Simulation, State Estimation and Control of Nonlinear Superheater Attemporator using Neural Networks

Bendtsen, Jan Dimon; Sørensen, O.

Publication date:
1999

Document Version
Også kaldet Forlagets PDF

Simulation, State Estimation and Control of Nonlinear Superheater Attemporator using Neural Networks

Jan Dimon Bendtsen* Ole Sørensen*

May 18, 1999

Abstract

This paper considers the use of neural networks for nonlinear state estimation, system identification and control. As a case study we use data taken from a nonlinear injection valve for a superheater attemporator at a power plant. One neural network is trained as a nonlinear simulation model of the process, then another network is trained to act as a combined state and parameter estimator for the process. The observer network incorporates smoothing of the parameter estimates in the form of regularization. A pole placement controller is designed which takes advantage of the sample-by-sample linearizations and state estimates provided by the observer network. Simulation studies show that the nonlinear observer-based control loop performs better than a similar control loop based on a linear observer.

Keywords: Neural Networks, Nonlinear State Estimation and Control, Power Plant Control, Extended Kalman Filters.

1 Introduction

In control applications it is not always the most feasible to model the process in question using first principles. The process can be difficult to model accurately, or there can be nonlinear elements in the process which can be difficult or costly to quantify, and which may change from nominal values when the process is running. Such issues as friction, stiction, saturation, and general wear can produce unexpected effects. It is therefore of interest to examine data-driven approaches to modelling and control, and for nonlinear processes it is natural to choose nonlinear system identification methods.

Neural networks have been considered for control purposes in several publications, notably: [5] where model reference adaptive control is studied; in [7] an inverse, dead-beat controller is considered; see also the references in [3]. More recent studies include [6], in which the observability problem is addressed, and [8] on which the present paper to some extend builds.

*With the Department of Control Engineering, Aalborg University, Fredrik Bajersvej 7C, DK-9220 Aalborg East, Denmark. Email: dimon@control.auc.dk , os@control.auc.dk.

In this paper a neural network is trained as a nonlinear simulation model of a valve used to control the temperature of steam fed into a power plant superheater. After training this network acts as a simulator for the control object. A neural network is normally considered to be a non-parametric model, but we will show how to estimate linearized parameters from the network weights and neuron functions. We then train another neural network model where we incorporate *regularization* to avoid too rapid fluctuations in said parameter estimates. This network is used as a state observer and parameter estimator at the same time, thus in essence solving the extended Kalman filter problem. The chosen control law is recalculated at every sample time with respect to the sample-by-sample parameter and state estimates.

Remark 1 Using this approach is in fact equivalent to using *Gain Scheduling* control, with an infinite resolution. At any given operating point (which does not have to be an equilibrium) we extract a linearized state space model which is valid in and immediately around that particular point. We then calculate a control law based on this local linearized model. ◁

1.1 The valve control loop

The application takes its starting point in an existing attemporator control loop where a nonlinear hollow-cone valve acts as an actuator for a cooler before a superheater. This particular control loop is situated at the Danish power plant I/S Vestkraft Unit 3.

The segment of the power plant's steam circuit where the control loop in question is found is shown in figure 1. The cooler itself works by injecting cooling water into the steam flow, thereby lowering the temperature of the steam. The cooling water is fed to the cooler through the hollow-cone valve which has a nonlinear profile with saturation and varying slope (see figure 2).

One of the major difficulties in modeling the valve characteristics is that it changes slowly over time, because the cooling water that flows through the valve slowly clogs the small holes in the cone with magnetite. In effect the valve gets a more and more pronounced dead-zone nonlinearity, as indicated in the figure. Thus it will be desirable to model and control the valve loop based on measurements rather than first principles.

The outline of the rest of the paper is as follows. Section 2

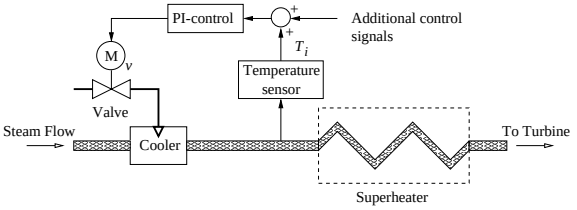

Figure 1: *Sketch of the valve and cooler loop. The relevant control variables are the valve setting v and the inlet steam temperature T_i. The term 'Additional control signals', whenever applicable, covers outside control influences such as reference setpoints, feedback from the outlet steam temperature from after the superheater or additive feedforward [4].*

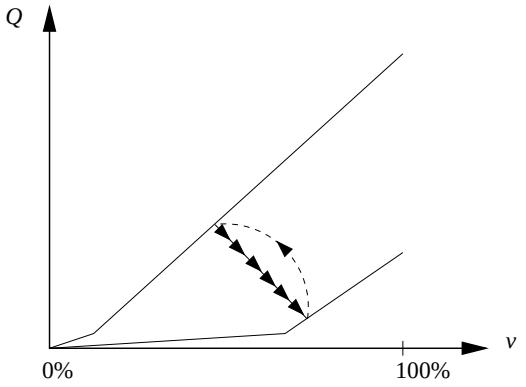

Figure 2: *Sketch of varying valve characteristics as a function of the valve position v. As time passes the valve is gradually clogged, decreasing the flow Q through the valve at a given v. During plant stops the valve is cleaned, causing the characteristic to return to the starting curve.*

discusses the chosen network structure and training algorithm as well as how to extract the parameter estimates. Section 3 addresses the control law, while section 4 shows some simulation results. Finally we conclude with a few remarks in section 5.

2 Modeling

A discrete-time nonlinear system will in the present context be one that can be written on the following general form:

$$x_{k+1} = f(x_k, u_k, d_k) \quad (1)$$
$$y_k = h(x_k) \quad (2)$$

in which k is the discrete sample number, $x \in \mathbb{R}^n$ is a state vector, $u \in \mathbb{R}^m$ is the input vector and $y \in \mathbb{R}R^p$ is the output. $d \in \mathbb{R}^{m_d}$ is a vector of disturbances included to improve the modeling capabilities. Whenever relevant, we will furthermore include a vector $\theta \in \mathbb{R}^{n_\theta}$ containing some set of parameters sufficient to describe the model at time k in the description. They are generally assumed to vary much slower than the states x.

f and h are given functions which may be nonlinear, time-varying and multi-variable. They are assumed to behave 'reasonably', though; that is, they are supposed to be locally Lipschitz on the relevant compact subset of the state space.

We wish to identify a nonlinear (neural network) mapping $\mathcal{M}$

$$\hat{x}_{k+1} = \mathcal{M}(\hat{x}_k, u_k, d_k, \varepsilon_k) \quad (3)$$
$$\hat{y}_k = H\hat{x}_k \quad (4)$$
$$\varepsilon_k = y_k - \hat{y}_k \quad (5)$$

based on samples of system in- and outputs $\{u_k, y_k\}_{k=0}^N$ (the *training set*) such that the prediction error ε defined by equations (3)–(5) becomes small. $\mathcal{M}$ is chosen to be a multi-layer perceptron (abbreviated MLP) and $\hat{(\cdot)}$ indicates estimates. We choose the output mapping to be fixed, $H = \begin{bmatrix} I & 0 \end{bmatrix}$, where I and 0 are identity and zero matrices of appropriate dimensions.

2.1 The MLP

The MLP is composed of layers of parallel couplings of single perceptrons. A perceptron is essentially a simple function $\phi : \mathbb{R} \to \mathbb{R}$ acting on a weighted sum of input signals

$$z = \phi\left(\sum_{i=1}^{n} \theta^i z_{in}^i + \theta^b\right)$$

The neuron function ϕ can be either linear or nonlinear; the functions traditionally used in multi-layer perceptron neurons are the unit gain, the hyperbolic tangent, the sigmoid and the gauss functions. We will in this paper only use $\tanh(\cdot)$ and linear neurons functions.

When several perceptrons are joined, the result is an MLP. Parallel groupings of neurons are called layers, and layers of perceptrons which do not produce an output from the network as such are called hidden layers. It is chosen only to consider MLPs with a single hidden layer since this is the simplest structure and it has been shown that provided there is a sufficient number of neurons in the hidden layer the MLP can act as a universal approximator [2].

A compact representation of the MLP can be achieved by collecting the weights leading to each neuron as rows in a weight matrix Θ, the biases in the vector θ_b, the in- and outputs to and from the network in two vectors z_{in} and z_{out} and the neuron functions in a single vector function ϕ. This produces the block diagram shown in figure 3.

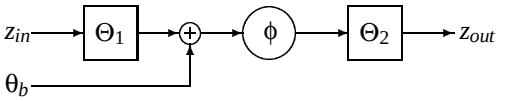

Figure 3: *Matrix block diagram of an MLP.*

The output from the multi-layer perceptron can thus be written as:

$$z_{out} = \Theta_2\phi(\Theta_1 z_{in} + \theta_b) \quad (6)$$

Accordingly the network in- and output vectors are chosen as

$$z_{in} = \begin{bmatrix} \hat{x}_k^T & u_k^T & d_k^T & \varepsilon_k^T \end{bmatrix}^T \text{ and } z_{out} = \begin{bmatrix} \hat{x}_{k+1} \end{bmatrix}$$

Because the state estimates are fed back and used as inputs to the network, it is necessary to use recursive training of the model. We wish to minimize the squared prediction error; con-

sequently the following recursive performance functional will be used:

$$J_k = \sum_{i=1}^{k} \lambda^{k-i} \frac{1}{2} \varepsilon_k^T \varepsilon_k = \lambda J_{k-1} + \frac{1}{2} \varepsilon_k^T \varepsilon_k$$

where $\varepsilon_k = y_k - \hat{y}_k = y_k - H\hat{x}_k$. The so-called forgetting factor λ is chosen slightly less than 1, e.g. $\lambda = 0.997$. This factor ensures that old, outdated information is 'forgotten' with time.

The Back Propagation Error Algorithm which is normally used for training multi-layer perceptrons is a first-order gradient method and thus it is not very efficient when the performance surface does not change rapidly. For this reason it is often advantageous to use a second-order search method, such as the Gauss-Newton algorithm. This approach to minimum search is based on a second-order Taylor expansion of the performance function at iteration i. At the minimum performance the Taylor expansion is zero, and the parameter update can after rearranging terms be written as

$$\theta_{i+1} = \theta_i - \left(\frac{d^2 J}{d\theta_i d\theta_i^T} \right)^{-1} \frac{dJ}{d\theta_i}$$

where θ is a vector comprised of all the weights in the MLP model. This is a well-known and quite effective minimization rule. Like the Back Propagation Error Algorithm it can onlt guarantee finding a local minimum of the performance surface and a certain amount of trial-and-error must be expected when using it. It works quite well in practice, however. In order to update the parameters we need recursive expressions for the model gradient ψ, the performance gradient G_J and the Hessian matrix H_J. The algorithm is summarized in the box.

At sample k calculate:

1. State estimates: $\hat{x}_k = \mathcal{M}(\theta_k, z_{in})$
2. Prediction error: $\varepsilon_k = y_k - H\hat{x}_k$
3. Model gradient: $\psi_k = \frac{d\hat{y}_k^T}{d\theta} = \frac{d\mathcal{M}(\theta_{k-1}, z_{in})^T}{d\theta} H^T$
4. Performance gradient: $G_{J,k} = -\psi_k \varepsilon_k$
5. Hessian: $H_{J,k} = \lambda H_{J,k-1} + \psi_k \psi_k^T$
6. Parameters: $\theta_{k+1} = \theta_k + H_{J,k}^{-1} G_{J,k}$

Next sample

2.2 Parameter estimation

The small-signal gains over the MLP can be found by differentiating with respect to the inputs:

$$M_k = \frac{dz_{out}}{dz_{in}^T} = \frac{d\mathcal{M}(z_{in})}{dz_{in}^T} \tag{7}$$

Assuming the MLP is sufficiently trained, it estimates the state vector in the model

$$\begin{aligned} \hat{x}_{k+1} &= \hat{\Phi}_k(y_k)\hat{x}_k + \hat{\Gamma}_k(y_k)u_k + \hat{\Gamma}_k^d(y_k)d_k \\ &\quad + \hat{K}_k(y_k)\varepsilon_k \qquad (8) \\ \hat{y}_k &= H\hat{x}_k \qquad (9) \end{aligned}$$

in effect solving the extended Kalman filter problem. We can find the parameter matrices in that particular model structure since differentiation of equation (8) with respect to the network input yields:

$$\frac{d\hat{x}_{k+1}}{dz_{in}^T} = \begin{bmatrix} \hat{\Phi}_k & \hat{\Gamma}_k & \hat{\Gamma}_k^d & \hat{K}_k \end{bmatrix} = M_{k+1} \tag{10}$$

Thus, the gain matrix may be interpreted directly as estimates of the matrices in the sample-by-sample linearized state space model. The gain matrix itself is calculated from the MLP weights by using equation (6) in equation (7) and applying the chain rule of differentiation:

$$\begin{aligned} M_k &= \frac{d(\Theta_2 \phi(\Theta_1 z_{in} + \theta_b))}{dz_{in}^T} \\ &= \Theta_2 \frac{d\phi(\Theta_1 z_{in} + \theta_b)}{d(\Theta_1 z_{in})^T} \frac{d(\Theta_1 z_{in})}{dz_{in}^T} \\ &= \Theta_2 \phi'(\Theta_1 z_{in} + \theta_b)\Theta_1 \qquad (11) \end{aligned}$$

where ϕ' is short for the derivative of the neuron function vector with respect to its input vector.

2.3 Regularization

Ideally, the elements of the state and input matrices are smooth functions of the sample number. However, if the elements of $\hat{\Phi}_k$ and $\hat{\Gamma}_k$ are fluctuating rapidly there is a greater risk of losing reachability. Also, numerical problems in calculating the feedback control law may arise. As a consequence it is desirable to introduce some means of smoothing the parameter estimates while still allowing them to change in response to input and states.

In estimation theory, regularization is a method for obtaining models with good generalization abilities. Basically the idea is to place a prior probability density function on the parameters (weights) of the model, producing a regularized performance function

$$\mathcal{J}(\theta) = J(\theta) + \delta \frac{1}{2} (\theta - \vartheta)^T (\theta - \vartheta) \tag{12}$$

where δ is a scalar parameter that determines the degree of regularization. ϑ is a chosen, fixed parameter vector, typically 0.

After the differentiations of the performance functional it is observed that during training we simply need slightly modified versions of the gradients and Hessian matrices. The Gauss-Newton learning method with regularization can be summarized as in the following box. We will use this method to train an observer network which is connected in parallel with the plant network.

At sample k calculate:

1. State estimates: $\hat{x}_k = \mathcal{M}(\theta_k, z_{in})$
2. Prediction error: $\varepsilon_k = y_k - H\hat{x}_k$
3. Model gradient: $\psi_k = \frac{d\hat{y}_k^T}{d\theta} = \frac{d\mathcal{M}(\theta_{k-1}, z_{in})^T}{d\theta} H^T$
4. Perf. gradient: $G_{\mathcal{J},k} = -\psi_k \varepsilon_k + \delta\theta_k$
5. Hessian: $H_{J,k} = \lambda H_{J,k-1} + \psi_k \psi_k^T$
6. Regularized Hessian: $H_{\mathcal{J},k} = H_{J,k} + \delta I$
7. Parameters: $\theta_{k+1} = \theta_k + H_{\mathcal{J},k}^{-1} G_{\mathcal{J},k}$

Next sample

2.4 Simulation model

The data used for training and test set was collected during the spring of 1998. The data collection took place with a sample time of ten seconds, covering a total period of 485 hours. It was decided to use the first 1.5×10^5 samples (417 hours) as the training set. The measurements involved were: the valve setting (control signal u), the steam flow (disturbance d) and the steam temperature (output y). State estimates and prediction errors as defined above were also used in the network.

Several different MLP configurations were attempted in which we varied the number of states and neurons. The process was identified as being of second order, and it was found that the best simulation results were achieved with two linear and three tanh-neurons in the hidden layer and two linear neurons in the output layer after training for 15 epochs with the entire training set.

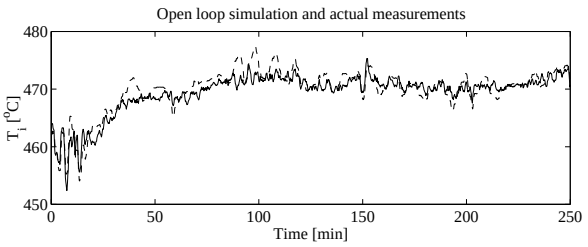

Figure 4: *Open loop MLP simulation (- -) of test set plotted together with the corresponding actual measurements (—).*

Figure 4 shows an open-loop noise-free simulation of a 1500 samples sequence. The simulated output (dashed line) is plotted together with the actual measured samples (full line). The simulation shows good agreement with the actual process, even without any correction from the process samples ($\varepsilon_k \equiv 0$ in the input to the network).

3 Control

Having trained a simulation model and an observer network we can now design the control law. Since full state information is not available, it is chosen as

$$u_k = -L\hat{x}_k \tag{13}$$

where L is a state feedback matrix which places the eigenvalues of the closed loop system.

In order to include integral action in the controller we will augment the description with an integral state. The discrete-time equivalent of a differentiation, $\Delta = 1 - q^{-1}$, is applied to the model, giving a differential model of the states:

$$\begin{aligned} \Delta\hat{x}_{k+1} &= \hat{\Phi}_k \Delta\hat{x}_k + \hat{\Gamma}_k \Delta u_k + \Gamma_k^d \Delta d_k \\ \Delta y_k &= H\Delta\hat{x}_k \end{aligned}$$

Then the augmented state space model is introduced in which the original state vector is supplemented with the integral output state. The augmented state vector is defined as:

$$x_k^{aug} = \begin{bmatrix} \Delta\hat{x}_k \\ y_k \end{bmatrix}$$

After a little manipulation we arrive at

$$\begin{aligned} x_{k+1}^{aug} &= \begin{bmatrix} \hat{\Phi}_k & 0 \\ H\hat{\Phi}_k & I \end{bmatrix} \begin{bmatrix} \Delta\hat{x}_k \\ y_k \end{bmatrix} + \begin{bmatrix} \hat{\Gamma}_k \\ H\hat{\Gamma}_k \end{bmatrix} \Delta u_k + \begin{bmatrix} \hat{\Gamma}_k^d \\ H\hat{\Gamma}_k^d \end{bmatrix} \Delta d_k \\ y_k &= \begin{bmatrix} 0 & I \end{bmatrix} x_k^{aug} \end{aligned}$$

We will denote the augmented matrices with $(\cdot)^{aug}$. For example, the notation $\hat{\Gamma}^{aug}$ will be used for the augmented input matrix including estimates.

Remark 2 In order to suppress the disturbance we include a simple least-squares attenuation $L^{\Delta d}$ in the feedforward path from the disturbance measurement. $\triangleleft$

It is seen that the characteristic polynomium for the linearized, augmented closed-loop system at sample number k is given by:

$$P_k(z) = \det(zI - (\hat{\Phi}_k^{aug} - \hat{\Gamma}_k^{aug} L_k^{aug}))$$

In other words, if the desired closed loop poles are given as the roots of the polynomial $P_d(z)$, then the control law $L_{aug,k}$ must be chosen so that $P_k(z) = P_d(z)$. This can for instance be achieved by solving *Ackermann's formula* (see e.g. [1]).

The controller can be separated into two contributions, one concerning the differential states and one concerning the integral state. To this is added the contribution from the external disturbance:

$$\begin{aligned} \Delta u_k &= -L_k^{aug} \hat{x}_k^{aug} - L_k^{\Delta d} \Delta d_k \\ &= -L_k^{\Delta x} \Delta\hat{x}_k - L_k^y y_k - L_k^{\Delta d} \Delta d_k \end{aligned}$$

So far, the control law has been aiming at driving the augmented states to 0. Introducing the reference output r_k and subtracting it from the output y_k yields a tracking error $e_k = y_k - r_k$, which can be driven to 0 instead. The final step in the regulator design is then to integrate the differential control signal (remembering that $\Delta u_k = u_k - u_{k-1}$):

$$u_k = -L_k^{\Delta x} \Delta\hat{x}_k - L_k^y e_k - L_k^{\Delta d} \Delta d_k + u_{k-1} \tag{14}$$

The control loop including disturbances and observer is shown in figure 5.

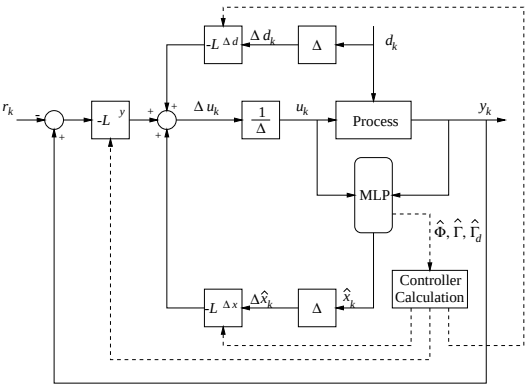

Figure 5: *Closed loop including the process model, the trained neural net observer and the state controller.*

4 Results

Simulations were run with observer networks with different regularization factors in order to find the most suitable observer. The best control results were achieved with a regularization factor of $\delta = 10^{-7}$. A one-hour simulation is shown in figure 6, in which the system is supposed to follow a series of steps. A disturbance sequence taken from the measured data and was used and the noise was set to zero, $\varepsilon_k \equiv 0$, on the input to the process network. The true system is of course not noise free, but in order to demonstrate the system behavior as clearly as possible it was chosen not to include noise in the simulations.

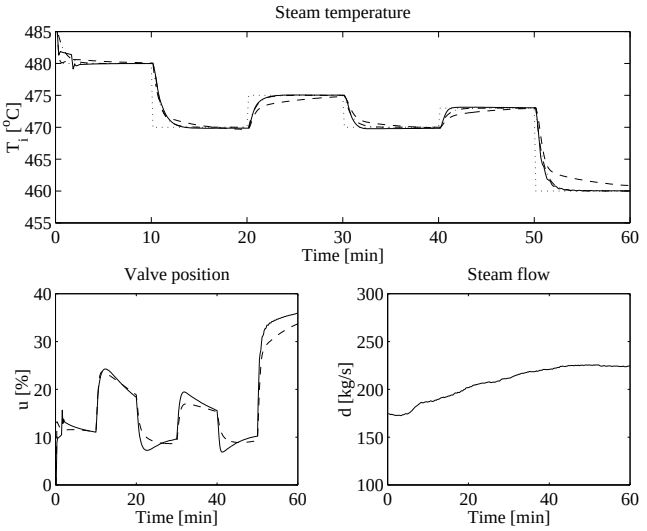

Figure 6: *Simulation of control loop. Top: Output. Bottom left: Control signal; Bottom right: Disturbance. Legend: reference (···), MLP-based control (—), linear observer-based control (– –), ideal system (– · –).*

The control law was tuned so that the system should be critically damped with a dominant double pole in $z = 0.8$. The desired characteristic polynomial was thus chosen as $P_d(z) = z^3 - 2z^2 + 1.28z - 0.256$. For comparison we simulated an ideal linear system with an identical set of poles following the same reference sequence, and it may be seen from the figure that the trajectories of the controlled system (full line) and the ideal system (dashdotted line) are almost identical.

A normal state observer based on linear system identification was also constructed and inserted in place of the MLP observer network in the control loop, again with the same desired pole location. The output of this control loop is also plotted in figure 6 (the dashed line). It can be seen that, even though the performance of this system may be deemed satisfying in itself, it is certainly not as close to the ideal trajectory as the MLP-observer-based control system.

5 Concluding remarks

In this paper the results of training a nonlinear neural network state space model of a pre-cooler process based on samples taken from the actual process were presented. It was identified as a second-order model and it was found that in addition to the control signal and the steam temperature (the output) it was necessary to include the steam flow through the pre-cooler as a disturbance. Using this model as a simulation model for the process, an observer-based pole placement controller which took advantage of the sample-by-sample linearizations also provided by the observer network was designed. The observer network was a regularized MLP, i.e. not identical to the process network. We compared the performance of the control law to an ideal system and a linear observer-based pole placement controller, and it was found that the MLP-observer-based control loop performed closer to the ideal than the linear version.

It is possible to make the algorithm presented in the paper adaptive by allowing online training of the observer network weights. This might indeed be a good idea in case of an actual test on the power plant system, since it is likely that the valve process will have changed its behaviour somewhat in the period of time that has passed since the collection of the data.

References

[1] K.J. Åström and B. Wittenmark. *Computer Controlled Systems*. Prentice-Hall International, 1989.

[2] K. Hornik et al. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[3] K.J. Hunt et al. Neural networks for control systems - a survey. *Automatica*, 6:1083–1120, 1992.

[4] T. Mølbak. Advanced control of superheater steam temperatures—an evaluation based on practical applications. *Control Engineering Practice*, 7:1–10, 1999.

[5] K.S. Narendra and A.U. Levin. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, 1990.

[6] K.S. Narendra and A.U. Levin. Control of nonlinear dynamical systems using neural networks: Observability, identification and control. *IEEE Transactions on Neural Networks*, 7:30–42, 1996.

[7] D. Psaltis, A. Sideris, and A. A. Yamamura. A multilayered neural network controller. *IEEE Control System Magazine*, 8:17–21, 1988.

[8] O. Sørensen. Neural network for non-linear adaptive control. In *Symposium on Robotics and Cybernetics*. IEEE, 1996.