Adaptive, Predictive Controller for Optimal Process Control*

S.K. Brown,[a] C.C. Baum,[b] P.S. Bowling,[a] K.L. Buescher,[b] V.M. Hanagandi,[b]
R.F. Hinde, Jr.,[c] R.D. Jones,[d] W.J. Parkinson[c]

[a] *Accelerator Operations and Technology Division*
[b] *Applied Theoretical Physics Division*
[c] *Engineering Sciences and Applications Division*
[d] *Center for Adaptive Systems Applications, Inc.*
Los Alamos National Laboratory, Los Alamos, NM 87545

ABSTRACT

One can derive a model for use in a Model Predictive Controller (MPC) from first principles or from experimental data. Until recently, both methods failed for all but the simplest processes. First principles are almost always incomplete and fitting to experimental data fails for dimensions greater than one as well as for non-linear cases. Several authors[1] have suggested the use of a neural network to fit the experimental data to a multi-dimensional and/or non-linear model. Most networks, however, use simple sigmoid functions and backpropagation for fitting. Training of these networks generally requires large amounts of data and consequently very long training times.

In 1993 we reported on the tuning and optimization of a negative-ion source using a special neural network[2]. One of the properties of this network (CNLSnet), a modified radial basis function network, is that it is able to fit data with few basis functions. Another is that its training is linear resulting in guaranteed convergence and rapid training. We found the training to be rapid enough to support real-time control.

This work has been extended to incorporate this network into an MPC using the model built by the network for predictive control. This controller has shown some remarkable capabilities in such non-linear applications as continuous-stirred exothermic tank reactors and high-purity fractional distillation columns[3]. The controller is able not only to build an appropriate model from operating data but also to train the network continuously so that the model adapts to changing plant conditions.

The controller is discussed as well as its possible use in various of the difficult control problems that face this community.

INTRODUCTION

Remarkable progress has been made during the last several decades in the control of many different kinds of processes. A large body of theory has been developed to aid in this venture. “Classical” control theory deals well with linear processes and even with some non-linear processes as long as they are unidimensional. The problem is that there are no processes that are truly linear. All processes demonstrate some non-linear characteristics. How then can we say we have made remarkable progress? The control “surfaces” of many processes, although non-linear, are treated as approximately linear, i.e., they contain areas that are very nearly linear that are connected by discontinuities. As long as control is maintained within these very nearly linear areas, the process can be controlled. Approximate methods have been developed that provide excellent control of such processes. However, processes that demonstrate true non-linearity and have more than one dimension turn out to be not only in the majority but represent the most interesting, i.e., those processes for which good control would be most beneficial. Many processes in the chemical industry fall into this category. The optimal tuning of a surface-plasma negative-ion source is another example. The ion source tuning experiment was discussed at the last ICALEPCS conference[2]. Several authors[1] have suggested that the use of neural networks be applied to such cases. These networks demonstrate the ability not only to address non-linear problems but also demonstrate the ability to adapt to processes through experimental data.

The perceptron is one of the oldest and most basic classes of adaptive networks[4]. It was born from an attempt to model the functions of the brain, hence the neuro-physiological terms. The fundamental unit is called a neuron. Several neurons are gathered into a mathematical layer. Several layers are mathematically stacked together making a network. Each neuron has a number of inputs each of which may have a different contribution to the output. To provide for this

*Work supported in part by the U.S. Department of Energy

difference, each input is allowed to have its own weight. These inputs are added together and the output is passed through a filter that turns on (provides an output of 1) if the sum exceeds some value or turns off (provides an output of 0) if the sum is lower than the critical value. Figure 1 attempts to demonstrate this idea. A neuron of this type is able to

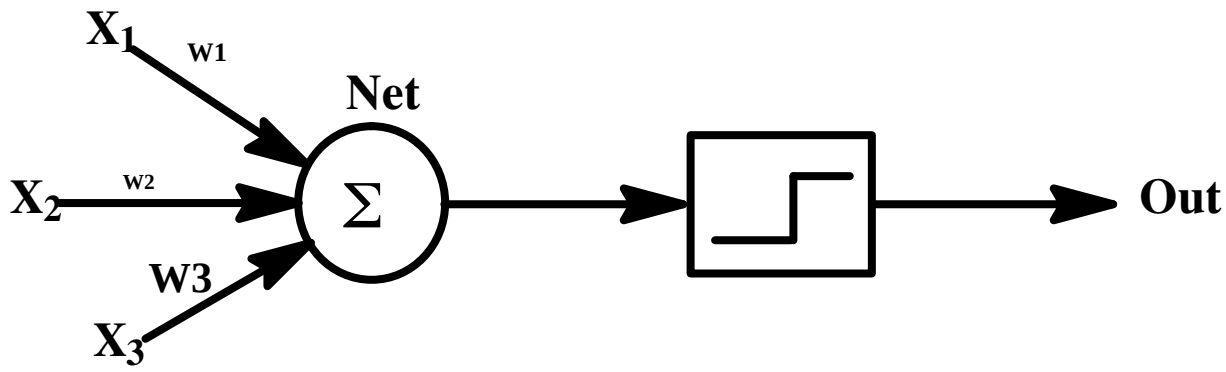

Figure 1. Diagram of a perceptron showing its functional parts.

perform linear discrimination, i.e., can classify inputs as belonging to one particular set or another since it uses a linear combination of inputs to perform the classification. By adding more neurons and allowing them to interact, more than one discriminant can be generated providing for classification into more than one set. Thus, a network of two neurons with two inputs, each inputting to both neurons, and two outputs is able to generate the two input/two output truth tables for the logical functions AND, OR, NAND, and NOR. One of the big disappointments in the early days was the inability of a single layer perceptron such as just described to represent the simple XOR function. However, by adding an additional layer the XOR can be simulated.

The process of weight adjustment is often called “learning” and it is this feature that makes neural networks so useful. As new input/output pairs are generated they are used to adjust the weights. Rosenblatt[4] proposed a training algorithm for single layer perceptrons:

1. Apply the input vector and calculate the output.
2. Adjust the weights.
    a. If the output is correct, perform no adjustment.
    b. if the output is zero but should be one, add each input to its corresponding weight.
    c. If the output is one but should be zero, subtract each input from its corresponding weight.
3. Repeat steps 1. and 2. for all training vectors as many times as necessary.

Symbolically this becomes:

$$\delta = Target - OUT$$
$$\Delta_\iota = \eta\ \delta\ IN_\iota$$

We’ve introduced η as a learning rate so the speed of learning can be controlled. The weight adjustment for the $n+1$st iteration step becomes:

$$w_i^{n+1} = w_i^n + \Delta_i^n$$

Since the hidden layers, if there are any, are not connected to the output, it is impossible to train them this way. We can, however, form an error term in the usual way, i.e., calculate the sum of the squares of the differences of all the targets and the outputs. This error term will be a minimum when the output of the network is a good least squares approximation to the target output. If this error term depends smoothly on the weights, we can simply differentiate the error term with respect to the weights and change the weights so that the error is moved in the direction of the negative of the gradient. When the gradient reaches zero, we will have reached a minimum (albeit it might be a local minimum). There is a problem with the perceptron, however. The step function filter that we used is not a smooth (differentiable) function. We can, however, approximate to it with a smooth function called a sigmoid. Figure 2 shows a sigmoid function. The reduction of the overall error which caused us to change to a sigmoid function was required to provide for training of neurons in the hidden layer(s). This process of backing the error into the hidden layer is known as

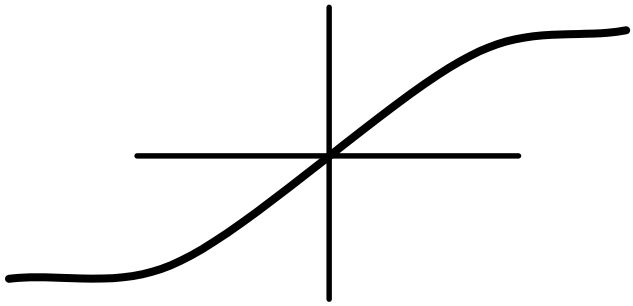

Figure 2. A sigmoid function.

backpropagation. The multilayer perceptron with backpropagation learning is currently the most popular network for real applications. It has the nice property of being able to extract global features from data that would normally be hard to find. It does have some problems, however, which must be either dealt with or lived with[5]:

1. There is nothing to guarantee that the minimum found by the learning algorithm is or is even close to the global minimum.
2. Weights can become so large that the sigmoids become saturated and paralysis of the learning can occur.
3. Training data can be overfit, thus decreasing the ability of the network to generalize.
4. Training is often very slow and requires much data.

Since this paper is not about perceptrons, why, one might ask, have we gone through all of this. There are two reasons. Firstly, from an historical perspective it is interesting and therefore good information to have and secondly, the reasons for and methods of training all types of networks are based on the methodology just described. Since the multilayer perceptron is the most popular network, often when people consider neural networks, the multilayer perceptron is the only network considered. If one purchases a network package for a computer it most likely will be a multilayer perceptron that uses a backpropagation algorithm for training. There are other networks. The body of this paper will consider a different network.

NETWORKS AS FUNCTION APPROXIMATIONS

What is really happening when we apply training data to a set of functions and adjust the weights of the inputs? In reality, we are approximating the relationship of a set of input data to an output function by using a set of basis functions. If we were to generate a perfect fit, we would be able to use a set of coordinates for which there had been no training and would be able to predict what the output would be. In other words, by accomplishing a multidimensional “curve” fit, we will have generated a model of the process from which we had obtained operating data. When fitting a function with a set of basis functions the more properties the basis functions have in common with the function to fit, the fewer bases will be required and the faster the fit will converge. Note that a one-dimensional periodic function is easy to fit with a series of sine functions, a Fourier series. If one thinks about fitting that same periodic function with a set of sigmoid functions, it is intuitively obvious that many sigmoids would be required to supply all the inflection points, positive and negative slopes, maxima and minima, etc., that would be required. This is one of the reasons that a network built of multilayer perceptrons requires many nodes and much training data. We must expose the network to all the vagaries of the function we’re trying to fit because the sigmoid doesn’t extrapolate with anything that is interesting. The question then becomes, what kind of basis function can we use that extrapolates better and contains more of the properties of the function we’re trying to fit.

Using ideas of self organization and recurrancy and introducing a cost function that is related to physical properties such as free energy and entropy, one can derive a basis function with some really nice properties. Detailing those arguments and providing the derivation is far too burdensome for this paper. However, one can demonstrate the network and training algorithms by just choosing a basis function of a particular form. That is the approach we will

take. Let us identify the function we're trying to approximate (fit) by $g(\vec{x})$. We begin with the identity:

$$g(\vec{x}) = \frac{\sum_{j=1}^{N} g(\vec{x})\varrho_j(\vec{x})}{\sum_{k=1}^{N} \varrho_k(\vec{x})}$$

Now here is the magic. We define the basis functions as multidimensional Gaussian functions:

$$\varrho_j(\vec{x}) = \beta_j e^{[-\beta_j|\vec{x} - \vec{x}_j|^2]}$$

where β_j is the width of the Gaussian and $\vec{x}_j$ is the center and $\vec{x}$ is one of the input vectors. This is called a modified radial basis function (RBF). Note that the function spreads in all dimensions from the center. $g(\vec{x})$ can be approximated by a Taylor expansion about $\vec{x}_j$ where only the first two terms are retained:

$$g(\vec{x}) \simeq \phi(\vec{x}) = \sum_{j=1}^{N}[f_j + (\vec{x}–\vec{x}_j) \cdot \vec{d}_j]\frac{\varrho_j(\vec{x})}{\sum_{k=1}^{N} \varrho_k(\vec{x})}$$

The f_j is the zero order term in the Taylor expansion and the d_j and its associated coefficient is the gradient term. Defined in this way, this network differs from the usual RBF in two ways:

1. a normalization term and
2. the addition of the gradient term which, by the way, is linear.

We will regard these terms as adaptive weights and since they are linear, the training can be very fast. Further, since they are linear, the cost function (least square) is quadratic in both and consequently has only a single minimum. Because the basis function itself is multi-dimensional and has, in any single dimension, both positive and negative curvature as well as two inflection points and a maximum, it takes on more of the character of the function we're trying to fit. Therefore, it takes fewer basis functions to generate a good fit. Because of these properties, it also extrapolates well which means it requires less training data. We will use these properties to good benefit.

A training method can be derived using gradient ascent. We will add a learning rate. The method is known as the α-LMS method. Using η as the learning rate coefficient, the learning method for both *f* and *d* follow:

$$f_j^{p+1} = f_j^p + \eta[g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{\varrho_j(\vec{x}_p)\sum_{k=1}^{N} \varrho_k(\vec{x}_p)}{\sum_{l=1}^{N}[\varrho_l^2(\vec{x}_p) + \beta_l(\vec{x}_p - \vec{x}_l)^2\varrho_l^2(\vec{x}_p)]}$$

$$d_j^{p+1} = d_j^p + \eta[g(\vec{x}_p) - \phi(\vec{x}_p)]\frac{\beta_j(\vec{x}_p - \vec{x}_j)\varrho_j(\vec{x}_p)\sum_{k=1}^{N} \varrho_k(\vec{x}_p)}{\sum_{l=1}^{N}[\varrho_l^2(\vec{x}_p) + \beta_l(\vec{x}_p - \vec{x}_l)^2\varrho_l^2(\vec{x}_p)]}$$

This network has been named by its originators the Connectionist Normalized Local Spline (CNLS) Network. We will refer to it as CNLSnet. To reiterate, the CNLSnet interpolates smooth functions well; it learns fast; local minima are not a problem; the network does not become paralyzed at extreme weight values; few nodes are required to approximate smooth functions. However, CNLSnet will fail if given too much irrelevant or redundant data. This is typically not a problem when it is used as part of a control strategy.

MODEL PREDICTIVE CONTROL

Much of "classical" control theory depends on the closed loop feedback of an error between the target value where we want the plant to be and where the plant actually is. This error is then passed through a transform to provide a correction term to the control. This is a linear process and depends on the linearity of the plant for performance. One can play a few tricks to maintain good linear control even with non-linear plants. The most straightforward way is to operate the plant

in a limited region maintaining approximate linearity. Of course, this won't work in all cases. Another trick often used with PID controllers is to generate a table of different gains that are applied as the process moves from one region to another. This is called gain scheduling. In this way, good and sometimes even excellent control may be obtained. By paying attention to plant design and optimizing the process, efficiencies of ninety-eight percent[1] are quite typical in the petrochemical industry. The remaining two percent inefficiencies are normally due to the difficulty of optimization during transient conditions when process parameters change in an unplanned or nonlinear manner. Good nonlinear optimization might raise this to ninety-nine percent efficiency. Even though this doesn't seem much it could mean a $10 million dollar per year savings for a billion dollar per year plant.

Model predictive control breaks the feedback and makes control decisions based upon a comparison between where the plant is and where the model of the plant says it should be. With a perfect model one should be able to obtain perfect control. Although model predictive control has been widely used in the chemical and petroleum industries[1], difficulties in producing good first-principle models has limited its effectiveness. The usual procedure is to develop a linear model, based on empirical data and use that model in an optimization routine. Artificial neural networks provide another avenue. Using the same empirical data a neural network model of the plant is built. Because neural network models can "learn" to represent any well-behaved nonlinear function[6], they should be able to represent the plant more accurately, and thus lead to better performance.

Before describing the controller that is the subject of this paper, we need to discuss two uses of a neural network model and the implications of those uses on the training of the network. The first type of use was reported on during ICALEPCS '93[2]. In this application, an attempt was made, prior to any process control, to fully map and model the control surface. Thus, training data was gathered by incrementing the four independent variables over their allowed ranges. This generated 2401 data points. The model was tested against the control surface by finding the optimum operating point in the model and setting the control variables accordingly. Once the process had settled, the real operating point was compared to the model. If the model had not yet converged, that input/output vector was added to the training data and the network was retrained. This process continued until agreement was reached between the model and the process, in this case a negative ion source. At this point, real process control could begin and operation was optimized. This was basically a steady-state process with little, if any, transient problems. Long term, slow drifts seemed to be the only transients. However, the model seemed to change from one operating period to another thus the optimal operating point was unknown from one operating period to the next. This meant that the model had to be rebuilt each time the ion source was started. This type of model building and control was most appropriate for his kind of process.

Another usage is characteristic of processes where *a priori* knowledge indicates where the process should be operated. However, the output parameters may be dramatically changed during operation and their inputs or operating parameters may undergo drastic transients. This is characteristic of an ultra-purity fractional distillation column. The column is normally started with the same operating conditions. Once normal operation is attained the schedule may call for changes in the output to generate different products or the feed to the column can change dramatically. Also, most of these columns are outside, exposed to the weather and although they are normally well insulated, a rain or snowstorm can cause drastic changes in the operating parameters. It is the job of the controller to correct for these changes and maintain desired operation. The total control surface is not of much interest in this scenario. Further, because of the costs involved, nominal operation must be attained as quickly as possible. The operating scheme would be to fit the model to a few localized points around the normal operating point. As the process changes or is changed the controller will adapt to the new conditions by first extrapolating the model and then building a new model with the new data as it becomes available. Since training and adaptation is a continuing process, it can be done with a small amount of data and is, therefore, very fast. If time is included in the training vector, the model will also be able to predict temporal trajectories. Using this data, the process output can be made to follow certain desired trajectories as the operating parameters are changed, e.g., reduce or even eliminate the amount of overshoot as the process approaches some setpoint.

The procedure that is used to minimize the least-square error is a conjugate direction method known as the Fletcher-Reeves method[7]. This method is better than gradient descent because it uses line searches in directions that are orthogonal to the previous search direction. It has additional requirements, however. It requires both a training differential and a training gradient. It turns out that these are easy to come by. Recalling:

$$g(\vec{x}) \simeq \phi(\vec{x}) = \sum_{j=1}^{N} [f_j + (\vec{x}-\vec{x}_j) \cdot \vec{d}_j] \frac{\varrho_j(\vec{x})}{\sum_{k=1}^{N} \varrho_k(\vec{x})}$$

This can be written:

$$\phi(\vec{x}) = \vec{w}^T \cdot \vec{\varrho}(\vec{x}).$$

Since this is linear we can form:

$$\vec{\phi}_{t+1}(\vec{x}) - \vec{\phi}_t(\vec{x}) = \Delta\vec{\phi}(\vec{x}) = (\vec{w}_{t+1} - \vec{w}_t)^T \cdot \vec{\varrho}(\vec{x})$$

$$\Delta\phi(\vec{x}) = \Delta\vec{w}^T \cdot \vec{\varrho}(\vec{x}).$$

We now have the training differential. It uses the same basis functions and the same training methods. We play the same trick one more time to obtain the gradient.

LOS ALAMOS MPC CONTROLLER

The MPC controller that is the subject of this paper uses CNLSnet to build its model. It uses temporal data to provide trajectories so that as setpoints are changed it will follow a predetermined function as it changes the control parameter with time. It does this by taking one time step and comparing where it thinks it should be with where the trajectory says it should be. The next step corrects any error as well as moving toward the setpoint. This continues until the setpoint is reached. If large enough errors are found during this process, network training takes place. Actually, one additional provision has been built-in. When first starting control, the controller has only three basis functions (actually this number is adjustable but three seems to work well). As it controls the process, if it finds a large enough error between where it thinks it is and where it really is, it has the ability to add additional basis functions up to a maximum, another adjustable controller parameter. The controller also has the ability to move a basis function to a more appropriate place should that be deemed necessary. This provides the controller with the greatest amount of flexibility as well as targeting only the appropriate part of the control surface.

To date, the controller has been applied to two simulations and has been installed in a commercial control system in preparation for implementation on a high-purity fractional distillation column in Ponca City, OK. The two simulations are an exothermic chemical reaction in a Continuously Stirred Tank Reactor (CSTR) and the high purity distillation of tritium in a set of four, ganged, cryogenic fractional distillation columns at the Tritium System Test Assembly at Los Alamos.

The equations that govern the behavior of the CSTR follow.

$$\frac{dX_A}{dt} = -X_A + D_a(1 - X_A)e\left(\frac{T_R}{1 + \frac{T_R}{\gamma}}\right)$$

$$\frac{dT_R}{dt} = -T_R + BD_a(1 - X_A)e\left(\frac{T_R}{1 + \frac{T_R}{\gamma}}\right) + \beta(T_J - T_R)$$

$$\frac{dT_J}{dt} = \delta_1\delta_2(T_{J_0} - T_J) + \beta_J\delta_1(T_R - T_J)$$

All parameters are dimensionless. X_A is the concentration of species *A*. T_R is the temperature inside the reactor. The temperature of the jacket is T_J and T_{J_0} is the temperature of the heat bath which controls the temperature inside the reactor. The rest of the parameters are constants such as heat transfer coefficients, activation energies, etc. These equations clearly demonstrate the non-linear nature of the process. Figure 3 is a schematic of a CSTR showing the reaction vessel, the jacket and the stirrer. The schematic also shows some of the controllers that are used to control the process.

Figure 4 is the steady state operating curve of the CSTR. Note that except at the points where the gain changes sign, T_R varies approximately linearly with T_{J_0}.

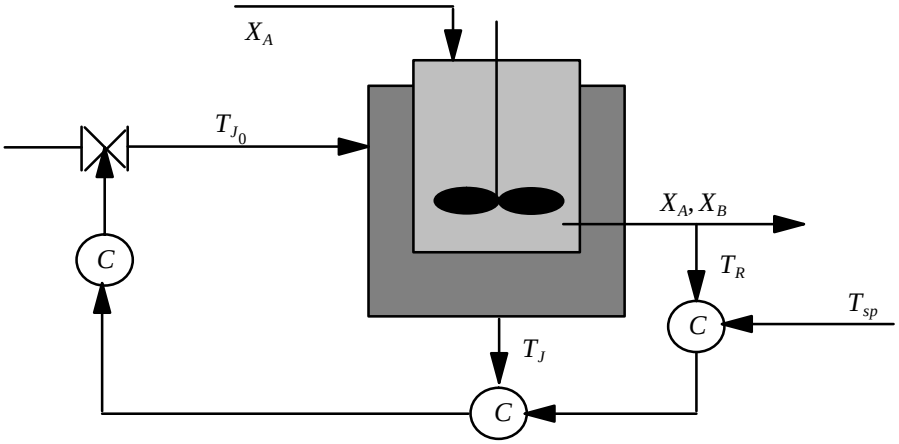

Figure 3. Diagram of CSTR

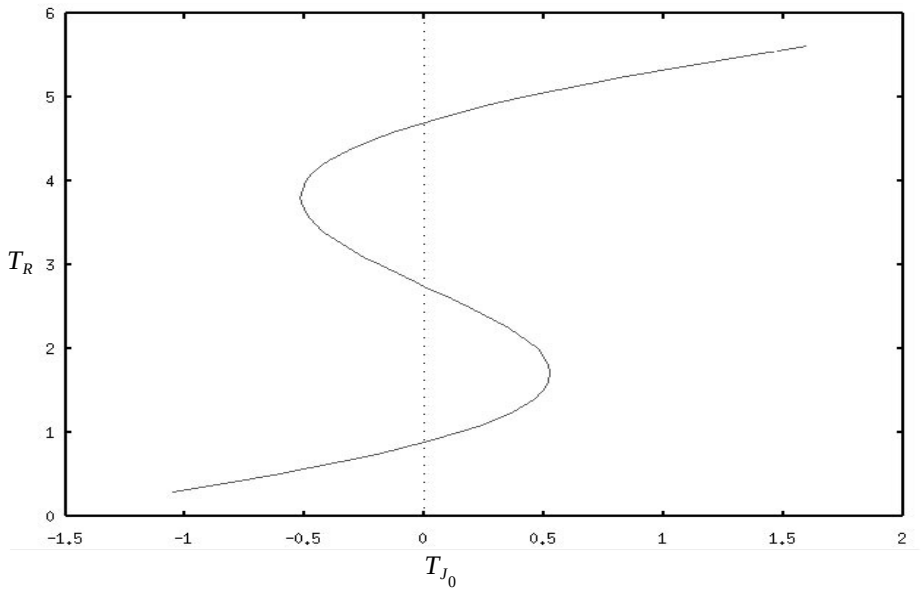

Figure 4. Steady State Operating Curve of CSTR

It turns out that the best operating place is on the part of the curve that exhibits negative gain. Controlling the process in this region is normally very difficult. This makes an interesting trial for the MPC controller. Figure 5 shows the data taken during the training run. Note the limits of the unstable region. When the training session started the control was set for the process to be below the unstable region and it settled into a steady state. The control was raised but not so much as to cross the stability boundary then was lowered. The process followed along as expected. The control was increased again but this time to a point so that the stability boundary was crossed and the process crossed the unstable region and operated on the upper part of the curve. The control was then reduced and the process returned through the unstable region to the lower part of the operating curve. It is important to note the relatively few data points that were generated while the process was in the unstable reason. The reason that this is important is because that is where we are going to attempt to run the process. Figure 6 demonstrates just such a run. Note that whether there is a change in the setpoint or change in the plant (feed disturbance), the MPC is able to hold the process on setpoint in the unstable region as well as the stable region. When the MPC was applied to the simulation of a high-purity fractional distillation it again performed well, maintaining control in both linear and non-linear regimes.

CONCLUSIONS

The capabilities of model predictive control where the model is built by a neural network and is continuously adapted to the process has been clearly demonstrated. The controller has been taken from a theoretical proof-of-principle to a

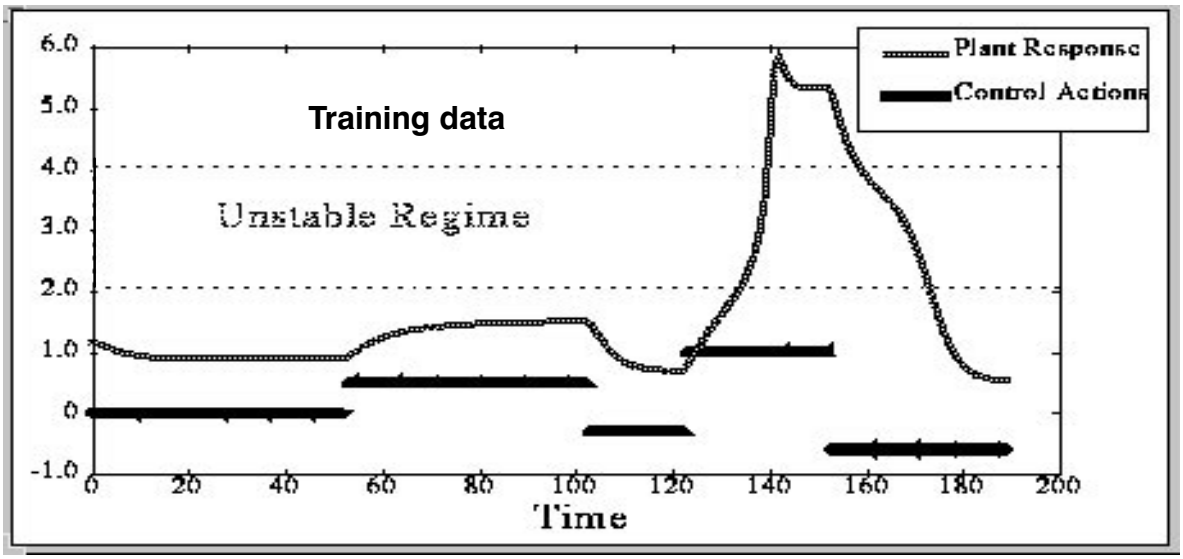

Figure 5. Data taken during training

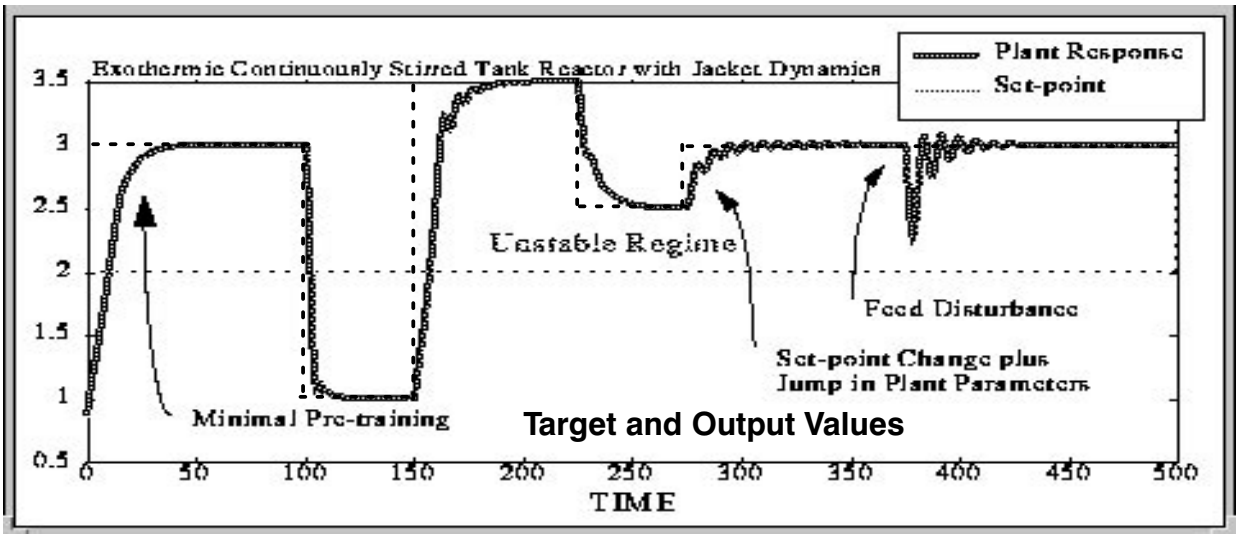

Figure 6. Control run showing MPC maintaining control in unstable region

controller ready for commercial use. This controller can be used in many different control strategies. We have shown its capability in two. In some sense, it might be more appropriate to take the place of PID control in linear processes because one would not have to worry about finding appropriate gains for the PID control. The MPC will adapt to the process. The trade-off is a bit more complexity with which to deal. We are in the process of attempting to implement this technology to provide adaptability in a state space formalism. The impetus for this work comes from the beam steering feedback done several years ago at SLAC and reported during ICALEPCS '91[8]. We will continue to improve on this technology as new and interesting control problems present themselves.

REFERENCES

[1] P.J. Werbos, T. McAvoy, and T. Su, **Handbook of Intelligent Control** (Van Nostrand Reinhold, New York, NY, 1992) 286.

[2] W.C. Mead, S.K. Brown, R.D. Jones, P.S. Bowling, C.W. Barnes, *Adaptive Optimization and Control Using Neural Networks*, Proceedings of the 1993 ICALEPCS Conference, Berlin, Germany, 1993, 309–315.

[3] C.C. Baum, K.L. Buescher, R.D. Jones, W.J. Parkinson, and M.J. Schmitt, *Two-Timescale, Model Predictive Control of Two Simulated Chemical Plants: Lagged-CSTR and Distillation Column*, Los Alamos Report

(LA-UR-94-1039).
[4] F. Rosenblatt, “The perceptron: A probablilistic model for information stage and organization in the brain,” *Psychological Review*, **65**, (1958).
[5] J. Galbraith, R.B. Webster, C.D. Bergman, R.D. Jones, **Introduction to Adaptive Computation**, Unpublished book.
[6] E. Blum, Approximation theory and feedforward networks, *Neural Networks*, 4, 1991.
[7] D.G. Luenberger, **Linear and Nonlinear Programming**. Reading, MA, Addison Wesley, 1984.
[8] L. Hendrickson, S. Allison, T. Gromme, T. Himel, K. Krauter, F. Rouse, R. Sass and H. Shoaee, *Generalized Fast Feedback System in the SLC*, Proceedings of the 1991 ICALEPCS Conference, Tsukubu, Japan, 1991.