaluate patterns it has been trained upon, but will also be able to interpolate between these patterns to describe unfamiliar but related input patterns. It is the ability to develop functional relationships between the input and output patterns that allows the BPN to accurately evaluate a wide range of error patterns, provided all the patterns fall within the limits of the training set. Thus the BPN's in this work are trained on a sampling of the entire range of patterns and are trained to relate each pattern with the adjustments used to create that pattern, δ_κ and δ_τ. This knowledge is then used to adapt the model parameters in a manner which restores desired performance.

BPN training

During training, the weights are continually adapted in a manner to minimize the error between the desired and actual outputs of each pattern. The total error to be minimized is defined as:

$$E = \sum_{p=1}^{P} (\mathbf{d}^p - \mathbf{z}^p)^2 \tag{3}$$

where $\mathbf{d}^p$ is the desired output pattern for the pth input pattern, $\mathbf{z}^p$ is the actual output pattern based on the present values of the connection weights and P is the total number of training patterns. The output pattern may be a vector of output values as shown in Eq. 3 or just a single output value, depending on the application.

The standard algorithm for weight adaptation, known as the generalized delta rule (GDR), is discussed in detail in the literature (Caudill, 1988; Rumelhart and McClelland, 1986). GDR is a steepest descent algorithm in which the weights are adjusted after the presentation of each pattern. However, it has been shown (Leonard and Kramer, 1990; Goryn and Kaveh, 1989) that this is not a very robust algorithm, and often thousands of presentations of the complete training set are required to sufficiently minimize E.

The standard GDR training equation is given by:

$$\mathbf{w}_k = \mathbf{w}_{k-1} + \eta \mathbf{g}_k^p \tag{4}$$

where $\mathbf{w}$ is the vector of weights, k is the particular iteration, $\mathbf{g}^p$ is the steepest descent gradient calculated after every pattern presentation, equivalent to $-dE^p/d\mathbf{w}$, and η is a gain factor which controls the rate of convergence. Note that in this equation k refers to the presentation of one pattern and not the entire training set.

GDR is not robust for three primary reasons. First, it makes an adaptation in the weights following each pattern presentation. Thus the adaptation is based on E^p, the error from a single pattern presentation, instead of the total error E which is the true objective function to be minimized. While a given weight adaptation may be appropriate to minimize E^p, it is not necessarily appropriate for the overall error, E.

Secondly, GDR uses steepest descent to make the weight adaptations. It is well documented (Press et al., 1986; Edgar and Himmelblau, 1988) that steepest descent directions are not optimal, especially when faced with long, narrow valleys in the objective function. As steepest descent directions are required to be orthogonal to each other, if the initial search direction is not either parallel or perpendicular with a dominant

axis in the error surface, such directions may require many steps to reach the minimum of the objective function.

The third problem with GDR is that the gain factor, η, is held constant, and thus is not the optimal value for every step. While the chosen value of η may minimize E^p in one direction, it is unlikely that it will find the minimum of E^p in subsequent directions. Some practitioners will heuristically adjust η during training, while others have suggested adapting η based on the local gradient (Minai and Williams, 1990). However, there is still a heuristic component involved in either approach, and more importantly the chosen value of η does not guarantee to find the minimum along the given search direction.

Conjugate gradient method

To overcome the deficiencies of GDR, this work employs a more robust training algorithm known as the conjugate gradient (CG) method. CG fixes the weights for the entire presentation of the training set, and then adapts the weights based on the overall error, E. Also, as the name implies, CG uses conjugate directions to choose the new search direction, which are more efficient than steepest descent directions (Press et al., 1986; Edgar and Himelblau, 1988). Finally, CG optimizes the gain factor at every step such that the objective function is minimized along the given search direction.

The conjugate gradient method of network training is based on the standard two-step iterative procedure characteristic of multivariable minimization. The first step is to determine the current search direction from the present point in the independent variable space, and the second is to minimize the function in the chosen direction.

In training a BPN the independent variables are the vector of weights, $\mathbf{w}$, where the dimension of $\mathbf{w}$ is a function of the number of neurons. The function to be minimized, $f(\mathbf{w})$, is equal to E as defined by Eq. 3. To evaluate the function, the entire training set is passed through the network with $\mathbf{w}$ fixed, and the total error between desired outputs, $\mathbf{d}$, and the actual outputs, $\mathbf{z}$, is calculated via Eq. 3.

The CG method chooses the new search direction such that it is conjugate to the previous search direction. If n represents a complete presentation of the training set, the new search direction at iteration n, $\mathbf{s}_n$, satisfies the equation:

$$\mathbf{s}_n \mathbf{H} \mathbf{s}_{n-1} = 0 \tag{5}$$

where $\mathbf{H}$ is the Hessian matrix. The standard CG algorithm (Fletcher and Reeves, 1964) solves this equation for s_n to give the new search direction:

$$\mathbf{s}_n = \mathbf{g}_n + \gamma \mathbf{s}_{n-1} \tag{6}$$

where $\mathbf{g}_n$ is the steepest descent gradient based on the total error, $-dE/d\mathbf{w}$. The scalar parameter γ is then solved by the following equation (Polak, 1971):

$$\gamma = \frac{(\mathbf{g}_n - \mathbf{g}_{n-1})^T \mathbf{g}_n}{\mathbf{g}_{n-1}^T \mathbf{g}_{n-1}} \tag{7}$$

Upon selection of a new search direction, the weights are updated as:

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \lambda s_n \tag{8}$$

where λ is found by doing a one-dimensional line minimization in the given search direction.

Calculation of the steepest descent vector, $\mathbf{g}$, follows naturally by extension from the Generalized Delta Rule. The elements of $\mathbf{g}^p$ used in Eq. 4 are given by:

$$g_{ij} = \epsilon_j z_i \tag{9}$$

where the subscript j corresponds to a neuron in the layer above that described by i, and z_i is the output of neuron i sent to neuron j. The error term, ϵ_j, for output neurons, is:

$$\epsilon_j = f'(z_j)[d_j - z_j] \tag{10}$$

and for middle layer neurons is:

$$\epsilon_i = f'(z_i) \sum_j \epsilon_j w_{ij} \tag{11}$$

where d_j is the desired output from neuron j and $f'(z_j)$ is the derivative of the sigmoidal function evaluated at z_j (Rumelhart and McClelland, 1986).

Thus $\mathbf{g}^p$ is calculated at every pattern presentation in GDR. To extend this to CG, the calculation of $\mathbf{g}$ at iteration n is simply the summation of the elements over a complete presentation of the training set:

$$\mathbf{g} = \sum_{p=1}^{P} \mathbf{g}^p \tag{12}$$

where P is the total number of patterns in the training set.

The procedure operates in the following manner. The weights are initialized at random values, and following a complete pass of the training set through the network, the initial search direction, $\mathbf{s}_1$, is chosen to be equal to $\mathbf{g}_1$. The value of λ which minimizes E in the direction $\mathbf{s}_1$ is then found. The weights are then updated appropriately and the procedure repeated, with successive search directions found using Eq. 6. The procedure is terminated when E falls below a specified value or fails to be sufficiently minimized by successive iterations.

BPN adaptation

As previously stated, the goal of the adaptive algorithm is to adjust two parameters, model gain and time constant, based on the error pattern resulting from a set point change. The adaptations are based on BPN predictions of the adjustment multipliers, δ_κ and δ_τ, by which to adjust the current model parameters to obtain values which restore desired performance. To accomplish this goal, the obvious approach is to train a single BPN with two outputs, such that it simultaneously associates a particular error pattern with both a gain and time constant adjustment. However, the authors' experience indicates the BPN's interpolative abilities, which are essential to the method's success, are hindered when it attempts to simultaneously interpolate in both dimensions of the pattern space shown in Figure 1. When this single BPN is presented with an unfamiliar pattern, it has difficulty determining the contribution of each parameter, K_p or τ_p, to the degree of deviation from the desired pattern. This inhibits the network's

interpolative abilities, thereby producing inaccurate predictions of δ_κ and δ_τ.

To overcome this problem, two separate BPN's are trained such that the first network predicts the K_p adjustment multiplier, δ_κ, while the second network predicts the τ_p adjustment multiplier, δ_τ, where both predictions are based on the same error pattern. Each network contains one output neuron. Since the limits on the neuron outputs are $[-1, 1]$ as given by Eq. 2, and δ_κ and δ_τ range from 0.1 to 10, the desired outputs used for the training patterns are, for the first BPN:

$$d_\kappa = \log(\delta_\kappa) \tag{13}$$

and, for the second:

$$d_\tau = \log(\delta_\tau) \tag{14}$$

Note that this approach assigns the error pattern representing desired closed loop performance, where $\delta_\kappa = 1$ and $\delta_\tau = 1$, an output of 0.0 from each BPN.

In addition to the 50 error samples, each training pattern includes a 51st value which is the adjustment multiplier of the parameter opposite that which the given BPN is predicting. Thus, the BPN designed to predict δ_κ is trained with patterns that contain the 50 error samples along with the value of δ_τ used to create that pattern. Similarly, the BPN to predict δ_τ uses the same 50 error samples along with the value of δ_κ used to create that pattern.

To accurately match the training patterns to actual closed loop error patterns and to further aid in making the method process independent, the training patterns are normalized. This is done by assigning the initial maximum error value in the error pattern to 1.0 and adjusting the other 49 error samples accordingly. Upon implementation, incoming error patterns are similarly normalized.

Both BPN's are trained on the same set of 64 error patterns, which is a sufficient number to span the entire pattern space of Figure 1. Once the trained networks are presented with unfamiliar patterns on-line, the interpolative abilities of the BPN architecture allow the networks to translate an error pattern into appropriate values of δ_κ and δ_τ, rather than just matching the pattern with the closest pattern in the training set. Both networks required 7 middle neurons to sufficiently minimize E in Eq. 3, and each needed approximately 400 presentations of the given training set to converge.

Once the two networks are trained, they are implemented in the adaptive strategy shown in Figure 3. As previously discussed, BPN No. 1 produces an output which describes, through a simple scaling factor, a prediction of δ_κ while BPN No. 2 likewise describes δ_τ. To obtain δ_κ and δ_τ from the continuous valued BPN outputs, Eqs. 13 and 14 are rewritten in terms of the on-line BPN outputs z_κ and z_τ, rather than the training pattern outputs d_κ and d_τ, such that:

$$\delta_\kappa = 10^{z_\kappa} \tag{15}$$

and

$$\delta_\tau = 10^{z_\tau} \tag{16}$$

Upon presentation of an actual error pattern, the strategy

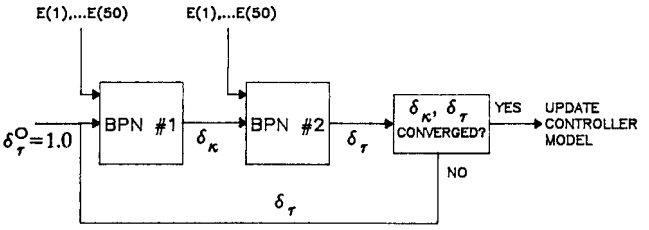

Figure 3. Adaptive BPN strategy.

is initialized by setting δ_τ, the 51st input to the first net, to 1.0. Thus the strategy initially assumes that τ_p is at its desired value. The first BPN produces an output, z_κ, which is converted to δ_κ by Eq. 15. The second net uses this value of δ_κ along with the same error pattern to produce an output, z_τ, which is converted to δ_τ by Eq. 16. With this new estimate for δ_τ, the procedure is then repeated for as many iterations as necessary until each net converges, within 10 percent, on values for both δ_κ and δ_τ. Typically, only two or three iterations are required for convergence. By fixing one parameter while searching for the other, each net is only interpolating in one row or one column of the pattern space at a time, eliminating the problems associated with two dimensional interpolation.

Once the appropriate combination of δ_κ and δ_τ is determined, model parameter adjustment is straightforward. δ_κ and δ_τ describe the multiplicative factor by which the model parameters K_p and τ_p vary from their desired values. Thus the controller model adaptations at sample number t are:

$$K_p(t) = K_p(t-1)\delta_\kappa \tag{17}$$

and

$$\tau_p(t) = \tau_p(t-1)\delta_\tau \tag{18}$$

It is important to note that this approach is developed such that it is not the actual parameter values that are important, but rather, the ratios of the current estimates of K_p and τ_p to their values which restore desired performance. As a result, the BPN's trained with the ideal process can be applied to significantly different processes without retraining.

VQN-Adaptive Strategy

Vector quantizer network

The second network employed in this work is a vector quantizer network (VQN). Classical vector quantizers (Gray, 1984) are algorithms used for data compression where high dimensional data vectors are encoded to low dimensional data vectors to reduce storage and transmittance requirements. Upon receiving or recalling the encoded data, a codebook uses it to create a reproduction vector. The goal is to minimize any distortion or error between the original data vector and its associated reproduction vector.

The pattern classification network detailed here is considered a vector quantizer in that 50 element input vectors are encoded into two array indices that indicate the input vector's corresponding pattern class. This in turn points to model parameter adjustment multipliers, δ_κ and δ_τ, effectively compressing the input data vector.

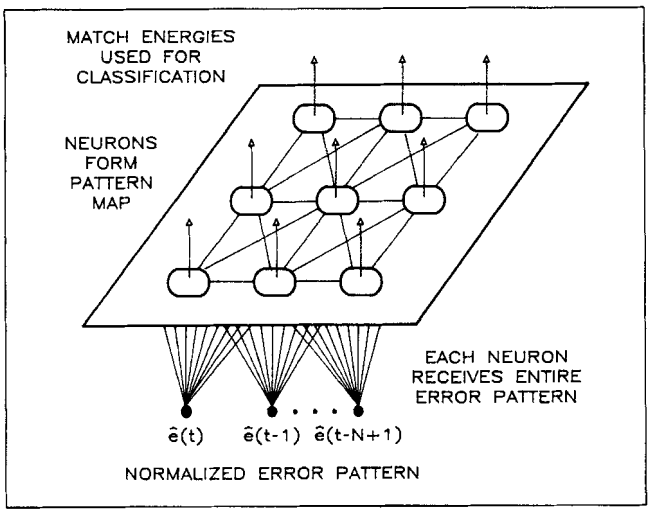

Figure 4. Vector quantizer network structure.

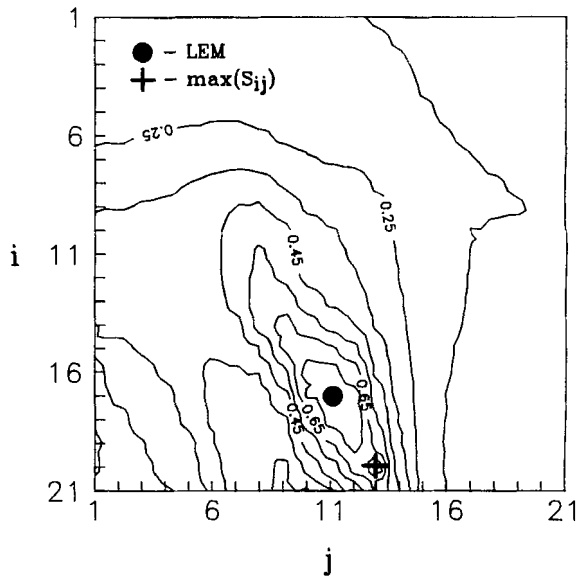

Figure 5. Local energy topology example.

The VQN architecture, shown in Figure 4, is comprised of two layers of neurons. The first layer of neurons receives and distributes the input pattern just as in the BPN. The neurons in the second layer are arranged in a two-dimensional array, where each neuron in this layer represents a discrete pattern class. The organization of the neurons forms a pattern map which is a discrete representation of the entire pattern space, a portion of which is summarized in Figure 1. These neurons use a simple threshold logic nonlinearity instead of the sigmoidal nonlinearity employed by BPN neurons. This imposes constraints of 0.0 and 1.0 on the output from each neuron.

Upon presentation of a normalized input pattern, each neuron outputs a match energy representative of how well the input pattern matches that neuron's pattern class. Just as with the BPN, whenever a set point change is made, 50 controller error samples are collected and cast to form an error pattern vector:

$$\mathbf{e}^T = [e(t)\ e(t-1). \ . \ . \ e(t-49)] \tag{19}$$

where sample number t is the most recent sample and sample number t-49 is the sample just prior to the set point change. The pattern class vector for neuron ij is also a vector of length 50 and is expressed:

$$\mathbf{p}_{ij}{}^T = [p_{ij}(1)\ p_{ij}(2). \ . \ . \ p_{ij}(50)] \tag{20}$$

These error and pattern class vectors are then normalized so that they may be compared on a common scale. This is expressed as:

$$\hat{\boldsymbol{e}} = \|\mathbf{e}\|_E \quad \hat{\mathbf{p}}_{ij} = \|\mathbf{p}_{ij}\|_E \tag{21}$$

where, if $\hat{\mathbf{x}} = \|\mathbf{x}\|_E$ denotes the Euclidean normalization of vector $\mathbf{x}$, the normalization of element n is computed:

$$\hat{x}(n) = \frac{x(n)}{\left[\sum_{n=1}^{50} x(n)^2\right]^{1/2}} \tag{22}$$

Each neuron computes a match energy between the normalized error vector and its own pattern class vector by taking their inner product. The match energy for each neuron ij is thus:

$$S_{ij} = \hat{\boldsymbol{e}}^T \hat{\mathbf{p}}_{ij} \tag{23}$$

A perfect match would have a match energy $S_{ij} = 1.0$. Any match that is less than perfect has an energy $-1.0 \leq S_{ij} < 1.0$, but the threshold logic nonlinearity constrains this value between 0.0 and 1.0.

Local energy maximum

A simple method for determining the most representation pattern class is choosing that neuron with the greatest match energy. Unfortunately, this may lead to an incorrect classification. Such a result may arise in the presence of measurement noise or process nonidealities that can skew individual match energy computations. For this reason, the local energy maximum (LEM) is used to determine the most representative pattern class.

The individual match energies of the neurons in the two-dimensional array form an energy topology, an example of which is shown in Figure 5. This shows the energies for each of the patterns in the pattern map, where the grid is set up in the same manner as the pattern map in Figure 1. Thus each row i is a set of patterns created by a particular mismatch between the K_p of the model and the K_p of the process. Similarly, each column j is a set of patterns created by a particular mismatch in τ_p. The point denoted by (11, 11) at the center of the array represents the desired pattern, where $\delta_\kappa = \delta_\tau = 1.0$. The local energy (LE) is computed at each neuron by summing the energies of that neuron and its four closest neighbors:

$$LE_{ij} = S_{ij} + S_{i+1,j} + S_{i-1,j} + S_{i,j+1} + S_{i,j-1} \tag{24}$$

The error pattern is then classified as that pattern represented by the neuron at the center of the LEM.

The benefit of such an approach is shown in Figure 5. The neighbors of the neuron with the largest match energy, denoted by a cross, have match energies that are much smaller in comparison. This suggests that the largest match energy is an aberration and that classifying the error pattern based solely on the individual match energies may lead to an inappropriate model adaptation. The LEM, however, is generally located at the apex of the major peak of the energy topology, denoted by a filled circle in Figure 5. This suggests that the neuron at the center of the LEM correctly classifies the error vector.

As a precaution, the match energy of the neuron at the center of the LEM is considered. If the error vector, $\hat{\mathbf{e}}$, is unlike any of the pattern class vectors, the $\hat{\mathbf{p}}_{ij}$ at the center of the LEM may be a poor match. To ensure the reliability of each classification, the match energy of this neuron is compared to a vigilance factor, V_f, similar to that of Carpenter and Grossberg (1987). If $S_{ij} \geq V_f$, the pattern classification is considered an appropriate match and the network output is used for process model adaptation. If the match energy does not meet this vigilance test, however, the network has failed to appropriately classify the error pattern and no adaptation is made. In this work, the vigilance factor is chosen to be 0.90.

VQN organization and implementation

The output of the VQN is not a continuous value. Rather, it is a discrete classification that points to a pair of model parameter adjustment multipliers, δ_κ and δ_τ. For the network to precisely classify error patterns and thus improve the accuracy of the model parameter adjustments the entire pattern space must be much more finely discretized than the 64 patterns required for training the BPN.

To balance the desire for more pattern classifications with the need for a moderate computation load, the error transient pattern space summarized in Figure 1 is represented as 441 distinct pattern classes. Each pattern consists of 50 error samples and is cast to form a vector. The vectors are then normalized as in Eq. 22 and these 441 vectors become the pattern class vectors, $\hat{\mathbf{p}}_{ij}$, for the 441 neurons. They are organized in order of parameter adjustment in a 21 by 21 array with the desired pattern in the center.

Upon implementation, each transient results in a vector of 50 controller error samples which is normalized to produce the error pattern $\hat{\mathbf{e}}$. This pattern is multiplied by all 441 pattern class vectors $\hat{\mathbf{p}}_{ij}$ as in Eq. 23 to produce 441 match energies, S_{ij}.The LEM then points to appropriate model parameter adjustment estimates. As long as the vigilance test has been passed, the model parameter updates are performed using Eqs. 17 and 18.

Control Algorithms

Internal controller model

This work employs a first-order (FO) model as the internal controller model. This model is written in difference form as:

$$y'(t+1|t) = y(t) + a_1 \Delta y(t-1) + b_1 \Delta u(t-1) \tag{25}$$

where

$$a_1 = e^{-\Delta t/\tau_p};\ b_1 = K_p\,(1-a_1) \tag{26}$$

and $y'(t+1|t)$ is the predicted value of $y(t+1)$, the true output at sample number $t+1$, based on information available at sample number t. The adaptation strategy is to adjust both K_p and τ_p in the internal model as shown in Eqs. 17 and 18 based on the network output. As previously discussed, sample time is defined such that 50 samples contains sufficient dynamic information about the character of the process.

Adaptive GPC algorithm

The GPC algorithm, a long range predictive control algorithm, has a performance objective based on a series of future predictions and control actions:

$$J(t) = \sum_{j=N_1}^{N_2} (y'(t+j|t) - y_{sp}(t+j))^2 + \sum_{j=1}^{N_u} Q(t)\,(\Delta u(t+j-1))^2 \tag{27}$$

where $y'(t+j|t)$ is the predicted value of the output $t+j$ steps into the future based on information at time t, N_1 and N_2 are the near and far horizons, respectively, for costing the difference between the predicted set point and response trajectory and N_u is a horizon for computing future control actions. All control actions beyond this horizon are assumed to be zero. $Q(t)$, the penalty on the activity of the controller, is the single adjustable parameter. The predicted output is computed from multiple applications of Eq. 25.

Equation 27 is minimized via recursion on the Diophantine identity (Lalonde and Cooper, 1989; Clarke et al., 1987) to produce $\mathbf{u}$, a vector of unconstrained future control increments:

$$\mathbf{u} = (\mathbf{G}_f^T \mathbf{G}_f + \mathbf{K})^{-1}\, \mathbf{G}_f^T\, (\mathbf{y}_{sp} - \mathbf{y}_0) \tag{28}$$

where $\mathbf{G}_f$ is a model parameter matrix associated with predicting the future output based only on future control actions, $\mathbf{y}_0$ is that component of the output prediction based only on control actions already implemented, and $\mathbf{K}$ is a matrix of future penalties on controller activity. For every controller action, the vector $\mathbf{u}$ is computed but only the first of these future control actions is implemented. At the next sampling instant, a new future control vector is again computed.

To implement this algorithm, the minimum costing horizon, N_1, is set to the time elapsed in samples before the system begins to make a positive response to a change in input. Since dead time is not the focus of this work, N_1 is set equal to 1, the one sample dead time inherent in discrete systems. To cost an entire trajectory, the maximum horizon, N_2, should encompass the completion of a response. Using a FO model, then $N_2 = 5\tau_p/\Delta t$.

The control horizon for this implementation is specified as $N_u = 1$. Hence, with only a single future control action being computed, $\mathbf{K} = Q(t)$. This penalty on controller activity is specified in this work as:

$$Q(t) = \beta K_p(t)^2 \tag{29}$$

This casts the two terms of Eq. 27 into consistent units and

works toward keeping them proportionate in size. The pre-multiplier β in Eq. 29 is a positive integer value that becomes the single tuning knob available to the operator and is chosen in the demonstrations such that a set point change with the correct model parameters approximates the desired response shown in Figure 1.

Recursive least squares based GPC

For comparison, this work also demonstrates a traditional recursive least squares (RLS) procedure to update the linear difference controller model (for example, Åström and Wittenmark, 1989; Ljung, 1987; Seborg et al., 1986). The popular RLS modifications, which are necessary in nonlinear and non-stationary applications where the model must be continually linearized to the changing local character, include variable forgetting factors (for example, Fortescue et al., 1981) and covariance resetting (for example, Goodwin et al., 1983). These and related modifications increase the sensitivity of RLS model updating to new data. To ensure the model remains descriptive of the current character of the application, the modifications are to be activated during periods of sufficient input-output excitation.

The derivation and implementation of RLS based model updating for adaptive control is well documented (for example, Åström and Wittenmark, 1989; Seborg et al., 1986). Briefly, if a sampled data vector is defined:

$$\Psi^T(t-1) = [\Delta y(t-1) \Delta u(t-1)] \tag{30}$$

and a parameter vector is defined:

$$\Theta^T(t-1) = [a_1 b_1] \tag{31}$$

then, $\Theta'(t)$, an estimate of the parameter vector at the current sample number, can be computed:

$$\Theta'(t) = \Theta'(t-1) + \mathbf{P}(t)\Psi(t-1) \\ (y(t) - \Psi^T(t-1)\Theta'(t-1)) \tag{32}$$

where the covariance matrix of the estimation error is computed:

$$\mathbf{P}(t) = \mathbf{P}(t-1) - \mathbf{P}(t-1)\Psi(t-1) \\ (\Psi^T(t-1)\mathbf{P}(t-1)\Psi(t-1) + 1)^{-1}\Psi^T(t-1)\mathbf{P}(t-1) \tag{33}$$

RLS based model updating is implemented in this work with the intention of giving the method a reasonable opportunity to succeed. Hence, all parameters are properly converged at start-up. The RLS modification employed is covariance resetting, which is implemented with the benefit of complete knowledge of when the local character of the process will change. Specifically, just prior to every set point change, the RLS algorithm is resensitized by resetting the diagonal elements of the covariance matrix, $\mathbf{P}$, as:

$$\mathbf{P}_{i,i} = 1.0 \tag{34}$$

Since the expected controller response after every set point change is a series of manipulations of the input, this approach permits the algorithm to immediately begin exploiting the dynamic input-output information to update the controller model.

As with the pattern based GPC algorithm, the first-order difference model given by Eq. 25 is employed for predicting the future system output for the RLS implementation. Since there is no ANN to directly estimate a current value for K_p and τ_p, the parameters required for the computations of a_1 and Eq. 29 are obtained from model parameters as:

$$K_p = \frac{b_1}{(1-a_1)} \qquad \tau_p = -\frac{\Delta t}{\ln(a_1)} \tag{35}$$

Controller start-up

Some process specific characteristics must be determind at the start-up operating regime for controller implementation namely the model parameters K_p and τ_p. Methods for determining these parameters need not be sophisticated, and to emphasize this point, the controller designs in all demonstrations are based on the regression of a FO model to data collected from a single open loop step test made at the strategy operating regime. As stated previously, sample time is determined separately for each application in the demonstrations.

First-Order Process Demonstration

Set point tracking via the GPC algorithm is demonstrated on a simulated first-order process that experiences significant changes in process gain. The use of a first-order process ensure that there is no plant-model mismatch. However, recall that the pattern space for the adaptive strategy is developed with a second-order process chosen as the ideal process. Thus this demonstration illustrates how the idealized pattern space is applicable to processes of different model order.

The simulated first-order process, as illustrated in Figure is a pH control process. The process is comprised of an acidic stream and a basic stream which are combined to feed a stirred tank reactor. The control objective is to maintain the pH out of the reactor at set point by manipulating $F_b(t)$, the flow rate of the basic stream to the reactor. As $F_b(t)$ changes, total food rate to the reactor is held constant by a flow controller which adjusts $F_a(t)$, the flow rate of the acidic stream. This flow

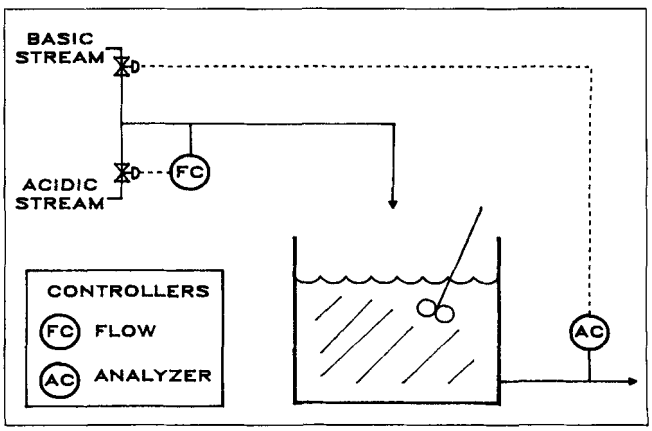

Figure 6. First-order process diagram.

controller is assumed to react instantaneously, resulting in a constant reactor volume and constant total flow rates in and out of the reactor.

The model for the reactor is similar to the model described in Li et al. (1988) and Orava and Niemi (1974). Assuming that H^+ and OH^- exist in equilibrium at all times, let C denote the concentration difference:

$$C = H^+ - OH^- = H^+ - K_w/H^+ \tag{36}$$

$K_w = 10^{-14}$ is the equilibrium constant. A mass balance on the reactor then results in the dynamic equation:

$$\frac{dC(t)}{dt} = \frac{F_a(t)C_a(t) + F_b(t)C_b(t) - (F_a(t) + F_b(t))C(t)}{V} \tag{37}$$

Solving the quadratic equation for H^+ expressed in Eq. 36 and is using the common log of the positive root results in the expression for $pH(t)$ out of the tank:

$$pH(t) = -\log\{0.5[C(t) + (C(t)^2 + 4.0*10^{-14})^{0.5}]\} \tag{38}$$

Each ANN based adaptive strategy is combined with the ***GPC*** algorithm in tracking a set point trajectory that moves the process output across the entire range of operation. The set point trajectory begins in the low pH region where K_p is small, steps up through the region where K_p increases by approximately three orders of magnitude and, as it continues up, moves operation into a region where K_p decreases to its original value. Due to the constant reactor volume and flow rates, τ_p remains essentially constant in this demonstration. A small amount of random error, $\sigma = 0.01$, is added to simulate measurement noise.

Fixed model GPC

An appreciation for the adaptive challenge presented by this process is shown in Figure 7. This shows the performance of a fixed model GPC algorithm applied to the pH process. The of point is increased by a pH of 0.5 at equal intervals of 75 samples. A value of $\beta = 20$ produces desired performance in the start-up regime where the model parameters are accurate. For this trial and subsequent adaptive GPC trials sample time is set at $\Delta t = 0.3\tau_p$, based on the true τ_p at start-up, and held constant thereafter. This allows sufficient time for a pattern to evolve that can be represented by the idealized pattern space.

As shown in Figure 7, highly aggressive control actions and input saturation appear in the middle region of the trajectory as the process gain increases dramatically while the model parameters remain fixed. As the pH increases to where the gain approaches its original value, the controller regains performance as the model again becomes descriptive of the process character.

BPN adaptation

Figure 8 shows the performance of the BPN-adaptive GPC algorithm in tracking the set point trajectory of the first-order process. The first set point steps move the process toward a dramatically higher gain. Although the error responses are

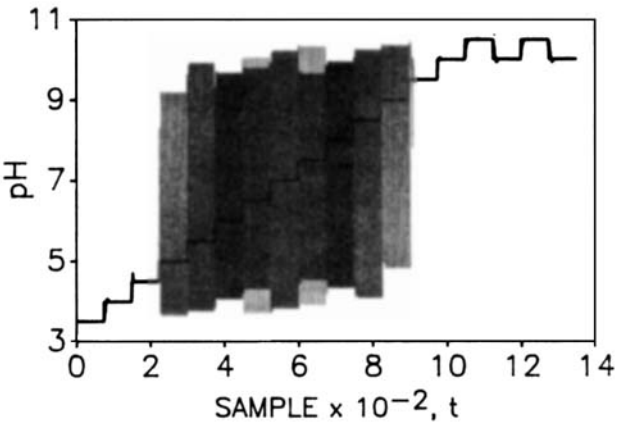

Figure 7. Fixed model GPC control of first-order process.

characterized by a large overshoot, the pattern based adaptation after each step succeeds in providing stable and fairly robust performance. After passing a pH of 7, the process gain decreases equally dramatically. Here, the error response tends to be somewhat overdamped, but the adaptation strategy again succeeds in maintaining stable performance.

The final three set point steps in the trajectory illustrate controller performance when operation remains within a region where the local process character is reasonably constant. Although each of the final three set point steps result in a process gain change which subsequently requires a small adaptation, performance approaches that of the desired error response.

Figure 9 illustrates the pattern recognition capabilities of the BPN. The solid line in Figure 9 is a normalized error pattern that results from the second set point step, at sample 150. The output produced by the BPN for model parameter adaptation after this set point step is $z_\kappa = 0.388$ and $z_\tau = 0.054$. The dashed line is the training pattern whose output, $d_\kappa = 0.300$ and $d_\tau = 0.041$, is closest to that of the actual error pattern. This figure demonstrates the interpolative abilities of the BPN as the on-line output from BPN No. 1, z_κ, is greater than that of the training pattern, d_κ, corresponding to its more aggressive behavior. Note that the value of z_τ close to 0.0 corresponds to little adjustment in τ_p.

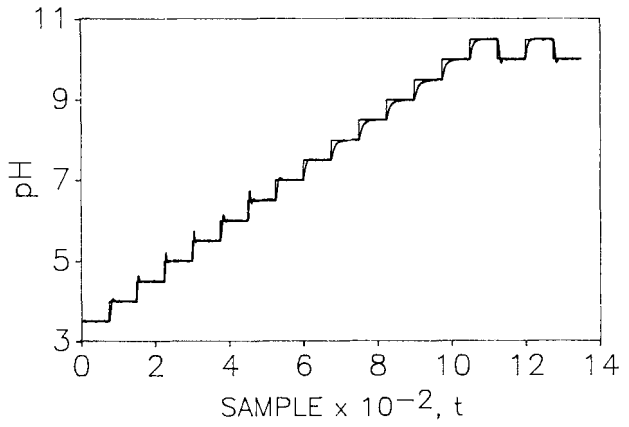

Figure 8. Adaptive BPN-based GPC control of first-order process.

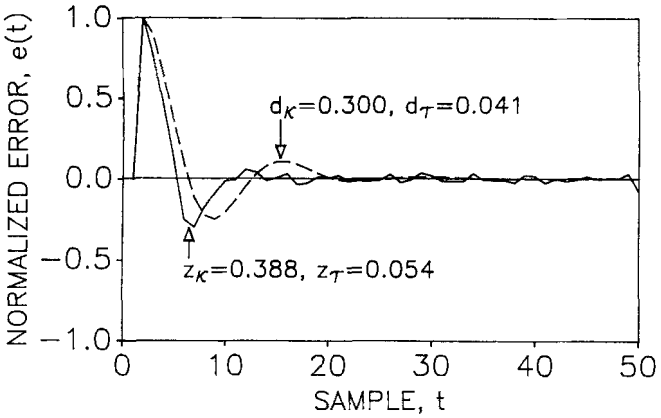

Figure 9. Pattern recognition of normalized error pattern of adaptive BPN-based GPC control for step 2 of first-order process.

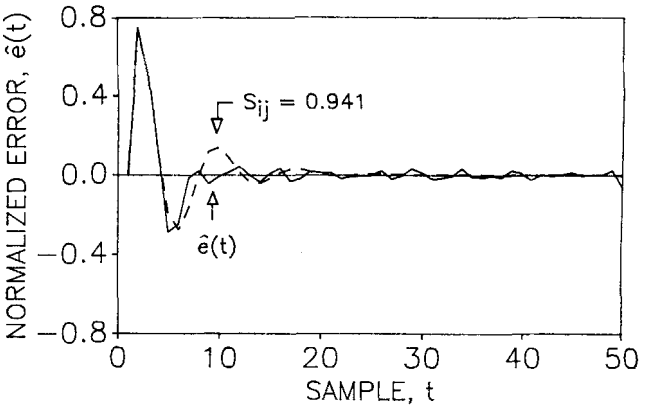

Figure 11. Pattern recognition of normalized error pattern of adaptive VQN-based GPC control for step 2 of first-order process.

VQN adaptation

Next, the performance of the VQN-adaptive GPC strategy is presented. As with the BPN adaptive GPC strategy, $\beta = 20$ provides a desired response in the start-up regime, and sample time is set to $\Delta t = 0.3\tau_p$ to collect sufficient data in the 50 error samples after each set point step.

The performance of the VQN-adaptive GPC strategy in tracking the first-order set point trajectory, shown in Figure 10, is similar to the performance of the BPN adaptive GPC strategy. Again, underdamped performance results in the first set point steps as the process gain increases, but continued adaptation maintains stable and reasonably robust performance. Once the *pH* increases above 7, performance tends toward a sluggish response due to the decreasing process gain. The final three set point steps in the trajectory again show that the controller approaches desired performance when the process character is reasonably constant.

Figure 11 illustrates the pattern recognition capabilities of the VQN. The solid line in Figure 11 is the Euclidean normalized error pattern that is sent to the VQN after the second set point step. The dashed line in the figure is the pattern class which is at the center of the LEM. Figure 11 shows the winning pattern class has a match energy $S_{ij} = 0.941$, and the classification for this oscillating pattern is accurate. As can be seen in the ensuing set point change, the model parameter adaptation resulting from this classification maintains good control performance.

RLS adaptation

As a comparison to the ANN-adaptive GPC algorithms for this nonlinear set point tracking challenge, Figure 12 shows the performance of the GPC algorithm which employs the RLS model updating scheme described earlier. Figure 12 shows the RLS-based GPC where the sensitivity of the parameter updating equations are increased through covariance resetting right before each set point step. This resets the algorithm just prior to what should be significant excitation, providing opportunity for convergence to the local character of the first-order process.

In spite of these considerations, the RLS-based GPC algorithm does not perform as well as the ANN-based controllers in the first set point steps. The performance is plagued by significant overshoot and poor damping. Further, the controller fails altogether when the process moves into the region

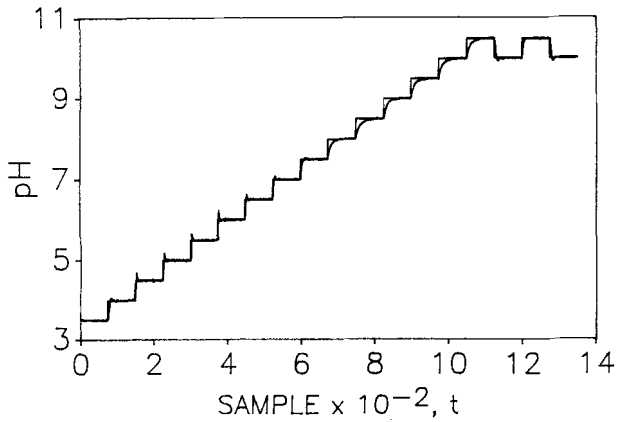

Figure 10. Adaptive VQN-based GPC control of first-order process.

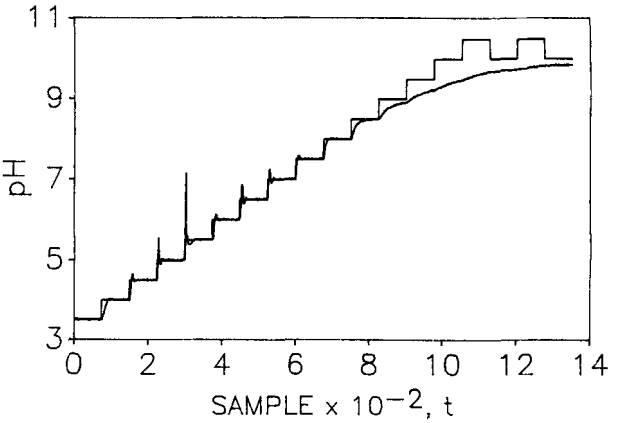

Figure 12. Adaptive RLS-based GPC control of first-order process.

Table 1. Systems Used in Simulated Third-Order Process Demonstration

Systems	Equations
1	$Y(s)=\frac{1}{(s+1)^3}U(s)$
2	$Y(s)=\frac{2}{(s+1)^3}U(s)$
3	$Y(s)=\frac{2}{(1.5s+1)^3}U(s)$
4	$Y(s)=\frac{1}{(1.5s+1)^3}U(s)$
5	$Y(s)=\frac{1}{(s+1)^3}U(s)$

where process gain is rapidly decreasing. This results from the inability of RLS to recognize that the incoming data lacks sufficient excitation for it to perform proper parameter updating.

Third-Order Process Demonstration

Set point tracking is next demonstrated on a simulated third-order process that experiences significant and abrupt changes in process character. The adaptive strategy is now challenged to maintain good performance in spite of both plant-model mismatch and the fact that once again the process is of different order from that which the ideal error patterns were developed.

The simulated third-order process consists of a series of five different systems, expressed in the Laplace domain in Table 1. At regular intervals, each new system is implemented and a series of four set point changes is made to demonstrate the adaptive strategy. The GPC algorithm for these demonstrations is preconverged for system 1. This is done using accurate estimates of the process gain and time constant and by specifying $\beta=20$. Sample time for this process is defined as $\Delta t=0.5\tau_p$, based on the true τ_p, and held constant thereafter. Again this is to insure the time scale of the error patterns will be similar to those of the patterns in the ideal pattern space.

At sample 1, system 2 is implemented. This significantly increases the process gain. System 3 is then implemented at sample 375, which maintains the increased gain and increases the overall time constant of the process. Implementation of system 4 at sample 675 maintains the increased time constant while decreasing the process gain back to its original value. Finally, at sample 975, the implementation of system 5 reduces the overall time constant of the process thereby returning the process to its original character.

Four set point step changes of a magnitude of 10.0 are made at 75 sample intervals in the form of a pyramid following the implementation of each system. These are designed to illustrate the ability of the adaptive strategies to converge on the current process character. The pyramid trajectories are provided to easily discern one system from another.

Fixed model GPC

Application of a fixed model GPC algorithm to the third-order process provides an appreciation for the adaptive challenge, as shown in Figure 13. The process output and the set point are shown plotted versus time expressed in samples. The

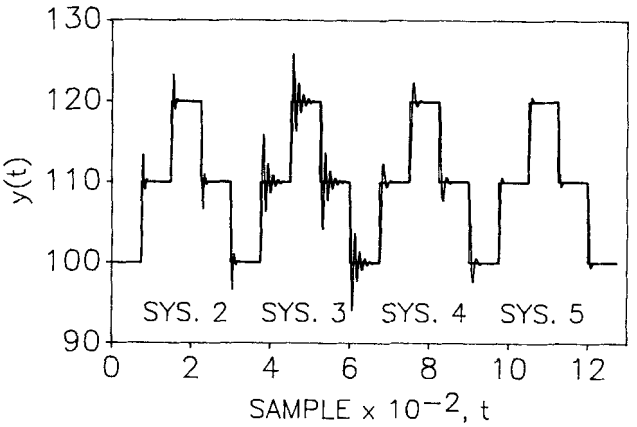

Figure 13. Fixed model GPC control of third-order process.

responses to the set point changes made during the implementation of systems 2, 3 and 4 are marked by excessive overshoot and slowly damping oscillations. The responses that result during the implementation of system 5 display desired controller performance as the process is returned to the state where it is accurately described by the model. It is acknowledged that plant-model mismatch contributes to the poor controller performance, but the purpose is to demonstrate that such deficiencies can be overcome by the adaptive strategies, as is demonstrated in the following section.

ANN adaptation

Figure 14a shows the performance of the BPN-adaptive GPC strategy for the third-order demonstration while Figure 14b shows performance of the VQN adaptive GPC strategy. Just as in the first-order demonstration, the pattern recognition capability of the two ANN's is comparable throughout. Typically, either adaptive ANN strategy is able to recover close to desired performance following two adaptations on each new system, as evidenced by the response to the third set point change for a given system.

For the implementation of systems 3 and 4, the BPN adaptive strategy has more difficulty converging on a desired response than does the VQN as it is not able to make as precise classifications as the VQN. However, it is able to maintain stable and reasonably robust performance throughout the trajectory. The VQN adapts to and maintains a desired response for all four systems. Due to the slower dynamics caused by the increased time constant following the implementation of system 3, the process does not respond as quickly in systems 3 and 4 and the resulting desired pattern displays more overshoot than in the other two systems. Nevertheless, the VQN is able to maintain robust performance throughout the trajectory.

Second-Order Process Demonstration

A pilot-scale laboratory experiment is used as the final demonstration process. This process, as shown in Figure 15, consists of two noninteracting tanks in series. The control objective is to maintain the liquid level in the second tank by manipu-

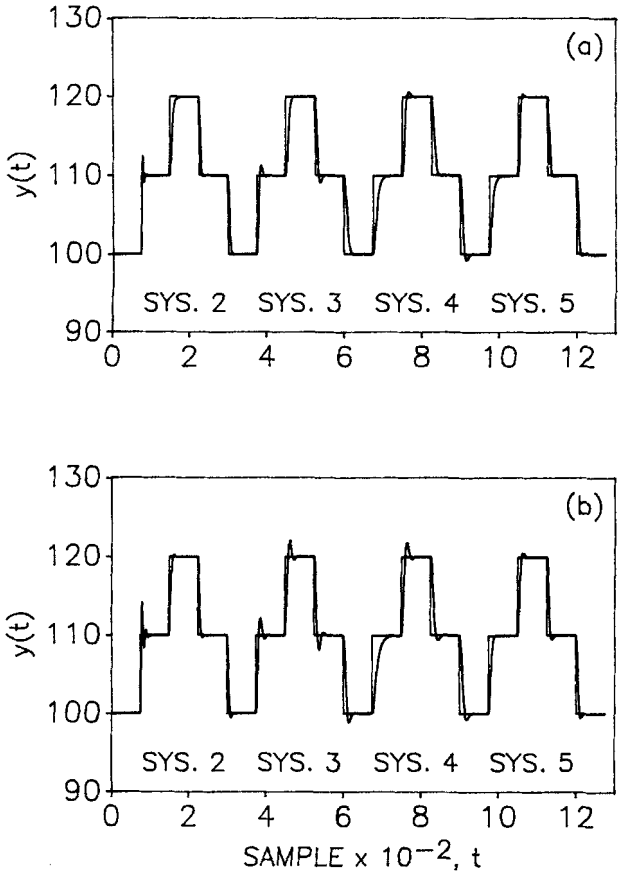

Figure 14. Adaptive (a) BPN- and (b) VQN-based GPC control of third-order process.

lating the flow rate into the first tank by means of a variable speed pump. The liquid level in the second tank is measured by using a pressure transducer and converting head to height using the liquid density, where the liquid in this experiment is water. Common to real applications, there is a significant amount of measurement noise present where $\sigma \simeq 0.02$.

Whereas there is no model order mismatch between the ideal pattern space and the process, this demonstration does consider the nonidealities of significant measurement noise and plant-model mismatch. Measurement noise will make pattern classification more difficult as it distorts the pattern, thus diminishing the distinctive features of a particular pattern.

The initial adaptive challenge is achieved by underestimating the process gain at the start of the set point trajectory. An estimate of one half the true value of K_p is used with the correct estimate of τ_p. The controller is the adaptive GPC using a $\beta = 25$, where sample time is set to $\Delta t = 0.3\tau_p$, based on the true τ_p at start-up, and held constant thereafter. These values produce the desired target pattern when the model parameter estimates are correct. Three set point steps are then made at 75 sample intervals between the initial height of 0.15 m and 0.20 m. These form a square wave and keep the process in a region of locally linear behavior, demonstrating the ability of the adaptive strategy to restore desired performance.

The fourth set point step, at sample 300, then increases the height to 0.45 m, moving operation into a region of higher process gain and time constant. Again, three additional set point steps are included between 0.45 m and 0.50 m to provide

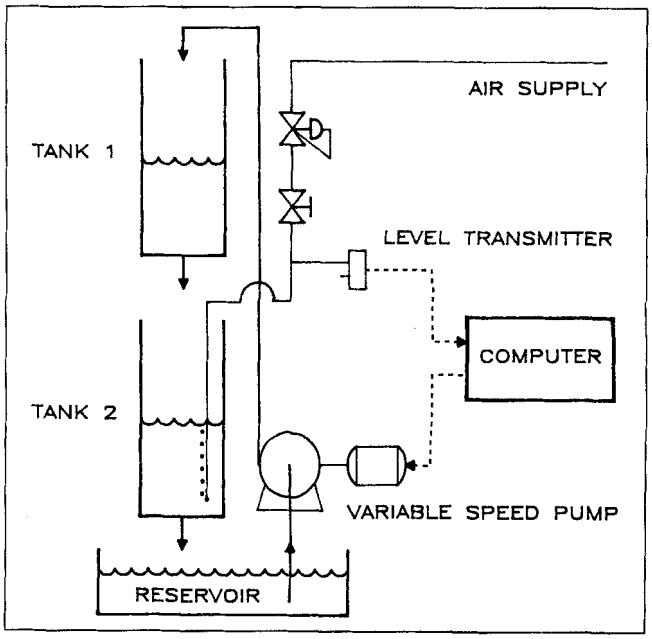

Figure 15. Second-order process diagram.

the adaptive strategy with the opportunity to converge in a region of locally linear behavior.

Fixed model GPC

The fixed model GPC algorithm is once again used to illustrate the adaptive challenge that is presented to the ANN-adaptive GPC strategies. Figure 16 shows the incorrect initial estimate of K_p results in underdamped responses for the first square wave, marked by excessive overshoot and slow damping. These conditions worsen as the process moves to the region of higher gain and time constant at 0.45 m. The result is a total loss of control and saturation of the process input for the remainder of the trajectory.

VQN adaptation

As previous demonstrations illustrate comparable performance between the two ANN strategies, this demonstration will

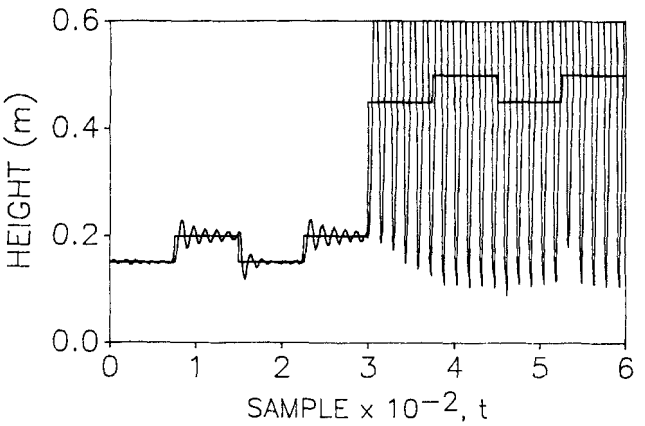

Figure 16. Fixed model GPC control of second-order process.

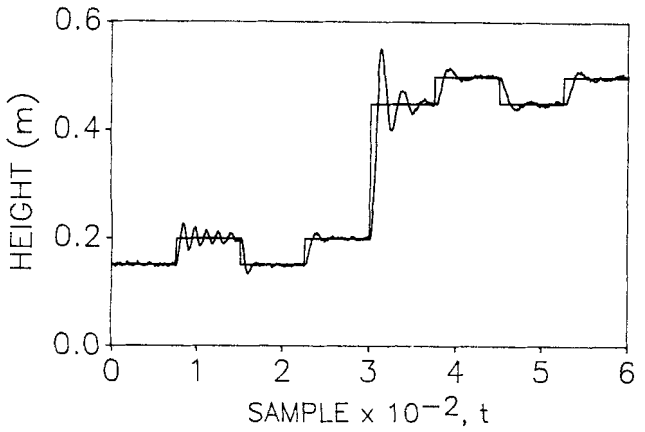

Figure 17. Adaptive VQN-based GPC control of second-order process.

focus solely on the VQN strategy. Figure 17 shows that the VQN adaptive GPC strategy is able to correct for the initial underestimation of the model gain and restore desired controller performance following two adaptations, as witnessed by the response to the third set point step at sample 225.

The fourth set point step, at sample 300, moves the process into a region of higher process gain and time constant, characterized by excessive overshoot and a slower response. The VQN adaptive strategy is able to use this information to restore nearly desired performance within two adaptations, as shown by the response to the sixth set point step at sample 450. While the presence of noise makes pattern classification more difficult, use of the local energy maximum and vigilance factor insures that the chosen pattern match will be a reasonable classification. Without them, it is possible that the noise could cause a single erroneous classification to be chosen as the winning exemplar pattern, as shown in Figure 5.

BPN vs. VQN

For the demonstrations presented, both ANN's proved equally capable in performing the pattern analysis required for the pattern based adaptive strategy. The issues that really separate the two ANN's for this strategy lie in the design and flexibility of the network architectures.

The BPN is employed here as a pattern translator. After converging the connection weights based on the training set, new patterns fed to the BPN are translated directly into two continuous valued outputs that are rescaled as estimates of the model parameter adjustment multipliers.

While the training of the BPN is faster using the CG method as opposed to more traditional methods, it is still computationally intensive. Not only does it take hundreds of presentations of the training set for convergence, but there is still a trial and error component to network design involving determination of the proper number of middle neurons and proper selection of initial weights that provide a good initial starting point. However, once converged and implemented on-line, little computation is required to translate each new input pattern to an associated output.

In some situations, a designer may have only a limited number of training patterns available or may be uncertain of the complete relationship between the input patterns and the network outputs. Hence, the ability of a BPN to develop such a relationship from a small training set can make it an attractive architecture.

For the adaptive strategy detailed in this work, the complete pattern space is known and a means is detailed for generating any number of patterns within the pattern space. Hence, the advantages offered by a network which can develop a complete input-output relationship from a limited training set are not significant in this situation.

The VQN employed here is a pattern classifier. The network possesses a number of discrete classes that span the entire pattern space. New patterns fed to the network are assigned to the most closely matching pattern class. To permit the small adaptive adjustments required of a robust strategy, many such pattern classifications are established.

With complete knowledge of the pattern space, as is the case for this adaptive strategy, the VQN pattern map is straightforward to establish. The computational disadvantage with the VQN occurs once on-line because every new pattern fed to the network must be compared against each pattern in the pattern map. The computation load is not excessive, however. For example, as detailed in this work, a match energy must be computed for 441 patterns, each represented by a vector of length 50, before the best match can be determined. This can be achieved in 22,050 multiplication plus 22,050 summation operations. While this is a significant number of operations, it is almost an instantaneous procedure when implemented on a contemporary computer system. For comparison, the two BPN's implemented in the adaptive strategy shown in Figure 3 require a combined 744 multiplication plus 744 summation operations for each iteration through the two networks. The strategy shown in Figure 3 typically converges on values of δ_κ and δ_τ within three iterations.

One characteristic of the BPN is that it always translates a new pattern into an output. However, because the sigmoidal transforming function used in each neuron of the BPN is asymptotic at its limits, pattern recognition near these limits, especially with highly oscillatory patterns, suffers in accuracy. Thus, for this adaptive strategy, poor model parameter adjustment estimates can result at these limits. Also, the BPN provides no mechanism for evaluating the accuracy of its translation, which can be disastrous in real process applications.

Classification at the limits of the pattern space is not possible for the VQN due to the nature of the local energy maximum method. This, however, is only true for the outermost neurons. Unfortunately, this ANN can have some difficulty in recognizing patterns with variations in frequency different from those in the pattern map. The VQN, however, provides a match energy that can be compared to a vigilance factor. Hence, the accuracy of each pattern classification can be immediately evaluated. For critical control operations, the importance of this capability cannot be underestimated.

Although on-line updating of the pattern space was not addressed within the scope of this study, such methods may prove necessary to make the adaptive strategy more robust in some applications. Because retraining a BPN requires all of the iterative computation and many of the same trial and error decisions as in the original design, the BPN architecture is not appropriate for such on-line updating. Updating the VQN pattern map, however, is just as straightforward as it is to initially establish the pattern map.

As this work has shown, the two ANN architectures prove equally capable as pattern analysis tools. However, considering the above comparisons, the VQN architecture is better suited for the pattern analysis required of the adaptive strategy detailed in this work.

Acknowledgment

Acknowledgment is gratefully made to the National Science Foundation through Grant CTS-9008596, and to the Foxboro company, for their support through an unrestricted donation.

Notation

a_1 = model parameter associated with system output
b_1 = model parameter associated with system input
C_a = concentration of acidic stream in first-order process
C_b = concentration of basic stream in first-order process
C = concentration difference of reactor in first-order process
$\mathbf{d}$ = desired BPN output pattern
d_j = desired BPN output of neuron j
d_κ = desired output for BPN No. 1
d_τ = desired output for BPN No. 2
$\boldsymbol{e}$ = recent controller error history vector for VQN
$\hat{\boldsymbol{e}}$ = normalized controller error history vector for VQN
$e(t)$ = system error at sample number t
E = error over entire training set in BPN training
E^p = error for a single pattern presentation in BPN training
F_a = feed rate of acidic stream in first-order process
F_b = feed rate of basic stream in first-order process
$\mathbf{g}$ = gradient vector for presentation of entire training set
$\boldsymbol{g}^p$ = gradient vector for presentation of one pattern
g_{ij} = gradient at a single neuron in BPN
$\boldsymbol{G}_f$ = parameter matrix used for predicting future output based on future control actions in GPC algorithm
H^+ = concentration of hydrogen ions in first-order process
k = iteration counter for each pattern presentation in BPN training
$\mathbf{K}$ = matrix of future penalties on controller activity in GPC algorithm
K_p = steady-state process gain
K_p^* = multiple of true gain used to develop training pattern
K_w = equilibrium constant in first-order process
LE_{ij} = local energy for neuron ij in VQN
n = iteration counter for complete presentation of BPN training set
N_1 = minimum costing horizon in GPC algorithm
N_2 = maximum costing horizon in GPC algorithm
N_u = control horizon in GPC algorithm
OH^- = concentration of hydroxide ions in first-order process
$\mathbf{p}_{ij}$ = vector of connection weights to neuron ij in VQN
$\hat{\mathbf{p}}_{ij}$ = normalized vector of connection weights to neuron ij in VQN
$\mathbf{P}$ = covariance matrix of estimation error in RLS algorithm
P = number of patterns in BPN training set
$Q(t)$ = penalty on controller activity in GPC algorithm
$\mathbf{s}$ = search direction in CG method
S_{ij} = matching score for neuron ij in pattern map in VQN
Δt = sample interval
t = time expressed as integer number of samples
$\boldsymbol{u}$ = vector of future control actions computed by GPC algorithm
$u(t)$ = manipulated system input at sample number t
V_f = vigilance factor in VQN
V = volume of reactor in first-order process
$\boldsymbol{w}$ = weight vector for BPN
w_{ij} = connection weight between neuron i and neuron j in BPN
$\mathbf{y}_o$ = component of output prediction based only on control actions already implemented in GPC algorithm
$\mathbf{y}_{sp}$ = vector of set points in GPC algorithm
$y(t)$ = system output at sample number t
$y'(t)$ = model estimate of true system output $y(t)$ at sample number t
$y_{sp}(t)$ = set point at sample number t
$\mathbf{z}$ = actual BPN output pattern
z_j = actual BPN output of neuron j
z_κ = actual output for BPN No. 1
z_τ = actual output for BPN No. 2

Greek letters

β = premultiplier used to adjust controller activity in GPC algorithm
γ = scalar used in calculating CG search direction
δ_κ = adjustment multiplier based on K_p mismatch
δ_τ = adjustment multiplier based on τ_p mismatch
ϵ_j = error term for neuron j in BPN
η = gain factor in GDR
$\boldsymbol{\theta}$ = parameter vector in RLS algorithm
$\boldsymbol{\theta}'$ = estimate of parameter vector in RLS algorithm
λ = step size in CG method
σ = standard deviation of measurement noise
τ_p^* = multiple of time constant used to develop training pattern
τ_p = process time constant
$\boldsymbol{\Psi}$ = sampled data vector in RLS algorithm

Literature Cited

Åström, K. J., "Auto-Tuning, Adaptation and Expert Control," *Proc. Amer. Control Conf.*, Boston, 1514 (1985).
Åström, K. J., and B. Wittenmark, *Adaptive Control*, Addison-Wesley, New York (1989).
Bristol, E. H., "Pattern Recognition: An Alternative to Parameter Identification in Adaptive Control," *Automatica*, **13**, 197 (1977).
Carpenter, G. A., and S. Grossberg, "Art 2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns," *Appl. Optics*, **26**, 4919 (1987).
Caudill, M., "Neural Networks Primer," *AI Expert*, **3**(6), 53 (1988).
Clarke, D. W., C. Mohtadi, and P. S. Tuffs, "Generalized Predictive Control: 1 and 2," *Automatica*, **23**, 137 (1987).
Cooper, D. J., and A. M. Lalonde, "Process Behavior Diagnostics and Adaptive Process Control," *Comp. and Chem. Eng.*, **14**, 541 (1990).
Edgar, T. F., and D. M. Himmelblau, *Optimization of Chemical Processes*, McGraw-Hill, New York (1988).
Fletcher, R., and C. M. Reeves, "Function Minimization by Conjugate Gradients," *Comp. J.*, **7**, 149 (1964).
Fortescue, T. R., L. S. Kershenbaum, and B. E. Ydstie, "Implementation of Self-Tuning Regulators with Variable Forgetting Factors," *Automatica*, **17**, 831 (1981).
Goodwin, G. C., H. Elliott, and E. K. Teoh, "Deterministic Convergence of a Self-Tuning Regulator with Covariance Resetting," *Proc. IEE-D*, **130**, 6 (1983).
Goryn, D., and M. Kaveh, "Conjugate Gradient Learning Algorithms for Multilayer Perceptions," *Proc. Midwest Symp. Circuits and Systems*, Part 2, 736 (1989).
Gray, R. M., "Vector Quantization," *IEEE ASSP Magazine*, **1**(4), 4 (1984).
Jau, J. Y., Y. Fainman, and S. H. Lee, "Comparison of Artificial Neural Networks with Pattern Recognition and Image Processing," *Appl. Optics*, **28**, 302 (1989).
Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin (1988).
Kohonen, T., "Self-Organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, **43**, 59 (1982).
Lalonde, A. M., and D. J. Cooper, "Automated Design, Implementation, and Adaptation of the Generalized Predictive Controller," *Proc. Amer. Control Conf.*, Pittsburgh, 1840 (1989).
Leonard, J., and M. A. Kramer, "Improvement of the Back-propagation Algorithm for Training Neural Networks," *Comp. and Chem. Eng.*, **14**, 337 (1990).
Li, W. C., L. T. Biegler, C. G. Economou, and M. Morari, "A Constrained Pseudo-Newton Control Strategy for Nonlinear Systems," *Proc. AIChE Meeting*, Washington, DC (1988).
Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Englewood Cliffs, NJ (1987).

Minai, A. A., and R. D. Williams, "Acceleration of Back-Propagation through Learning Rate and Momentum Adaptation," *Int. Joint Conf. Neural Networks*, **1**, 676 (1990).
Orava, P. J., and A. J. Niemi, "State Model and Stability Analysis of a pH Control Process," *Int. J. of Control*, **20**, 557 (1974).
Pao, Y. H., *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, New York (1989).
Polak, E., *Computational Methods in Optimization*, Academic Press, New York (1971).
Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, Cambridge (1986).
Rumelhart, D. E., and J. L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: 1. Foundations*, MIT Press, Cambridge (1986).
Seborg, D. E., T. F. Edgar, and S. L. Shah, "Adaptive Control Strategies for Process Control: a Survey," *AIChE J.*, **32**, 881 (1986).

Manuscript received Sept. 27, 1990, and revision received Aug. 3, 1991.