SECOND EDITION

The Industrial Electronics Handbook

INTELLIGENT SYSTEMS

Edited by
Bogdan M. Wilamowski
J. David Irwin

CRC Press
Taylor & Francis Group

3

Neural Network–Based Control

Mehmet Önder Efe
Bahçeşehir University

3.1 Background of Neurocontrol

The mysterious nature of human brain with billions of neurons and enormously complicated biological structure has been a motivation for many disciplines that seem radically different from each other at a first glance, for example, engineering and medicine. Despite this difference, both engineering and medical sciences have a common base when the brain research is the matter of discussion. From a microscopic point of view, determining the building blocks as well as the functionality of those elements is one critical issue, and from a macroscopic viewpoint, discovering the functionality of groups of such elements is another one. The research in both scales has resulted in many useful models and algorithms, which are used frequently today. The framework of artificial neural networks has established an elegant bridge between problems, which display uncertainties, impreciseness with noise and modeling mismatches, and the solutions requiring precision, robustness, adaptability, and data-centeredness. The discipline of control engineering with tools offered by artificial neural networks have stipulated a synergy the outcomes of which is distributed over an enormously wide range.

A closer look at the historical developments in the neural networks research dates back to 1943. The first neuron model by Warren McCulloch and Walter Pitts was postulated and the model is assumed to fire under certain circumstances, [MP43]. Philosophically, the analytical models used today are the variants of this first model. The book entitled *The Organization of Behaviour* by Donald Hebb in 1949 was another milestone mentioning the synaptic modification for the first time [H49]. In 1956, Albert Uttley

reported the classification of simple sets containing binary patterns, [U56], while in 1958 the community was introduced to the perceptron by Frank Rosenblatt. In 1962, Rosenblatt postulated several learning algorithms for the perceptron model capable of distinguishing binary classes, [R59], and another milestone came in 1960: Least mean squares for the adaptive linear element (ADALINE) by Widrow and Hoff [WH60]. Many works were reported after this and in 1982, John J. Hopfield proposed a neural model that is capable of storing limited information and retrieving it correctly with partially true initial state [H82]. The next breakthrough resulting in the resurgence of neural networks research is the discovery of error backpropagation technique [RHW86]. Although gradient descent was a known technique of numerical analysis, its application was formulated for feedforward neural networks by David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams in 1986. A radically different viewpoint for activation scheme, the radial basis functions, was proposed by D.S. Broomhead and D. Lowe, in 1988. This approach opened a new horizon particularly in applications requiring clustering of raw data. As the models and alternatives enriched, it became important to prove the universal approximation properties associated with each model. Three works published in 1989, by Ken-Ichi Funahashi, Kurt Hornik, and George Cybenko, proved that the multilayer feedforward networks are universal approximators performing the superpositions of sigmoidal functions to approximate a given map with finite precision [HSV89,F89,C89].

The history of neural network–based control covers mainly the research reported since the discovery of error backpropagation in 1982. Paul J. Werbos reported the use of neural networks with backpropagation utility in dynamic system inversion, and these have become the first results drawing the interest of control community to neural network–based applications. The work of Kawato et al. and the book by Antsaklis et al. are the accelerating works in the area as they describe the building blocks of neural network–based control [KFS87,APW91]. The pioneering work of Narendra and Parthasarathy has been an inspiration for many researchers studying neural network–based control, or neurocontrol [NP91]. Four system types, the clear definition of the role of neural network in a feedback control system, and the given examples of Narendra and Parthasarathy have been used as benchmarking for many researchers claiming novel methods. Since 1990 to date, a significant increase in the number of neural network papers has been observed. According to Science Direct and IEEE databases, a list showing the number of published items containing the words *neural* and *control* is given in Table 3.1, where the growing interest to neural control can be seen clearly.

In [PF94], decoupled extended Kalman filter algorithm was implemented for the training of recurrent neural networks. The justification of the proposed scheme was achieved on a cart-pole system, a bioreactor control problem, and on an idle speed control of an engine. Polycarpou reports a stable, adaptive neurocontrol scheme for a class of nonlinear systems, and demonstrates the stability using Lyapunov theorems [M96]. In 1996, Narendra considers neural network–based control of systems having different types of uncertainties, for example, the mathematical details embodying the plant dynamics are not known, or their structures are known but parameters are unavailable [N96]. Robotics has been a major implementation area for neural network controllers. In [LJY97], rigid manipulator dynamics is studied with an augmented tuning law to ensure stability and tracking performance. Removal of certainty equivalence and the removal of persistent excitation conditions are important contributions of the cited work. Wai uses the neural controller as an auxiliary tool for improving the tracking performance of a two-axis robot containing gravitational effects [W03]. Another field of research benefiting from the possibilities offered by neural networks framework is chemical process engineering. In [H99,EAK99], a categorization of schemes under titles predictive control, inverse model–based control, and adaptive control methods are presented. Calise et al. present an adaptive output feedback control approach utilizing a neural network [CHI01] while [WH01] focuses on enhancing the qualities of output regulation in nonlinear systems. Padhi et al. make use of the neural network–based control in distributed parameter systems [PBR01]. Use of neural network–based adaptive controllers in which the role of neural network is to provide nonlinear functions is a common approach reported several times in the literature [AB01,GY01,GW02,GW04,HGL05,PKM09]. Selmic and Lewis report the compensation of backlash

TABLE 3.1 Number of Papers Containing the Keywords Neural and Control between 1990 and 2008

Year	Items in Science Direct	Items in IEEE
2008	988	1355
2007	820	1154
2006	732	1254
2005	763	843
2004	644	869
2003	685	737
2002	559	820
2001	565	670
2000	564	703
1999	456	749
1998	491	667
1997	516	737
1996	489	722
1995	380	723
1994	326	731
1993	290	639
1992	234	409
1991	187	408
1990	176	270

with neural network–based dynamic inversion exploiting Hebbian tuning [SL01], Li presents a radial basis function neural network–based controller acting on a fighter aircraft [LSS01]. Chen and Narendra present a comparative study demonstrating the usefulness of a neural controller assisted by a linear control term [CN01]. A switching logic is designed and it is shown that neural network–based control scheme outperforms the pure linear and pure nonlinear versions of the feedback control law. Another work considering the multiple models activated via switching relaxes the condition of global boundedness of high-order nonlinear terms and improves the neurocontrol approach of Chen and Narendra [FC07]. An application of differential neural network–based control to nonlinear stochastic systems is discussed by Poznyak and Ljung [PL01]; nonlinear output regulation via recurrent neural networks is elaborated by [ZW01], and estimation of a Lyapunov function is performed for stable adaptive neurocontrol in [R01]. Predictive control approach has been another field of research that used the tools of neural networks framework. In [WW01], a particular neural structure is proposed, and this scheme is used to model predictive control. Yu and Gomm consider the multivariable model predictive control strategy on a chemical reactor model [YG03]. Applications of neurocontrol techniques in discrete-event automata is presented in [PS01], in reinforcement learning is reported in [SW01], and those in large-scale traffic network management is handled [CSC06]. Unmanned systems are another field that benefit from the neural network–based approaches. Due to the varying operating conditions and difficulty of constructing necessary process models, numerical data-based approaches become preferable as in the study reported in [K06], where a neural controller helps a proportional plus derivative controller and handles the time-dependent variations as its structure enables adaptation. This makes it sure that a certain set of performance criteria are met simultaneously. In [HCL07], a learning algorithm of proportional-integral-derivative type is proposed and a tracking control example is given in second-order chaotic system. Prokhorov suggests methods to train recurrent neural models for real-time applications [P07], and Papadimitropoulos et al. implement a fault-detection scheme based on an online approximation via neural networks [PRP07]. Two of the successful real-time results on spark ignition engines are

reported in [CCBC07,VSKJ07], where neural network models are used as internal model controller in [CCBC07] and as observer and controller in [VSKJ07]. In [YQLR07], tabu search is adapted for neural network training to overcome the problem of convergence to local minimum. Alanis et al. [ASL07] propose high-order neural networks used with backstepping control technique. Discrete time output regulation using neural networks is considered in [LH07], and use of neurocontrollers in robust Markov games is studied by [SG07]. Recently, radial basis function neural networks were employed for the control of a nonholonomic mobile robot in [BFC09] and fine-tuning issues in large-scale systems have been addressed in [KK09]. Another recent work by Ferrari reports an application of dynamic neural network–forced implicit model following on a tailfin-controlled missile problem [F09]. Direct adaptive optimal control via neural network tools is considered in [VL09], model predictive control for a steel pickling process is studied in [KTHD09], and issues in the sampled data adaptive control are elaborated in [P09].

In brief, in the 1980s, the first steps and models were introduced while the 1990s were the years of stipulating the full diversity of learning schemes, architectures, and variants. According to the cited volume of research, outcomes of this century are more focused on applications and integration to other modules of feedback systems. The approaches seem to incorporate the full power of computing facilities as well as small-size and versatile data acquisition hardware. Since the introduction of the McCulloch–Pitts neuron model, it can be claimed that parallel to the technological innovations in computing hardware, progress in neural network–based control systems seem to spread over wider fields of application. In what follows, the learning algorithms and architectural possibilities are discussed. The methods of neural network–based control are presented and an application example is given.

3.2 Learning Algorithms

A critically important component of neural network research is the way in which the parameters are tuned to meet a predefined set of performance criteria. Despite the presence of a number of learning algorithms, two of them have become standard methods in engineering applications, and are elaborated in the sequel.

3.2.1 Error Backpropagation

Consider the feedforward neural network structure shown in Figure 3.1. The structure is called feedforward as the flow of information has a one-way nature. In order to describe the parameter modification rule, a variable sub- and superscripting convention needs to be adopted. In Figure 3.2, the layers in a given network are labeled, and the layer number is contained as a superscript. The synaptic weight in between the node i in layer $k + 1$ and node j in layer k is denoted by w_{ij}^{k}. Let an input vector and output vector at time t_0 be defined as $I_0 = (u_1(t_0)u_2(t_0)\ldots u_m(t_0))$ and $O_0 = (y_1(t_0)y_2(t_0)\ldots y_n(t_0))$, respectively. An input–output pair, shortly a *pair* or sample, is defined as $S_0 = \{I_0, O_0\}$. Consider there are P pairs in a given data set, which we call training set. When the training set is given, one must interpret it as follows: When I_0 is presented to a neural network of appropriate dimensions, its response must be an

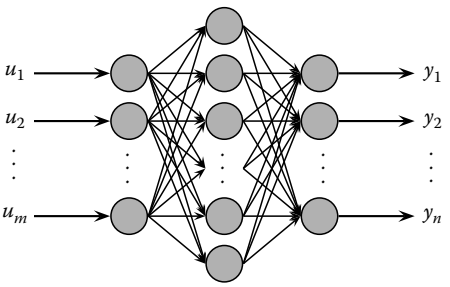

FIGURE 3.1 Structure of a feedforward neural network.

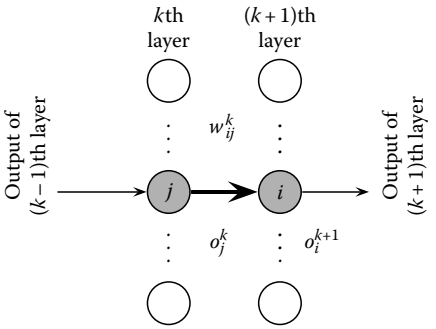

FIGURE 3.2 Sub- and superscripting of the variables in a feedforward neural network.

approximate of O_0. Based on this, the response of a neural network to a set of input vectors can be evaluated and the total cost over the given set can be defined as follows:

$$J(w) = \frac{1}{2P}\sum_{p=1}^{P}\sum_{i=1}^{n}\left(d_i^p - y_i^p(I_p, w)\right)^2 \tag{3.1}$$

where d_i^p is the target output corresponding to the ith output of the neural network that responds to pth pattern. The cost in (3.1) is also called the mean squared error (MSE) measure. Similarly, y_i^p is the response of the neural network to I_p. In (3.1), the generic symbol w stands for the set of all adjustable parameters, that is, the synaptic strengths, or weights.

Let the output of a neuron in the network be denoted by o_i^{k+1}. This neuron can belong to the output layer or a hidden layer of the network. The dependence of o_i^{k+1} to the adjustable parameters is as given below. Define $S_i^{k+1} := \sum_{j=1}^{n_k} w_{ij}^k o_j^k$ as the net sum determining the activation level of the neuron:

$$o_i^{k+1} = f\left(S_i^{k+1}\right) \tag{3.2}$$

where

$f(\cdot)$ is the neuronal activation function
n_k is the number of neurons in the kth layer

Gradient descent prescribes the following parameter update rule:

$$w_{ij}^k(t+1) = w_{ij}^k(t) - \eta\frac{\partial J(w)}{\partial w_{ij}^k(t)} \tag{3.3}$$

where

η is the learning rate chosen to satisfy $0 < \eta < 1$
the index t emphasizes the iterative nature of the scheme

Defining $\Delta w_{ij}^k(t) = w_{ij}^k(t+1) - w_{ij}^k(t)$, one could reformulate the above law as

$$\Delta w_{ij}^k(t) = -\eta\frac{\partial J(w)}{\partial w_{ij}^k(t)} \tag{3.4}$$

which is known also as the MIT rule, steepest descent, or the gradient descent, all referring to the above modification scheme.

3.2.1.1 Adaptation Law for the Output Layer Weights

Let $(k+1)$th layer be the output layer. This means $y_i = o_i^{k+1}$, and the target value for this output is specified explicitly by d_i. For the pth pair, define the output error at the ith output as

$$e_i^p = d_i^p - y_i^p(I_p, w) \tag{3.5}$$

After evaluating the partial derivative in (3.4), the updating of the output layer weights is performed according to the formula given in (3.6), where the pattern index is dropped for simplicity.

$$\Delta w_{ij}^k = \eta \delta_i^{k+1} o_j^k \tag{3.6}$$

where δ_i^{k+1} for the output layer is defined as follows:

$$\delta_i^{k+1} = e_i f'(S_i^{k+1}) \tag{3.7}$$

where $f'(S_i^{k+1}) = \partial f / \partial S_i^{k+1}$. One could check the above rule from Figure 3.2 to see the dependencies among the variables involved.

3.2.1.2 Adaptation Law for the Hidden Layer Weights

Though not as straightforward to show as that for the output layer, the weights in the hidden layer need an extra step as there are many paths through which the output errors can be backpropagated. In other words, according to Figures 3.1 and 3.2, it is clear that a small perturbation in w_{ij}^k will change the entire set of network outputs causing a change in the value of J. Now we consider $(k+1)$th layer as the hidden layer (see Figure 3.3). The general expression given by (3.6) is still valid and after appropriate manipulations, we have

$$\delta_i^{k+1} = \left(-\sum_{h=1}^{n_{k+2}} \delta_h^{k+2} w_{hi}^{k+1} \right) f'(S_i^{k+1}) \tag{3.8}$$

It is straightforward to see that the approach presented is still applicable if there are more than one hidden layers. In such cases, output errors are backpropagated until the necessary values shown in (3.8) are obtained.

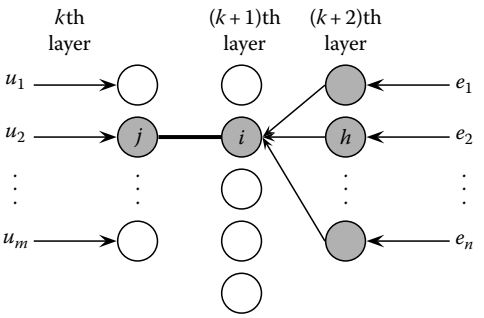

FIGURE 3.3 Influence of the output errors on a specific hidden layer weight.

Few modifications to the update law in (3.6) were proposed to speed up the convergence. Error backpropagation scheme starts fast, yet as time passes, it gradually slows down particularly in the regions where the gradient is small in magnitude. Adding a momentum term or modifying the learning rate are the standard approaches whose improving effect were shown empirically [H94].

3.2.2 Second-Order Methods

Error backpropagation was used successfully in many applications; however, its very slow convergence speed has been the major drawback highlighted many times. Being the major source of motivation, increasing the convergence speed has been achieved for feedforward neural networks in 1994 [HM94]. Hagan and Menhaj applied the Marquardt algorithm to update the synaptic weights of a feedforward neural network structure. The essence of the algorithm is as follows.

Consider a neural network having n outputs, and N adjustable parameters denoted by the vector $w = (w_1\ w_2\ \ldots\ w_N)$. Each entry of the parameter vector corresponds to a unique parameter in an ordered fashion. If there are P pairs over which the interpolation is to be performed, the cost function qualifying the performance is as given in (3.1) and the update law is given in (3.9).

$$w_{t+1} = w_t - \left(\nabla_w^2 J(w_t)\right)^{-1} \nabla_w J(w_t) \tag{3.9}$$

where t stands for the discrete time index. Here, $\nabla_w^2 J(w_t) = 2H(w_t)^{\mathrm{T}} H(w_t) + g(H(w_t))$ with $g(H(w_t))$ being a small residual, and $\nabla_w J(w_t) = 2H(w_t)^{\mathrm{T}} E(w_t)$ with E and H being the error vector and the Jacobian, as given in (3.10) and (3.11), respectively. The error vector contains the errors computed by (3.5) for every training pair, and the Jacobian contains the partial derivatives of each component of E with respect to every parameter in w.

$$E = \begin{pmatrix} e_1^1 & \cdots & e_n^1 & e_1^2 & \cdots & e_n^2 & \cdots & e_1^P & \cdots & e_n^P \end{pmatrix}^{\mathrm{T}} \tag{3.10}$$

$$H(w) = \begin{pmatrix} \frac{\partial e_1^1(w)}{\partial \omega_1} & \frac{\partial e_1^1(w)}{\partial \omega_2} & \cdots & \frac{\partial e_1^1(w)}{\partial \omega_N} \\ \frac{\partial e_2^1(w)}{\partial \omega_1} & \frac{\partial e_2^1(w)}{\partial \omega_2} & \cdots & \frac{\partial e_2^1(w)}{\partial \omega_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_n^1(w)}{\partial \omega_1} & \frac{\partial e_n^1(w)}{\partial \omega_2} & \cdots & \frac{\partial e_n^1(w)}{\partial \omega_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_1^P(w)}{\partial \omega_1} & \frac{\partial e_1^P(w)}{\partial \omega_2} & \cdots & \frac{\partial e_1^P(w)}{\partial \omega_N} \\ \frac{\partial e_2^P(w)}{\partial \omega_1} & \frac{\partial e_2^P(w)}{\partial \omega_2} & \cdots & \frac{\partial e_2^P(w)}{\partial \omega_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_n^P(w)}{\partial \omega_1} & \frac{\partial e_n^P(w)}{\partial \omega_2} & \cdots & \frac{\partial e_n^P(w)}{\partial \omega_N} \end{pmatrix} \tag{3.11}$$

Based on these definitions, the well-known Gauss–Newton algorithm can be given as

$$w_{t+1} = w_t - \left(H(w_t)^{\mathrm{T}} H(w_t)\right)^{-1} H(w_t)^{\mathrm{T}} E(w_t) \tag{3.12}$$

and the Levenberg–Marquardt update can be constructed as

$$w_{t+1} = w_t - \left(\mu I + H(w_t)^{\mathrm{T}} H(w_t)\right)^{-1} H(w_t)^{\mathrm{T}} E(w_t) \tag{3.13}$$

where

$\mu > 0$ is a user-defined scalar design parameter for improving the rank deficiency problem of the matrix $H(w_k)^{\mathrm{T}} H(w_k)$

I is an identity matrix of dimensions $N \times N$

It is important to note that for small μ, (3.13) approximates to the standard Gauss–Newton method (see (3.12)), and for large μ, the tuning law becomes the standard error backpropagation algorithm with a step size $\eta \approx 1/\mu$. Therefore, Levenberg–Marquardt method establishes a good balance between error backpropagation and Gauss–Newton strategies and inherits the prominent features of both algorithms in eliminating the rank deficiency problem with improved convergence. Despite such remarkably good properties, the algorithm in (3.13) requires the inversion of a matrix of dimensions $nP \times N$ indicating high computational intensity. Other variants of second-order methods differ in some nuances yet in essence, they implement the same philosophy. Conjugate gradient method with Polak–Ribiere and Fletcher–Reeves formulas are some variants used in the literature [H94]. Nevertheless, the problem of getting trapped to local minima continues to exist, and the design of novel adaptation laws is an active research topic within the realm of neural networks.

3.2.3 Other Alternatives

Typical complaints in the application domain have been to observe very slow convergence, convergence to suboptimal solutions rendering the network incapable of performing the desired task, oversensitivity to suddenly changing inputs due to the gradient computation, and the like. Persisting nature of such difficulties has led the researchers to develop alternative tuning laws alleviating some of these drawbacks or introducing some positive qualities. Methods inspired from variable structure control are one such class showing that the error can be cast into a phase space and guided toward the origin while displaying the robustness properties of the underlying technique [YEK02,PMB98]. Another is based on derivative-free adaptation utilizing the genetic algorithms [SG00]. Such algorithms refine the weight vector based on the maximization of a fitness function. In spite of their computational burden, methods that do not utilize the gradient information are robust against the sudden changes in the inputs. Aside from these, unsupervised learning methods constitute another alternative used in the literature. Among a number of alternatives, reinforcement learning is one remarkable approach employing a reward and penalty scheme to achieve a particular goal. The process of reward and penalty is an evaluative feedback that characterizes how the weights of a neural structure should be modified [SW01,MD05,N97].

3.3 Architectural Varieties

Another source of diversity in neural network applications is the architecture. We will present the alternatives under two categories: first is the type of connectivity, second is the scheme adopted for neuronal activation. In both cases, numerous alternatives are available as summarized below.

3.3.1 Structure

Several types of neural network models are used frequently in the literature. Different structural configurations are distinguished in terms of the data flow properties of a network, that is, feedforward models and recurrent models. In Figure 3.1, a typical feedforward neural network is shown. Three simple versions of recurrent connectivity are illustrated in Figure 3.4. The network in Figure 3.4a has recurrent neurons in the hidden layer, Figure 3.4b feeds back the network output and established recurrence externally, and Figure 3.4c contains fully recurrent hidden layer neurons.

Clearly, one can set models having multiple outputs as well as feedback connections in between different layers but due to the space limit, we omit those cases and simply consider the function of a neuron having feedback from other sources of information as in Equation 3.2 but now we have the net sum as follows:

$$S_i^{k+1} = \sum_{j=1}^{n_k} w_{ij}^k o_j^k + \sum_{i=1}^{R} \rho_i \xi_i \tag{3.14}$$

where

the second sum is over all feedback connections

ρ_i denotes the weight determining the contribution of the output ξ_i from a neuron in a layer

Clearly, the proper application of error backpropagation or Levenberg–Marquardt algorithm is dependent upon the proper handling of the network description under feedback paths [H94].

3.3.2 Neuronal Activation Scheme

The efforts toward obtaining the best performing artificial neural model has also focused on the neuronal activation scheme. The map built by a neural model is strictly dependent upon the activation function. Smooth activation schemes produce smooth hypersurfaces while sharp ones like the sign function produce hypersurfaces having very steep regions. This subsection focuses on the dependence of performance on the type of neuronal activation function. In the past, this was considered to some extent in [KA02,HN94,WZD97,CF92,E08]. Efe considers eight different data sets, eight different activation functions with networks having 14 different neuron numbers in the hidden layer. Under these conditions, the research conducted gives a clear idea about when an activation function is advisable. The goal of such a research is to figure out what type of activation function performs well if the size of a network is small [E08]. Some of the neuronal activation functions mentioned frequently in the literature are

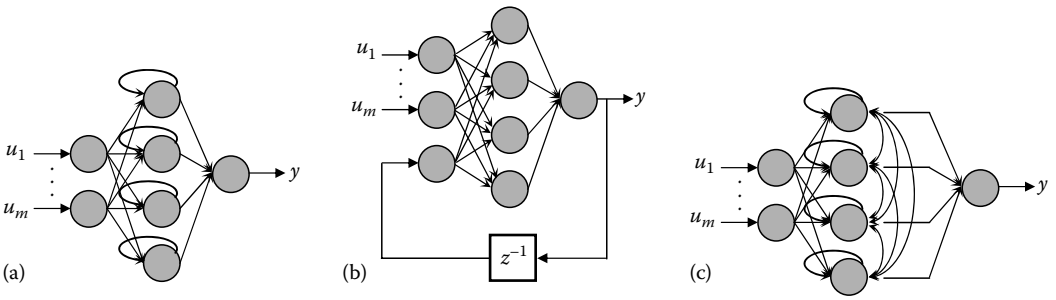

FIGURE 3.4 Three alternatives for feedback-type neural networks having single output.

TABLE 3.2 Neuronal Activation Functions

Name	Activation Function	Adjustables
Hyperbolic tangent	$f(x) = \tanh(x)$	—
Damped polynomial	$f(x) = (ax^2 + bx + c)\exp(-\lambda x^2)$	a, b, c, λ
Multilevel	$f(x) = \frac{1}{2M+1}\sum_{k=-M}^{M} \tanh(x - \lambda k)$	M, λ
Trigonometric	$f(x) = a\sin(px) + b\cos(qx)$	a, b, p, q
Arctangent	$f(x) = \operatorname{atan}(x)$	—
Sinc	$f(x) = \begin{cases} \frac{\sin(\pi x)}{\pi x} & x \neq 0 \\ 1 & x = 0 \end{cases}$	—
Logarithmic	$f(x) = \begin{cases} \ln(1+x) & x \geq 0 \\ -\ln(1-x) & x < 0 \end{cases}$	—

tabulated in Table 3.2, which shows the diversity of options in setting up a neural network, and choosing the best activation scheme is still an active research involving the problem in hand as well.

3.4 Neural Networks for Identification and Control

3.4.1 Generating the Training Data

A critically important component in developing a map from one domain to another is dependent upon the descriptive nature of the entity that lies between these domains and that describes the map implicitly. This entity is the numerical data, or the raw data to be used to build the desired mapping. To be more explicit, for a two-input single-output mapping, it may be difficult to see the global picture based on the few samples shown in Figure 3.5a, yet it is slightly more visible from Figure 3.5b and it is more or less a plane according to the picture in Figure 3.5c. Indeed, the sample points in all three points were generated using $y = 3u_1 + 4u_2$. This simple experiment shows us that in order to describe the general picture, the data must be distributed well around the questioned domain as well as it must be dense enough to deduce the general behavior.

In the applications reporting solutions to synthetic problems, the designer is free to generate as much data as he or she needs; however, in real-time problems, collecting data to train a neural network may be a tedious task, or even sometimes a costly one. Furthermore, for problems that have more than two inputs, utilizing the graphical approaches may have very limited usefulness. This discussion amounts to saying that if there are reasonably large number of training data describing a given process, a neural network–based model is a good alternative to realized the desired map.

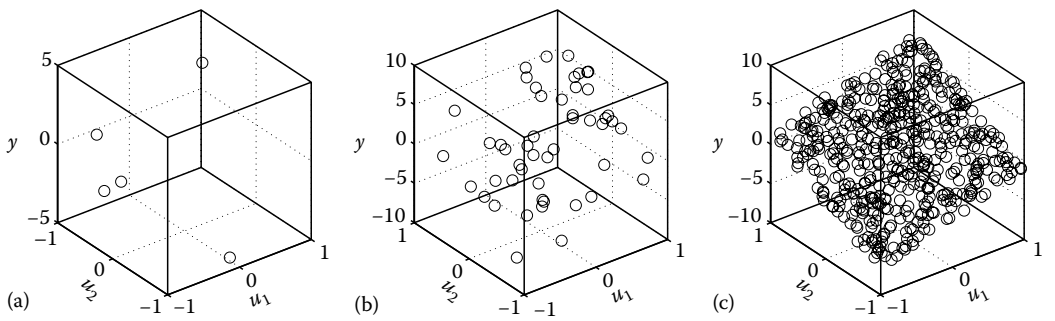

FIGURE 3.5 Changing implication of some amount of data.

3.4.2 Determining the Necessary Inputs: Information Sufficiency

Previous discussion can be extended to the concept of information sufficiency. For the plane given above, a neural model would need u_1 and u_2 as its inputs. Two questions can be raised:

- Would the neural model be able to give acceptable results if it was presented only one of the inputs, for example, u_1 or u_2?
- Would the neural model perform better if it was presented another input containing some linear combination of u_1 and u_2, say for example, $u_3 = \alpha_1 u_1 + \alpha_2 u_2$, $(\alpha_1, \alpha_2 \neq 0)$?

The answer to both questions is obviously no, however, in real-life problems, there may be a few hundred variables having different amounts of effect on an observed output, and some—though involved in the process—may have negligibly small effect on the variables under investigation. Such cases must be clearly analyzed, and this has been a branch of neural network–based control research studied in the past [LP07].

3.4.3 Generalization or Memorization

Noise is an inevitable component of real-time data. Consider a map described implicitly by the data shown in Figure 3.6. The circles in the figure indicate the data points given to extract the shown target curve. The data given in the first case, the left subplot, does not contain noise and a neural network that memorizes, or overfits, the given data points produce a network that well approximates the function. Memorization in this context can formally be defined as $J = 0$, and this can be a goal if it is known that there is no noise in the data. If the designer knows that the data is noisy as in the case shown in the middle subplot, a neural network that overfits the data will produce a curve, which is dissimilar from the target one. Pursuing memorization for the data shown in the right subplot will produce a neural model that performs even worse. This discussion shows that memorization, or overfitting, for real life and possibly noisy data is likely to produce poorly performing neural models; nevertheless, noise is the major actor determining the minimum possible value of J for a given set of initial conditions and network architecture.

In the noisy cases of Figure 3.6, a neural network will start from an arbitrary curve. As training progresses, the curve realized by the neural model will approach the target curve. After a particular instant, the neural model will produce a curve that passes through the data points but is not similar to the target curve. This particular instant is determined by utilizing two sets during the training phase. One is used for synaptic modification while the other is used solely for checking the cost function. If the cost functions computed over both sets decrease, the network is said to generalize the given map. If the cost for training set continues to decrease while that for the test set starts increasing, the neural network is said to start overfitting the given data, and this instant is the best instant to stop the training.

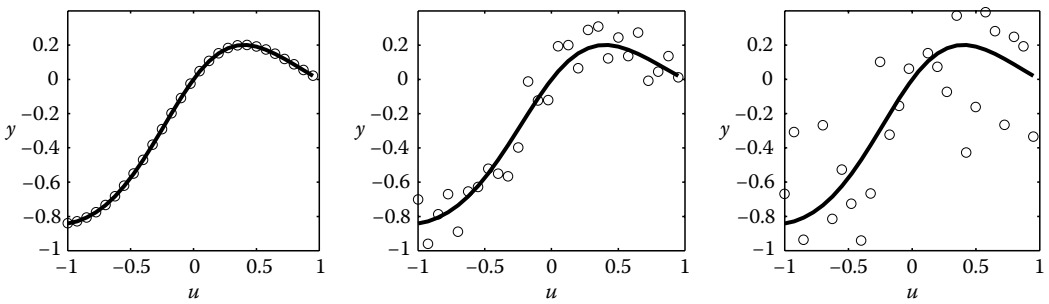

FIGURE 3.6 Effect of noise variance on the general perception.

3.4.4 Online or Offline Synthesis

A neural network–based controller can be adaptive or nonadaptive depending on the needs of the problem in hand and the choice of the user. Nonadaptive neural network controller is trained prior to installation and its parameters are frozen. This approach is called offline synthesis (or training). The other alternative considers a neurocontroller that may or may not be pretrained but its weights are refined during the operation. This scheme requires the extraction of an error measure to set up the update laws. The latter approach is more popular as it handles the time variation in the dynamical properties of a system and maintains the performance by altering the controller parameters appropriately. This scheme is called online learning. A major difference between these methods is the number of training data available at any instant of time. In the offline synthesis, the entire set of data is available, and there is no time constraint to accomplish the training, whereas in the online training, data comes at every sampling instant and its processing for training must be completed within one sampling period, and this typically imposes hardware constraints into the design.

3.5 Neurocontrol Architectures

Consider the synthetic process governed by the difference equation $x_{t+1} = f(x_t, u_t)$, where x_t is the value of state at discrete time t and u_t is the external input. Assume the functional details embodying the process dynamics are not available to follow conventional design techniques, and assume it is possible to run the system with arbitrary inputs and arbitrary initial state values. The problem is to design a system that observes the state of the process and outputs u_t, by which the system state follows a given reference r_t. A neural network–based solution to this problem prescribes the following steps.

- Initialize the state x_t to a randomly selected value satisfying $x_t \in \mathcal{X} \in \Re$, where $\mathcal{X}$ is the interval we are interested in.
- Choose a stimulus satisfying $u_t \in \mathcal{U} \in \Re$, where $\mathcal{U}$ stands for the interval containing likely control inputs.
- Measure/evaluate the value of x_{t+1}.
- Form an input vector $I_t = (x_{t+1}\ x_t)$ and an output $O_t = u_t$ and repeat these four steps P times to obtain a training data set.
- Perform training to minimize the cost in Equation 3.1 and stop training when a stopping criterion is met.

A neural network realizing the map $u_t = \mathrm{NN}(x_{t+1}, x_t)$ would answer the question "What would be the value of the control signal if a transition from x_{t+1} to x_t is desired?" Clearly, having obtained a properly trained neural controller, one would set $x_{t+1} = r_{t+1}$ and the neurocontroller would drive the system state to the reference signal as it had been trained to do so.

If $x_t \in \Re^n$, then we have $r_t \in \Re^n$ and naturally $I_t \in \Re^{2n}$. Similarly, multiple inputs in a dynamic system would require a neurocontroller to have the same number of inputs as the plant contains and the same approach would be valid. A practical difficulty in training the controller utilizing the direct synthesis approach is that the designer decides on the excitation first, that is, the set $\mathcal{U}$, however, it can be practically difficult to foresee the interval to which a control signal in real operation belongs. This scheme can be called direct inversion. In the rest of this section, common structures of neurocontrol are summarized.

In Figure 3.7, identification of a dynamic system is shown. The trainer adjusts the neural network identifier (shown as NN-I) parameters in such a way that the cost given in (3.1) is minimized over the given set of training patterns. The identification scheme depicted in Figure 3.7, can also be utilized to identify an existing controller acting in a feedback control system. This approach corresponds to the mimicking of a conventional controller [N97].

In Figure 3.8, indirect learning architecture is shown. An input signal u is applied to the plant, it responds to the signal, which is denoted by y. A neural network controller, (NN-C in the figure) shown

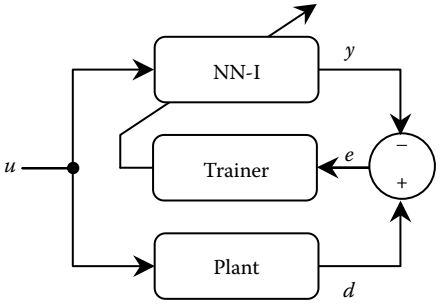

FIGURE 3.7 NN-based identifier training architecture.

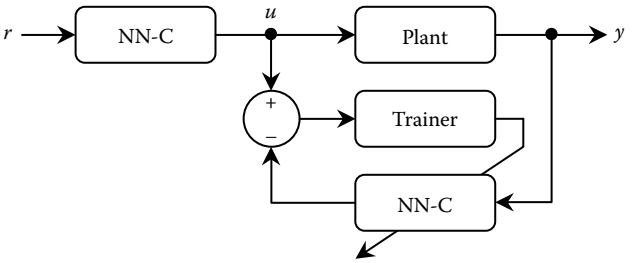

FIGURE 3.8 Indirect learning architecture.

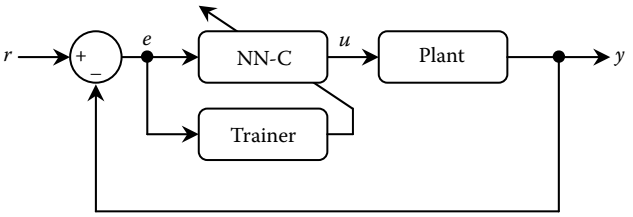

FIGURE 3.9 Closed-loop direct inverse control.

at the bottom, receives the response of the plant and tries to reconstruct the input u based on this observation. In the same time, the modified neural network is copied in front of the plant, which inverts the plant dynamics as time passes [PSY88,N97].

In Figure 3.9, closed-loop direct inverse control is depicted. The neural network acts as the feedback controller, and it receives the tracking error as the input. A trainer modifies the neural network parameters to force the plant response to what the command signal r prescribes. The tracking error is used directly as a measure of the error caused by the controller, and the scheme is called direct inversion in closed loop.

Open-loop version of the inversion method shown in Figure 3.9 is illustrated in Figure 3.10, which is also called the specialized learning architecture in the literature [PSY88]. According to the shown connectivity, the neural inverter receives the command signal (r) and outputs a control signal (u), in response to which the plant produces a response denoted by y and the error defined by $r - y$ is used to tune the parameters of the neural controller.

As highlighted by Narendra and Parthasarathy, indirect adaptive control using neural networks is done as shown in Figure 3.11. An identifier develops and refines the forward model of the plant, and a controller receives the equivalent value of the output error after passing through the neural model utilizing backpropagation technique. Such a method extracts a better error measure to penalize the

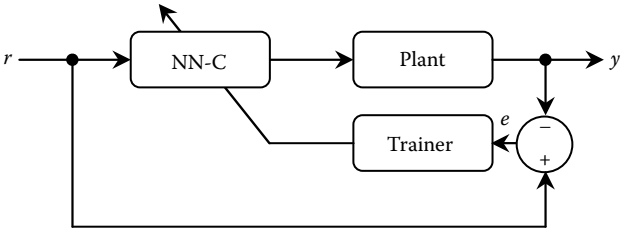

FIGURE 3.10 Specialized learning architecture.

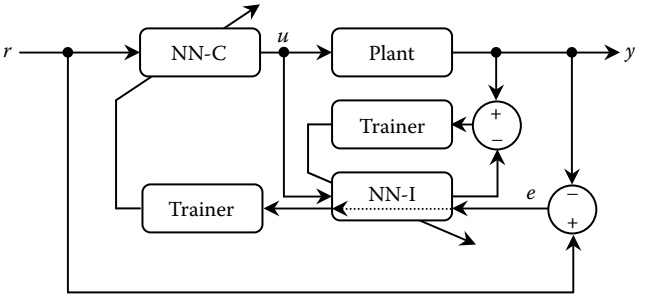

FIGURE 3.11 Indirect adaptive control scheme.

controller parameters. Since the controller tuning needs an extra propagation of information via the neural identifier, the scheme is called indirect adaptive control [NP91], or forward and inverse modeling [N97].

In many applications of neurocontrol, the plant, when discretized has the form $y_{t+1} = f(y_t, y_{t-1}, \ldots, y_{t-L_1}) + g(y_t, y_{t-1}, \ldots, y_{t-L_2})u_t$. Such models can enjoy the feedback linearization technique and when some set of consecutively collected numerical observations are available, one can proceed to develop the functions $f(\cdot)$ and $g(\cdot)$ separately, as shown in Figure 3.12. A classical control scheme supplied by the estimates of $f(\cdot)$ and $g(\cdot)$ can drive the system output to a given command signal adaptively.

Feedback error learning architecture proposed by [KFS87] is given in Figure 3.13, where a conventional controller is placed to stabilize the plant. It is assumed that the stabilizing effect provided

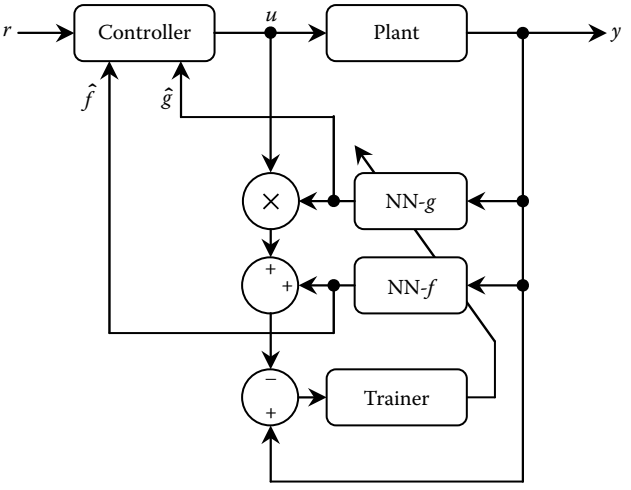

FIGURE 3.12 Feedback linearization via neural networks.

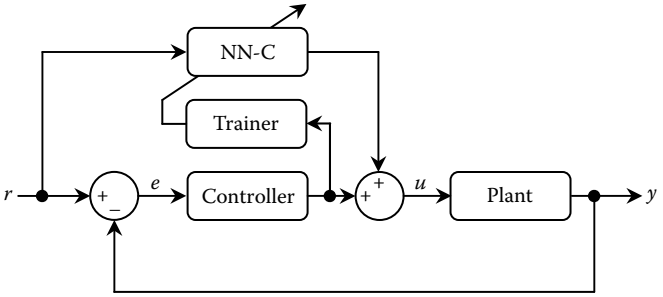

FIGURE 3.13 Feedback error learning architecture.

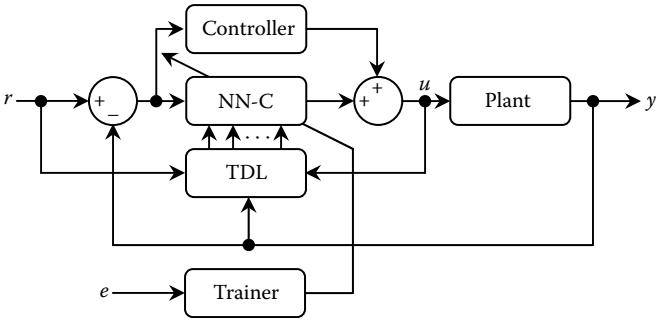

FIGURE 3.14 A typical neural network–based control architecture.

by the conventional controller may not be perfect and the neural controller provides a corrective control by acting in the feedforward path.

The typical diagrammatic view of neurocontrol is as illustrated in Figure 3.14, where there may be two controllers operated simultaneously and tapped delay line (TDL) block provides some history of relevant variables. A conventional controller—if used—maintains the stability of the closed loop and a neurocontroller complements it. Depending on the design and expectations, one of these controllers play the primary role while the other carries the auxiliary role. In every application example, a trainer should be provided an error measure quantifying the distance between the current output and the target value of it.

3.6 Application Examples

3.6.1 Propulsion Actuation Model for a Brushless DC Motor–Propeller Pair

The dynamical model of an unmanned aerial vehicle (UAV) like the one shown in Figure 3.15 could be obtained using the laws of physics. Principally, a control signal to be applied to the motors must be converted to pulse width modulation (pwm) signals, then electronic speed controllers properly drive the brushless motors, and a thrust value is obtained from each motor–propeller pair. The numerical value of the thrust is dependent upon the type of the propeller, and the angular speed of the rotor in radians is $f_i = b\Omega_i^2$ with f_i is the thrust at ith motor, b is a constant-valued thrust coefficient, and Ω_i is the angular speed in rad/s. If the control inputs (thrusts) needed to observe a desired motion were immediately available, then it would be easier to proceed to the closed-loop control system design without worrying about the effects of the actuation periphery, which introduces some constraints shaping the transient and steady-state behavior of the propulsion. Indeed, the real-time picture is much more complicated as the vehicle is an electrically powered one and battery voltage is reducing. Such a change in the battery

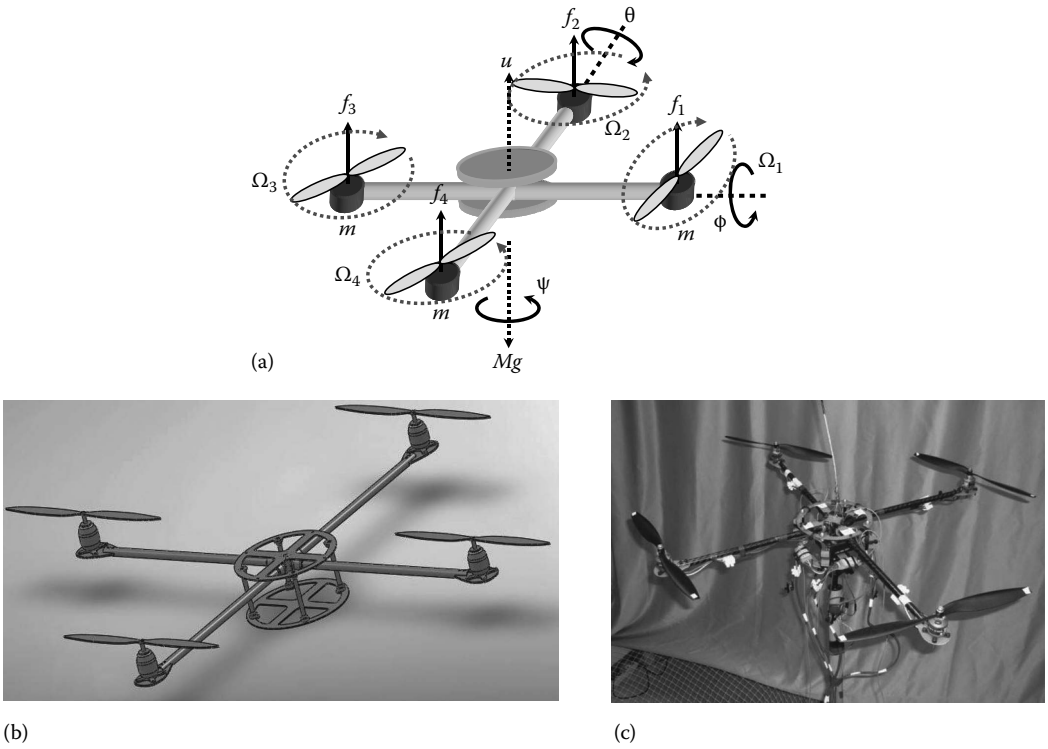

FIGURE 3.15 (a) Schematic view and variable definitions of a quadrotor-type UAV, (b) CAD drawing, and (c) real implementation.

voltage causes different lift forces at different battery voltage levels although the applied pwm level is constant, as seen in Figure 3.16. Same pwm profile is applied repetitively and as the battery voltage reduces, the angular speed at a constant pwm level decreases thereby causing a decrease in the generated thrust. Furthermore, the relation with different pwm levels is not linear, that is, same amount of change in the input causes different amounts of change at different levels, and this shows that the process to be modeled is a nonlinear one.

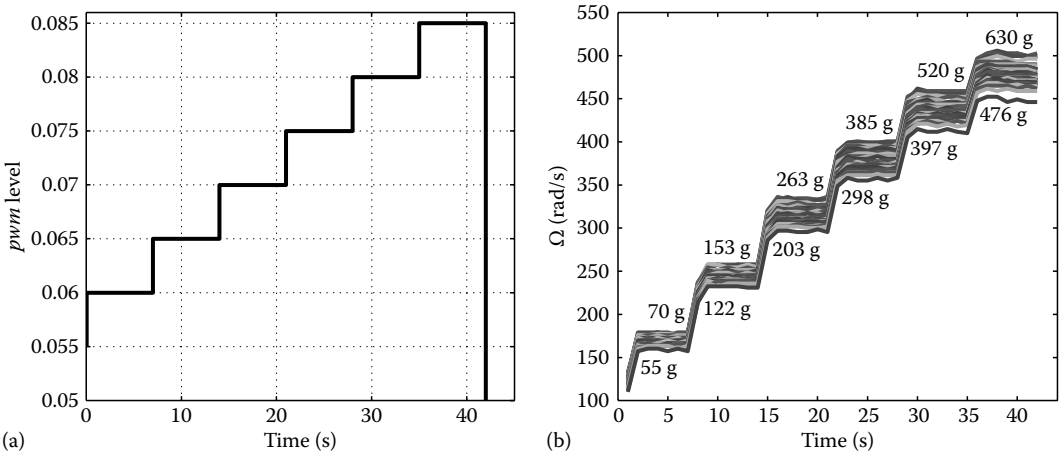

FIGURE 3.16 (a) Applied pwm profile and (b) decrease in the angular speed as the battery voltage decreases.

According to Figure 3.16, comparing the fully charged condition of the battery and the condition at the last experiment displays 15 g of difference for the lowest level, 154 g at the highest level, which is obviously an uncertainty that has to be incorporated into the dynamic model and a feedback controller appropriately. Use of neural networks is a practical alternative to resolve the problem induced by battery conditions. Denoting $V_b(t)$ as the battery voltage, a neural network model performing the map $y_{pwm} = \mathrm{NN}(\Omega_c, V_b)$ is the module installed to the output of a controller generating the necessary angular speeds. Here Ω_c is the angular speed prescribed by the controller. Another neural network that implements $y_\Omega = \mathrm{NN}(V_b, pwm, \sigma_1(pwm))$ is the module installed to the inputs of the dynamic model of the UAV. The function $\sigma_1(\cdot)$ is a low-pass filter incorporating the effect of transient in the thrust value. The dynamic model contains f_is that are computed using Ω_is.

The reason why we would like to step down from thrusts to the pwm level and step up from pwm level to forces is the fact that brushless DC motors are driven at the pwm level and one has to separate the dynamic model of the UAV and the controller by drawing a line exactly at the point of signal exchange occurring at the pwm level. Use of neural networks facilitates this in the presence of voltage loss in the batteries.

In Figure 3.17, the diagram describing the role of aforementioned offline trained neural models are shown. In Figure 3.18, the results obtained with real-time data are shown. A chirp-like pwm profile was generated and some noise added to obtain a pwm signal to be applied. When this signal is applied as an input to any motor, the variation in the battery voltage is measured and filtered to guide

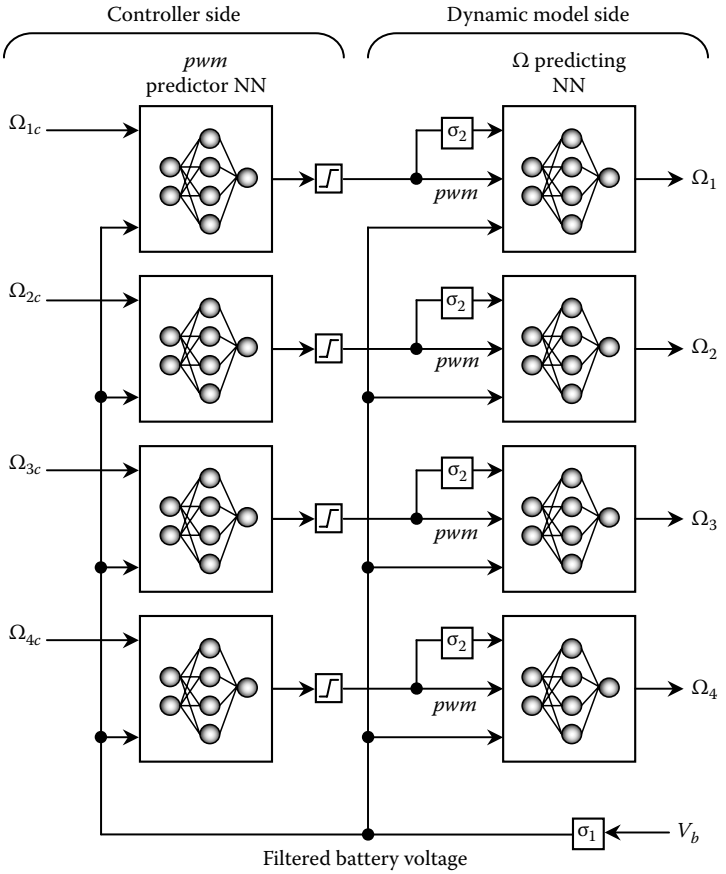

FIGURE 3.17 Installing the neural network components for handshaking at pwm level.

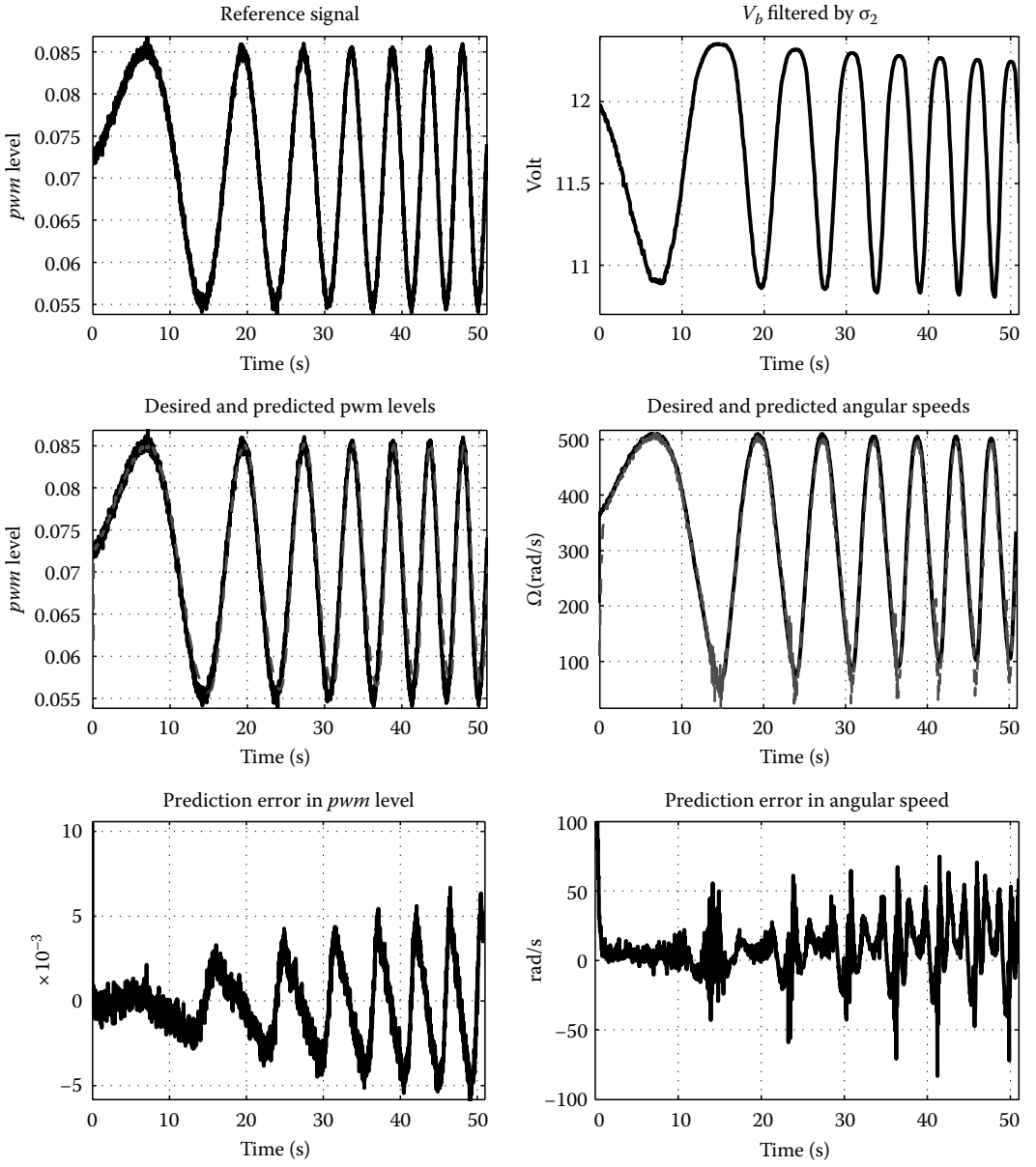

FIGURE 3.18 Performance of the two neural network models.

the neural models, as shown in the top right subplot. After that, the corresponding angular speed is computed experimentally. In the middle left subplot, the reconstructed pwm signal and the applied signal are shown together whereas the middle right subplot depicts the performance for the angular speed predicting neural model. Both subplots suggest a useful reconstruction of the signal asked from the neural networks that were trained by using Levenberg–Marquardt algorithm. In both models, the neural networks have single hidden layer with hyperbolic tangent-type neuronal nonlinearity and linear output neurons. The pwm predicting model has 12 hidden neurons with $J = 90.8285 \times 10^{-4}$ as the final cost, angular speed predicting neural model has 10 hidden neurons with $J = 3.9208 \times 10^{-4}$ as the final cost value.

Bottom subplots of Figure 3.18 illustrate the difference between the desired and predicted values. As the local frequency of the target output increases, the neural models start performing poorer yet the performance is good when the signals change slowly. This is an expected result that is in good compliance with the typical real-time signals obtained from the UAV in Figure 3.15.

3.6.2 Neural Network–Aided Control of a Quadrotor-Type UAV

The dynamic model of the quadrotor system shown in Figure 3.15 can be derived by following the Lagrange formalism. The model governing the dynamics of the system is given in (3.15) through (3.20), where the first three ODEs describe the motion in Cartesian space, the last three define the dynamics determining the attitude of the vehicle. The parameters of the dynamic model are given in Table 3.3.

$$\ddot{x} = (\cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi)\frac{1}{M}U_1 \tag{3.15}$$

$$\ddot{y} = (\cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi)\frac{1}{M}U_1 \tag{3.16}$$

$$\ddot{z} = -g + \cos\phi\cos\theta\frac{1}{M}U_1 \tag{3.17}$$

$$\ddot{\phi} = \dot{\theta}\dot{\psi}\left(\frac{I_{yy} - I_{zz}}{I_{xx}}\right) + \frac{j_r}{I_{xx}}\dot{\theta}\omega + \frac{L}{I_{xx}}U_2 \tag{3.18}$$

$$\ddot{\theta} = \dot{\phi}\dot{\psi}\left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right) - \frac{j_r}{I_{yy}}\dot{\phi}\omega + \frac{L}{I_{yy}}U_3 \tag{3.19}$$

$$\ddot{\psi} = \dot{\theta}\dot{\phi}\left(\frac{I_{xx} - I_{yy}}{I_{zz}}\right) + \frac{1}{I_{zz}}U_4 \tag{3.20}$$

where

$$\omega = \Omega_1 - \Omega_2 + \Omega_3 - \Omega_4 \tag{3.21}$$

$$U_1 = b\Omega_1^2 + b\Omega_2^2 + b\Omega_3^2 + b\Omega_4^2 = f_1 + f_2 + f_3 + f_4 \tag{3.22}$$

TABLE 3.3 Physical Parameters of the Quadrotor UAV

L	Half distance between two motors on the same axis	0.3 m
M	Mass of the vehicle	0.8 kg
g	Gravitational acceleration constant	9.81 m/s^2
I_{xx}	Moment of inertia around x-axis	15.67e−3
I_{yy}	Moment of inertia around y-axis	15.67e−3
I_{zz}	Moment of inertia around z-axis	28.346e−3
b	Thrust coefficient	192.3208e−7 N s^2
d	Drag coefficient	4.003e−7 N m s^2
j_r	Propeller inertia coefficient	6.01e−5

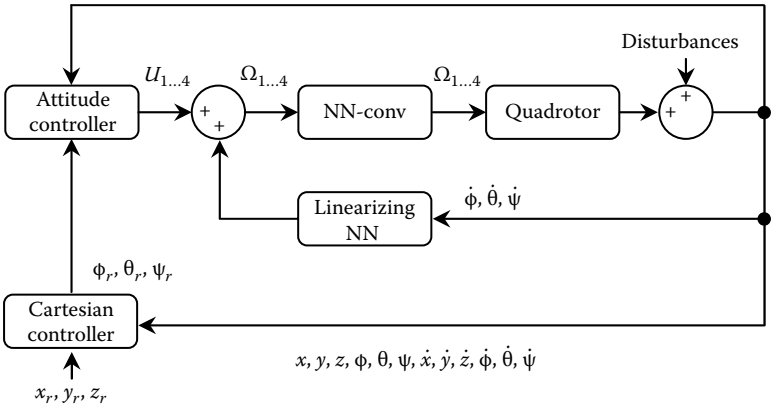

FIGURE 3.19 Block diagram of the overall control system.

$$U_2 = b\Omega_4^2 - b\Omega_2^2 = f_4 - f_2 \tag{3.23}$$

$$U_3 = b\Omega_3^2 - b\Omega_1^2 = f_3 - f_1 \tag{3.24}$$

$$U_4 = d(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2) \tag{3.25}$$

The control problem here is to drive the UAV toward a predefined trajectory in the 3D space by generating an appropriate sequence of Euler angles, which need to be controlled as well. The block diagram of the scheme is shown in Figure 3.19, where the attitude control is performed via feedback linearization using a neural network, and outer loops are established based on classical control theory.

The role of the linearizing neural network is to observe the rates of the roll ($\dot{\phi}$), pitch ($\dot{\theta}$), yaw ($\dot{\psi}$) angles, and the ω parameter, and to provide a prediction for the function given as follows:

$$\mathrm{NN} \approx F = (\dot{\phi}, \dot{\theta}, \dot{\psi}, \omega) = \begin{pmatrix} \dot{\theta}\dot{\psi}\left(\dfrac{I_{yy} - I_{zz}}{I_{xx}}\right) + \dfrac{j_r}{I_{xx}}\dot{\theta}\omega \\ \dot{\phi}\dot{\psi}\left(\dfrac{I_{zz} - I_{xx}}{I_{yy}}\right) - \dfrac{j_r}{I_{yy}}\dot{\phi}\omega \\ \dot{\theta}\dot{\phi}\left(\dfrac{I_{xx} - I_{yy}}{I_{zz}}\right) \end{pmatrix} \tag{3.26}$$

The observations are noisy and based on the dynamic model, a total of 2000 pairs of training data and 200 validation data are generated to train the neural network. Levenberg–Marquardt training scheme is used to update the network parameters and the training was stopped after 10,000 iterations, the final MSE cost is J = 3.3265e−7, which was found acceptable. In order to incorporate the effects of noise, the training data were corrupted 5% of the measurement magnitude to maintain a good level of generality. The neural network realizing the vector function given in (3.26) has two hidden layers employing hyperbolic tangent-type neuronal activation functions, the output layer is chosen to be a linear one. The first hidden layer has 12, and the second hidden layer has 6 neurons. With the help of the neural network, the following attitude controller has been realized:

$$U_2 = \frac{I_{xx}}{L}\left(K_\phi e_\phi + 2\sqrt{K_\phi}\,\dot{e}_\phi + \ddot{\phi}_r - \mathrm{NN}_1\right) \tag{3.27}$$

$$U_2 = \frac{I_{xx}}{L}\left(K_\theta e_\theta + 2\sqrt{K_\theta}\,\dot{e}_\theta + \ddot{\theta}_r - \mathrm{NN}_2\right) \tag{3.28}$$

$$U_4 = I_{zz}\left(K_\psi e_\psi + 2\sqrt{K_\psi}\,\dot{e}_\psi + \ddot{\psi}_r - \mathrm{NN}_3\right) \tag{3.29}$$

where

$e_\phi = \phi_r - \phi$
$e_\theta = \theta_r - \theta$
$e_\psi = \psi_r - \psi$
a variable with subscript r denotes a reference signal for the relevant variable

Since the motion in Cartesian space is realized by appropriately driving the Euler angles to their desired values, the Cartesian controller produces the necessary Euler angles for a prespecified motion in 3D space. The controller introducing this ability is given in (3.30) through (3.33). The law in (3.30) maintains the desired altitude, and upon writing it into the dynamical equations in (3.15) through (3.17) with small-angle approximation, we get the desired Euler angle values as given in (3.31) through (3.32). Specifically, turns around z-axis are not desired, and we impose $\psi_r = 0$ as given in (3.33).

$$U_1 = \frac{M(F_z + g)}{\cos\phi\cos\theta} \tag{3.30}$$

$$\phi_r = -\arctan\left(\frac{F_y}{F_z + g}\right) \tag{3.31}$$

$$\theta_r = \arctan\left(\frac{F_x}{F_z + g}\right) \tag{3.32}$$

$$\psi_r = 0 \tag{3.33}$$

where

$F_z = -4\dot{z}_r - 4(z - z_r)$
$F_y = -\dot{y}_r - (y - y_r)$
$F_x = -\dot{x}_r - (x - x_r)$
$K_\phi, K_\theta = 4, K_\psi = 9$

The results for the feedback control are obtained via simulations, as shown in Figures 3.20 and 3.21. In the upper row of Figure 3.20, the trajectory followed in the 3D space is shown first. The desired path is followed at an admissible level of accuracy under the variation of battery voltage shown in the middle subplot. The practice of brushless motor–driven actuation scheme modulates the entire system, and a very noisy battery voltage is measured. The filtered battery voltage is adequate for predicting the necessary pwm level discussed also in the previous subsection. The bottom row of Figure 3.20 shows the errors in Cartesian space. Since the primary goal of the design is to maintain a desired altitude, performance in z-direction is comparably good from the others. The errors in x- and y-directions are also due to the imperfections introduced by the small-angle approximation. Nevertheless, comparing with the trajectories in the top left subplot, the magnitudes of the shown errors are acceptable too. In Figure 3.21, the attitude of the vehicle and the errors in the Euler angles are shown. Despite the large initial errors, the controller is able to drive the vehicle attitude to its desired values quickly, and the prescribed desired attitude angles are followed with a good precision.

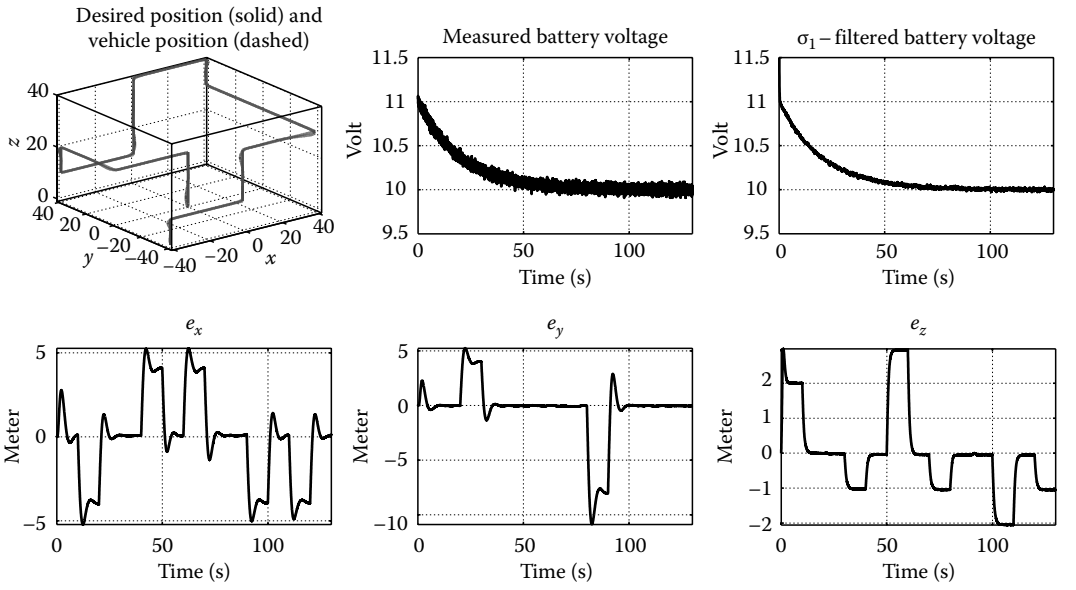

FIGURE 3.20 The trajectory followed in the Cartesian space.

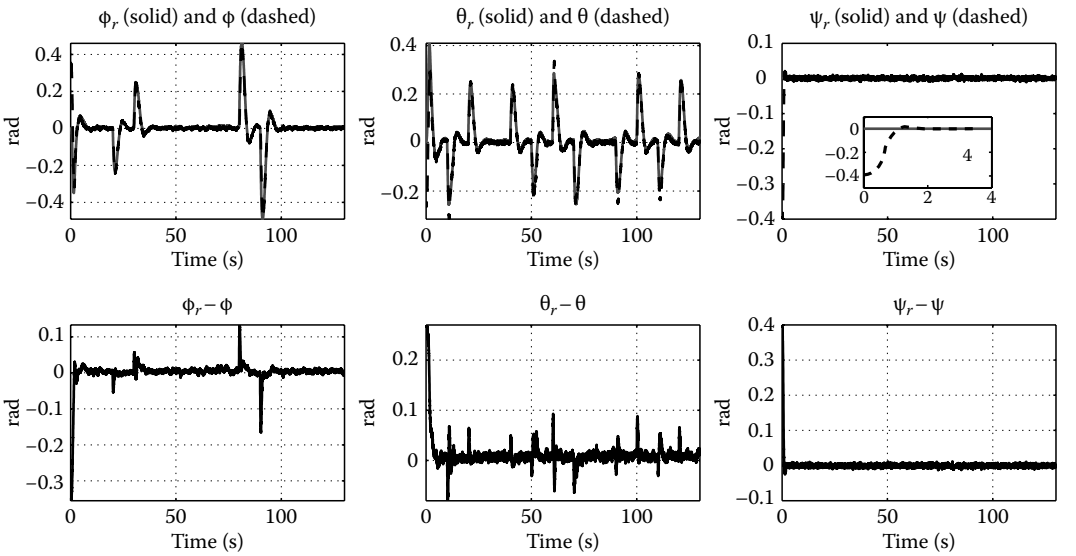

FIGURE 3.21 The attitude of the vehicle and the errors in the Euler angles.

Overall, the neural network structures in such a multivariable nonlinear feedback control example provide handshaking in between the actuation model and the dynamic model of the plant for alleviating the difficulties caused by the variations in the battery voltage; second, they provide linearization of the attitude dynamics to guide the vehicle correctly in the 3D space. In both positions, the neural networks function well enough as the application presented here is a good test bed for real-time data collection and numerical data-based approaches like neural network tools.

3.7 Concluding Remarks

Ever-increasing interest to neural networks in control systems led to the manufacturing of software tools that are of common use, hardware tools coping with computational intensity and a number of algorithms offering various types of training possibilities. Most applications of neural networks are involved with several number of variables changing in time and strong interdependencies among the variables. Such cases require a careful analysis of raw data as well as analytical tools to perform necessary manipulations. The main motivation of using neural networks in such applications is to solve the problem of constructing a function whose analytical form is missing but a set of data about it is available. With various types of architectural connectivity, activation schemes, and learning mechanisms, artificial neural networks are very powerful tools in every branch of engineering, and their importance in the discipline of control engineering is increasing parallel to the increase in the complexity being dealt with. In the future, the neural network models will continue to exist in control systems that request some degrees of intelligence to predict necessary maps.

Acknowledgments

This work is supported by TÜBİTAK, grant no. 107E137. The author gratefully acknowledges the facilities of the Unmanned Aerial Vehicles Laboratory of TOBB University of Economics and Technology.

References

[AB01] G. Arslan and T. Başar, Disturbance attenuating controller design for strict-feedback systems with structurally unknown dynamics, *Automatica*, 37, 1175–1188, 2001.

[APW91] P.J. Antsaklis, K.M. Passino, and S.J. Wang, An introduction to autonomous control systems, *IEEE Control Systems Magazine*, 11(4), 5–13, June 1991.

[ASL07] A.Y. Alanis, E.N. Sanchez, and A.G. Loukianov, Discrete-time adaptive backstepping nonlinear control via high-order neural networks, *IEEE Transactions on Neural Networks*, 18(4), 1185–1195, 2007.

[BFC09] M.K. Bugeja, S.G. Fabri, and L. Camilleri, Dual adaptive dynamic control of mobile robots using neural networks, *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 39(1), 129–141, 2009.

[C89] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals, and Systems*, 2, 303–314, 1989.

[CCBC07] G. Colin, Y. Chamaillard, G. Bloch, and G. Corde, Neural control of fast nonlinear systems—Application to a turbocharged SI engine with VCT, *IEEE Transactions on Neural Networks*, 18(4), 1101–1114, 2007.

[CF92] C.-C. Chiang and H.-C. Fu, A variant of second-order multilayer perceptron and its application to function approximations, *International Joint Conference on Neural Networks* (*IJCNN'92*), New York, vol. 3, pp. 887–892, June 7–11, 1992.

[CHI01] A.J. Calise, N. Hovakimyan, and M. Idan, Adaptive output feedback control of nonlinear systems using neural networks, *Automatica*, 37, 1201–1211, 2001.

[CN01] L. Chen and K.S. Narendra, Nonlinear adaptive control using neural networks and multiple models, *Automatica*, 37, 1245–1255, 2001.

[CSC06] M.C. Choy, D. Srinivasan, and R.L. Cheu, Neural networks for continuous online learning and control, *IEEE Transactions on Neural Networks*, 17(6), 1511–1532, 2006.

[E08] M.Ö. Efe, Novel neuronal activation functions for feedforward neural networks, *Neural Processing Letters*, 28(2), 63–79, 2008.

[EAK99] M.Ö. Efe, E. Abadoglu, and O. Kaynak, A novel analysis and design of a neural network assisted nonlinear controller for a bioreactor, *International Journal of Robust and Nonlinear Control*, 9(11), 799–815, 1999.

[F09] S. Ferrari, Multiobjective algebraic synthesis of neural control systems by implicit model following, *IEEE Transactions on Neural Networks*, 20(3), 406–419, 2009.

[F89] K. Funahashi, On the approximate realization of continuous mappings by neural networks, *Neural Networks*, 2, 183–192, 1989.

[FC07] Y. Fu and T. Chai, Nonlinear multivariable adaptive control using multiple models and neural networks, *Automatica*, 43, 1101–1110, 2007.

[GW02] S.S. Ge and J. Wang, Robust adaptive neural control for a class of perturbed strict feedback nonlinear systems, *IEEE Transactions on Neural Networks*, 13(6), 1409–1419, 2002.

[GW04] S.S. Ge and C. Wang, Adaptive neural control of uncertain MIMO nonlinear systems, *IEEE Transactions on Neural Networks*, 15(3), 674–692, 2004.

[GY01] J.Q. Gong and B. Yao, Neural network adaptive robust control of nonlinear systems in semi-strict feedback form, *Automatica*, 37, 1149–1160, 2001.

[H49] D.O. Hebb, *The Organization of Behaviour*, John Wiley & Sons, New York, 1949.

[H82] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences USA*, 79, 2554–2558, 1982.

[H94] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall PTR, Upper Saddle River, NJ, 1998.

[H99] M.A. Hussain, Review of the applications of neural networks in chemical process control—Simulation and online implementation, *Artificial Intelligence in Engineering*, 13, 55–68, 1999.

[HCL07] C.-F. Hsu, G.-M. Chen, and T.-T. Lee, Robust intelligent tracking control with PID-type learning algorithm, *Neurocomputing*, 71, 234–243, 2007.

[HGL05] F. Hong, S.S. Ge, and T.H. Lee, Practical adaptive neural control of nonlinear systems with unknown time delays, *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 35(4), 849–864, 2005.

[HM94] M.T. Hagan and M.B. Menhaj, Training feedforward networks with the Marquardt algorithm, *IEEE Transactions on Neural Networks*, 5(6), 989–993, November 1994.

[HN94] K. Hara and K. Nakayamma, Comparison of activation functions in multilayer neural network for pattern classification, *IEEE World Congress on Computational Intelligence*, Orlando, FL, vol. 5, pp. 2997–3002, June 27–July 2, 1994.

[HSV89] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, 2, 359–366, 1989.

[K06] V.S. Kodogiannis, Neuro-control of unmanned underwater vehicles, *International Journal of Systems Science*, 37(3),149–162, 2006.

[KA02] J. Kamruzzaman and S.M. Aziz, A note on activation function in multilayer feedforward learning, *Proceedings of the 2002 International Joint Conference on Neural Networks* (*IJCNN'02*), Honolulu, HI, vol. 1, pp. 519–523, May 12–17, 2002.

[KFS87] M. Kawato, K. Furukawa, and R. Suzuki, A hierarchical neural-network model for control and learning of voluntary movement, *Biological Cybernetics*, 57, 169–185, 1987.

[KK09] E.B. Kosmatopoulos and A. Kouvelas, Large scale nonlinear control system fine-tuning through learning, *IEEE Transactions on Neural Networks*, 20(6), 1009–1023, 2009.

[KTHD09] P. Kittisupakorn, P. Thitiyasoo, M.A. Hussain, and W. Daosud, Neural network based model predictive control for a steel pickling process, *Journal of Process Control*, 19, 579–590, 2009.

[LH07] W. Lan and J. Huang, Neural-network-based approximate output regulation of discrete-time nonlinear systems, *IEEE Transactions on Neural Networks*, 18(4), 1196–1208, 2007.

[LJY97] F.L. Lewis, S. Jagannathan, and A. Yeşildirek, Neural network control of robot arms and nonlinear systems, in O. Omidvar and D.L. Elliott, eds., *Neural Systems for Control*, pp. 161–211, Academic Press, San Diego, CA, 1997.

[LP07] K. Li and J.-X. Peng, Neural input selection—A fast model-based approach, *Neurocomputing*, 70, 762–769, 2007.

[LSS01] Y. Li, N. Sundararajan, and P. Saratchandran, Neuro-controller design for nonlinear fighter aircraft maneuver using fully tuned RBF networks, *Automatica*, 37, 1293–1301, 2001.

[M96] M.M. Polycarpou, Stable adaptive neural control scheme for nonlinear systems, *IEEE Transactions on Automatic Control*, 41(3), 447–451, 1996.

[MD05] J. Morimoto and K. Doya, Robust reinforcement learning, *Neural Computation*, 17(2), 335–359, 2005.

[MP43] W.S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nerveous activity, *Bulletin of Mathematical Biophysics*, 5, 115–133, 1943.

[N96] K.S. Narendra, Neural networks for control: Theory and practice, *Proceedings of the IEEE*, 84(10), 1385–1406, 1996.

[N97] G.W. Ng, *Application of Neural Networks to Adaptive Control of Nonlinear Systems*, Research Studies Press, Somerset, U.K., 1997.

[NP91] K.S. Narendra and K. Parthasarathy, Identification and control of dynamical systems using neural networks, *IEEE Transactions on Neural Networks*, 1, 4–27, 1990.

[P07] D. Prokhorov, Training recurrent neurocontrollers for real-time applications, *IEEE Transactions on Neural Networks*, 18(4), 1003–1015, 2007.

[P09] H.E. Psillakis, Sampled-data adaptive NN tracking control of uncertain nonlinear systems, *IEEE Transactions on Neural Networks*, 20(2), 336–355, 2009.

[PBR01] R. Padhi, S.N. Balakrishnan, and T. Randolph, Adaptive-critic based optimal neuro control synthesis for distributed parameter systems, *Automatica*, 37, 1223–1234, 2001.

[PF94] G.V. Puskorius and L.A. Feldkamp, Neurocontrol of nonlinear dynamical systems with Kalman filter trained recurrent networks, *IEEE Transactions on Neural Networks*, 5(2), 279–297, 1994.

[PKM09] J.-H. Park, S.-H. Kim, and C.-J. Moon, Adaptive neural control for strict-feedback nonlinear systems without backstepping, *IEEE Transactions on Neural Networks*, 20(7), 1204–1209, 2009.

[PL01] A.S. Poznyak and L. Ljung, On-line identification and adaptive trajectory tracking for nonlinear stochastic continuous time systems using differential neural networks, *Automatica*, 37, 1257–1268, 2001.

[PMB98] G.G. Parma, B.R. Menezes, and A.P. Braga, Sliding mode algorithm for training multilayer artificial neural networks, *Electronics Letters*, 34(1), 97–98, 1998.

[PRP07] A. Papadimitropoulos, G.A. Rovithakis, and T. Parisini, Fault detection in mechanical systems with friction phenomena: An online neural approximation approach, *IEEE Transactions on Neural Networks*, 18(4), 1067–1082, 2007.

[PS01] T. Parisini and S. Sacone, Stable hybrid control based on discrete-event automata and receding-horizon neural regulators, *Automatica*, 37, 1279–1292, 2001.

[PSY88] D. Psaltis, A. Sideris, and A.A. Yamamura, A multilayered neural network controller, *IEEE Control Systems Magazine*, pp. 17–21, April 1988.

[R01] G.A. Rovithakis, Stable adaptive neuro-control design via Lyapunov function derivative estimation, *Automatica*, 37, 1213–1221, 2001.

[R59] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 65, 386–408, 1959.

[RHW86] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, in D.E. Rumelhart and J.L. McClelland, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, MIT Press, Cambridge, MA, 1986.

[SG00] R.S. Sexton and J.N.D. Gupta, Comparative evaluation of genetic algorithm and backpropagation for training neural networks, *Information Sciences*, 129(1–4), 45–59, 2000.

[SG07] R. Sharma and M. Gopal, A robust Markov game controller for nonlinear systems, *Applied Soft Computing*, 7, 818–827, 2007.

[SL01] R.R. Selmic and F.L. Lewis, Neural net backlash compensation with Hebbian tuning using dynamic inversion, *Automatica*, 37, 1269–1277, 2001.

[SW01] J. Si and Y.-T. Wang, On-line learning control by association and reinforcement, *IEEE Transactions on Neural Networks*, 12(2), 264–276, 2001.

[U56] A.M. Uttley, A theory of the mechanism of learning based on the computation of conditional probabilities, *Proceedings of the First International Conference on Cybernetics*, Namur, Belgium, Gauthier-Villars, Paris, France, 1956.

[VL09] D. Vrabie and F. Lewis, Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems, *Neural Networks*, 22, 237–246, 2009.

[VSKJ07] J.B. Vance, A. Singh, B.C. Kaul, S. Jagannathan, and J.A. Drallmeier, Neural network controller development and implementation for spark ignition engines with high EGR levels, *IEEE Transactions on Neural Networks*, 18(4), 1083–1100, 2007.

[W03] R.-J. Wai, Tracking control based on neural network strategy for robot manipulator, *Neurocomputing*, 51, 425–445, 2003.

[WH01] J. Wang and J. Huang, Neural network enhanced output regulation in nonlinear systems, *Automatica*, 37, 1189–1200, 2001.

[WH60] B. Widrow and M.E. Hoff Jr, Adaptive switching circuits, IRE WESCON Convention Record, pp. 96–104, 1960.

[WW01] L.-X. Wang and F. Wan, Structured neural networks for constrained model predictive control, *Automatica*, 37, 1235–1243, 2001.

[WZD97] Y. Wu, M. Zhao and X. Ding, Beyond weights adaptation: A new neuron model with trainable activation function and its supervised learning, *International Conference on Neural Networks*, Houston, TX, vol. 2, pp. 1152–1157, June 9–12, 1997.

[YEK02] X. Yu, M.Ö. Efe, and O. Kaynak, A general backpropagation algorithm for feedforward neural networks learning, *IEEE Transactions on Neural Networks*, 13(1), 251–254, January 2002.

[YG03] D.L. Yu and J.B. Gomm, Implementation of neural network predictive control to a multivariable chemical reactor, *Control Engineering Practice*, 11, 1315–1323, 2003.

[YQLR07] J. Ye, J. Qiao, M.-A. Li, and X. Ruan, A tabu based neural network learning algorithm, *Neurocomputing*, 70, 875–882, 2007.

[ZW01] Y. Zhang and J. Wang, Recurrent neural networks for nonlinear output regulation, *Automatica*, 37, 1161–1173, 2001.